

HS Sockets Library v1.0

User Manual

Date: 28/06/2010

V1.0

HS Sockets Library v1.0.....	1
User Manual.....	1
Date: 28/06/2010.....	1
V1.0.....	1
1 Introduction.....	2
2 API Specification.....	2
2.1 HsSockInit.....	2
2.2 HsSockCleanUp.....	3
2.3 HsSockTcpConnect.....	3
2.4 HsSockTcpListen.....	4
2.5 HsSockTcpSend.....	5
2.6 HsSockUdpOpen.....	6
2.7 HsSockUdpSendto.....	7
2.8 HsSockClose.....	7
2.9 HsSockGetHostIpByName.....	8
2.10 HsSockGetDomain.....	8
2.11 HsSockInetAddr.....	9
2.12 HsSockInetNtoa.....	9
2.13 HsSockGetLocalAddress.....	10
2.14 HsSockSetReadSize.....	10
2.15 HsSockSetReadEnabled.....	11
2.16 Socket Event Callback.....	12
3 Secure Sockets API specification.....	14
3.1 HsSockTcpSwitchToSecureMode.....	14
3.2 HsSockTcpCheckSecureHandshakeDone.....	15
3.3 HsSockTcpCheckPeerCertSimple.....	16

1 Introduction

HS Sockets is a C source code library for interfacing to Windows network sockets layer. HS Sockets includes full C source code and binaries for Windows desktop 32 bit operating systems (Windows XP, Vista, 7), for Windows Mobile OS and for Linux OS

HS Sockets provides a C/C++ application with a simple interface to network socket layer and hides the complexity of network programming.

HS Sockets allows a programmer to create:

- TCP server and client applications
- UDP applications
- Mixed TCP server, client and UDP applications

HS Sockets supports handling multiple network connections concurrently.

Hs Sockets supports encrypted communication using OpenSSL library. The use of OpenSSL with Hs Sockets is optional.

Hs Sockets C library operates asynchronously in a non-blocking mode.

2 API Specification

2.1 *HsSockInit*

Declaration:

```
extern int HsSockInit(hs_sock_init_t *init);
```

Summary:

This function is used to initialize HS Sockets library. This function should be called first before any other functions are called. This function should be called once at your application start-up.

Parameters:

hs_sock_init_t *init – pointer to initialization structure holding HS Sockets library configuration parameters, defined as follows:

```
typedef struct
{
    usr_get_buf_fn_t *getbuf;           // get receive buffer callback
    int                ssl_enabled;     // 1=Use SSL; 0=SSL not used
} hs_sock_init_t;
```

*getbuf – pointer to callback function that is called by HsSock library whenever it needs to get a buffer for receiving data from a network socket.

The callback function is defined as follows:

```
/* get buffer from user space to write data to when received from network for a socket */
typedef unsigned char *usr_get_buf_fn_t(long usr_handle, int size);
```

usr_handle – is a user defined data passed in to a call to HsSockTcpConnect, HsSockUdpOpen or returned in response to event HS_SOCKET_EV_GET_USER_CTX.

size – buffer size required in bytes.

int ssl_enabled – set to 0 if SSL is not used. Set to 1 if SSL will be used by HsSock

Returns:

- HS_SOCKET_RC_OK – success
- HS_SOCKET_RC_INITIALIZED – library already initialized
- HS_SOCKET_RC_PARAM – invalid parameter(s)
- HS_SOCKET_RC_SOCKETINIT_FAILED – underlying network socket layer initialization failed

2.2 HsSockCleanUp

Declaration:

```
void HsSockCleanUp(void);
```

Summary:

This function is used to clean-up and de-initialize HS Sockets library. Use this function before you exit your application

Parameters:

None

Returns:

None

2.3 HsSockTcpConnect

Declaration:

```
int HsSockTcpConnect(long usr_handle,           // user handle
                    unsigned char *dest_ip,     // remote ip address (null terminated)
                    unsigned short dest_port,   // remote port
                    usr_event_fn_t *event_fn,   // callback to indicate event to user
                    long *sock_handle);        // socket handle returned here
```

Summary:

This function is used to establish a TCP connection to a remote TCP server. This is a non-blocking function. In case of failure HsSockTcpConnect returns immediately with corresponding return code. In case of success HsSockTcpConnect also returns immediately and the calling application processes the events in event callback function (specified in *event_fn) to determine the result of TCP connection attempt. The events and parameters are described in 2.16

Parameters:

long usr_handle – user application handle. Set it to your application context handle. HS Sockets shall associate this handle with the new socket session. This handle shall be passed as a parameter unchanged in user event callback.

unsigned char *dest_ip – remote IP address to connect to. This is a pointer to null terminated ASCII string buffer. The string must be in a dotted IP address format, for example “192.168.1.1”

unsigned short dest_port – remote TCP port to connect to

usr_event_fn_t *event_fn – pointer to event callback function that is used by HS Sockets library to indicate network events to user application, such as TCP connection established, connection closed, data received. The callback function prototype, events and parameters are specified below in this manual.

long *sock_handle – pointer to long socket session handle that is returned here by HS Sockets library if HsSockTcpConnect returns with success.

Returns:

- HS SOCK_RC_OK or HS SOCK_PENDING – success, connection initiated. This does not mean that connection is established. Check for socket events in the socket event callback function to determine the completion of this operation. Event HS SOCK_EV_CONNECTED will indicate that TCP connection has been established. Events HS SOCK_EV_CLOSED and HS SOCK_EV_CONN_FAILED will indicate failure.
- HS SOCK_RC_NOT_INIT – library not initialized
- HS SOCK_RC_PARAM – invalid parameter(s)
- HS SOCK_RC_CREATE_FAILED – underlying socket layer failed to create a socket
- HS SOCK_RC_CONN_FAILED – underlying socket layer connect failed
- HS SOCK_RC_NO_MEM – out of memory

2.4 HsSockTcpListen

Declaration:

```
int HsSockTcpListen(  
    unsigned short port,  
    long user_handle,  
    usr_event_fn_t *event_fn,  
    long *listen_sock_han);
```

Summary:

This function is used to start listening for incoming TCP connections on specified port number. This is a non-blocking function. In case of failure HsSockTcpListen returns immediately with corresponding return code. In case of success HsSockTcpListen also returns immediately and the calling application processes the events in event callback function (specified in *event_fn) to accept or reject incoming connections. The events and parameters are described in 2.16

Parameters:

unsigned short port – port number to listen on

long user_handle - user application handle. HS Sockets shall associate this handle with the listening socket session.

usr_event_fn_t *event_fn – pointer to event callback function that is used by HS Sockets library to indicate network events to user application. The callback function