NAME

SDP_GET8, **SDP_GET16**, **SDP_GET32**, **SDP_GET64**, **SDP_GET128**, **SDP_GET_UUID128**, **SDP_PUT8**, **SDP_PUT16**, **SDP_PUT32**, **SDP_PUT64**, **SDP_PUT128**, **SDP_PUT_UUID128**, **sdp_open**, **sdp_open_local**, **sdp_close**, **sdp_error**, **sdp_get_lcaddr**, **sdp_search**, **sdp_attr2desc**, **sdp_uuid2desc** - Bluetooth SDP routines

LIBRARY

Bluetooth Service Discovery Protocol User Library (libsdp, -lsdp)

SYNOPSIS

#include <bluetooth.h>
#include <sdp.h>

SDP_GET8(*b*, *cp*);

SDP_GET16(*s*, *cp*);

SDP_GET32(*l*, *cp*);

SDP_GET64(*l*, *cp*);

SDP_GET128(*l*, *cp*);

SDP_GET_UUID128(*l*, *cp*);

SDP_PUT8(*b*, *cp*);

SDP_PUT16(*s*, *cp*);

SDP_PUT32(*l*, *cp*);

SDP_PUT64(*l*, *cp*);

SDP_PUT128(*l*, *cp*);

SDP_PUT_UUID128(*l*, *cp*);

*void **
sdp_open(*bdaddr_t const *l*, *bdaddr_t const *r*);

*void **
sdp_open_local(*char const *control*);

int32_t
sdp_close(*void *xs*);

int32_t
sdp_error(*void *xs*);

int32_t
sdp_get_lcaddr(*void *xs, bdaddr_t *l*);

int32_t
sdp_search(*void *xs, uint32_t plen, uint16_t const *pp, uint32_t alen, uint32_t const *ap, uint32_t vlen, sdp_attr_t *vp*);

*char const * const*
sdp_attr2desc(*uint16_t attr*);

*char const * const*
sdp_uuid2desc(*uint16_t uuid*);

int32_t
sdp_register_service(*void *xss, uint16_t uuid, bdaddr_p const bdaddr, uint8_t const *data, uint32_t datalen, uint32_t *handle*);

int32_t
sdp_unregister_service(*void *xss, uint32_t handle*);

int32_t
sdp_change_service(*void *xss, uint32_t handle, uint8_t const *data, uint32_t datalen*);

DESCRIPTION

The **SDP_GET8**(), **SDP_GET16**(), **SDP_GET32**(), **SDP_GET64**() and **SDP_GET128**() macros are used to get byte, short, long, long long and 128-bit integer from the buffer pointed by *cp* pointer. The pointer is automatically advanced.

The **SDP_PUT8**(), **SDP_PUT16**(), **SDP_PUT32**(), **SDP_PUT64**() and **SDP_PUT128**() macros are used to put byte, short, long, long long and 128-bit integer into the buffer pointed by *cp* pointer. The pointer is automatically advanced.

SDP_GET_UUID128() and **SDP_PUT_UUID128**() macros are used to get and put 128-bit UUID into the buffer pointed by *cp* pointer. The pointer is automatically advanced.

The **sdp_open**() and **sdp_open_local**() functions each return a pointer to an opaque object describing SDP session. The *l* argument passed to **sdp_open**() function should point to a source BD_ADDR. If source BD_ADDR is NULL then source address NG_HCI_BDADDR_ANY is used. The *r* argument passed to **sdp_open**() function should point to a non-NULL remote BD_ADDR. Remote BD_ADDR cannot be NG_HCI_BDADDR_ANY. The **sdp_open_local**() function takes path to the control socket and opens a connection to a local SDP server. If path to the control socket is NULL then default */var/run/sdp* path will be used.

The **sdp_close**() function terminates active SDP session and deletes SDP session object. The *xs* parameter should point to a valid SDP session object created with **sdp_open**() or **sdp_open_local**().

The **sdp_error**() function returns last error that is stored inside SDP session object. The *xs* parameter should point to a valid SDP session object created with **sdp_open**() or **sdp_open_local**(). The error value returned can be converted to a human readable message by calling strerror(3) function.

The **sdp_get_lcaddr**() function returns the SDP session actual source BD_ADDR. The *xs* parameter should point to a valid SDP session object created with **sdp_open**(). The *l* parameter should point to a buffer in which the value for the requested BD_ADDR is to be returned. **sdp_get_lcaddr**() function is useful if the current SDP session has been opened with the NG_HCI_BDADDR_ANY value passed as a source address.

The **sdp_search**() function is used to perform SDP Service Search Attribute Request. The *xs* parameter should point to a valid SDP session object created with **sdp_open**() or **sdp_open_local**(). The *pp* parameter is a Service Search Pattern - an array of one or more Service Class IDs. The maximum number of Service Class IDs in the array is 12. The *plen* parameter is the length of the Service Search pattern. The *ap* parameter is an Attribute ID Range List - an array of one or more SDP Attribute ID Range. Each attribute ID Range is encoded as a 32-bit unsigned integer data element, where the high order 16 bits are interpreted as the beginning attribute ID of the range and the low order 16 bits are interpreted as the ending attribute ID of the range. The attribute IDs contained in the Attribute ID Ranges List must be listed in ascending order without duplication of any attribute ID values. Note that all attributes may be requested by specifying a range of 0x0000-0xFFFF. The *alen* parameter is the length of the Attribute ID Ranges List. The **SDP_ATTR_RANGE**(*lo*, *hi*) macro can be used to prepare Attribute ID Range. The *vp* parameter should be an array of *sdp_attr_t* structures. Each *sdp_attr_t* structure describes single SDP attribute and defined as follows:

```
struct sdp_attr {
        uint16_t        flags;
```

```
#define SDP_ATTR_OK           (0 << 0)
#define SDP_ATTR_INVALID      (1 << 0)
#define SDP_ATTR_TRUNCATED    (1 << 1)
        uint16_t      attr;  /* SDP_ATTR_xxx */
        uint32_t      vlen;  /* length of the value[] in bytes */
        uint8_t      *value; /* base pointer */
};
typedef struct sdp_attr       sdp_attr_t;
typedef struct sdp_attr *     sdp_attr_p;
```

The caller of the **sdp_search**() function is expected to prepare the array of *sdp_attr* structures and for each element of the array both *vlen* and *value* must be set. The **sdp_search**() function will fill each *sdp_attr_t* structure with attribute and value, i.e., it will set *flags*, *attr* and *vlen* fields. The actual value of the attribute will be copied into *value* buffer. Note: attributes are returned in the order they appear in the Service Search Attribute Response. SDP server could return several attributes for the same record. In this case the order of the attributes will be: all attributes for the first records, then all attributes for the second record etc.

The **sdp_attr2desc**() and **sdp_uuid2desc**() functions each take a numeric attribute ID/UUID value and convert it to a human readable description.

The **sdp_register_service**() function is used to register service with the local SDP server. The *xss* parameter should point to a valid SDP session object obtained from **sdp_open_local**(). The *uuid* parameter is a SDP Service Class ID for the service to be registered. The *bdaddr* parameter should point to a valid BD_ADDR. The service will be only advertised if request was received by the local device with *bdaddr*. If *bdaddr* is set to NG_HCI_BDADDR_ANY then the service will be advertised to any remote devices that queries for it. The *data* and *datalen* parameters specify data and size of the data for the service. Upon successful return **sdp_register_service**() will populate *handle* with the SDP record handle. This parameter is optional and can be set to NULL.

The **sdp_unregister_service**() function is used to unregister service with the local SDP server. The *xss* parameter should point to a valid SDP session object obtained from **sdp_open_local**(). The *handle* parameter should contain a valid SDP record handle of the service to be unregistered.

The **sdp_change_service**() function is used to change data associated with the existing service on the local SDP server. The *xss* parameter should point to a valid SDP session object obtained from **sdp_open_local**(). The *handle* parameter should contain a valid SDP record handle of the service to be changed. The *data* and *datalen* parameters specify data and size of the data for the service.

CAVEAT

When registering services with the local SDP server the application must keep the SDP session open. If SDP session is closed then the local SDP server will remove all services that were registered over the session. The application is allowed to change or unregister service if it was registered over the same session.

EXAMPLES

The following example shows how to get SDP_ATTR_PROTOCOL_DESCRIPTOR_LIST attribute for the SDP_SERVICE_CLASS_SERIAL_PORT service from the remote device.

```
bdaddr_t      remote;
uint8_t       buffer[1024];
void         *ss    = NULL;
uint16_t      serv  = SDP_SERVICE_CLASS_SERIAL_PORT;
uint32_t      attr  = SDP_ATTR_RANGE(
                        SDP_ATTR_PROTOCOL_DESCRIPTOR_LIST,
                        SDP_ATTR_PROTOCOL_DESCRIPTOR_LIST);
sdp_attr_t    proto = { SDP_ATTR_INVALID,0,sizeof(buffer),buffer };

/* Obtain/set remote BDADDR here */

if ((ss = sdp_open(NG_HCI_BDADDR_ANY, remote)) == NULL)
      /* exit ENOMEM */
if (sdp_error(ss) != 0)
      /* exit sdp_error(ss) */

if (sdp_search(ss, 1, &serv, 1, &attr, 1, &proto) != 0)
      /* exit sdp_error(ss) */

if (proto.flags != SDP_ATTR_OK)
      /* exit see proto.flags for details */

/* If we got here then we have attribute value in proto.value */
```

DIAGNOSTICS

Both **sdp_open**() and **sdp_open_local**() will return NULL if memory allocation for the new SDP session object fails. If the new SDP object was created then caller is still expected to call **sdp_error**() to check if there was connection error.

The **sdp_get_lcaddr**(), **sdp_search**(), **sdp_register_service**(), **sdp_unregister_service**(), and **sdp_change_service**() functions return non-zero value on error. The caller is expected to call **sdp_error**()

to find out more about error.

SEE ALSO

bluetooth(3), strerror(3), sdpcontrol(8), sdpd(8)

AUTHORS

Maksim Yevmenkin <*m_evmenkin@yahoo.com*>

BUGS

Most likely. Please report bugs if found.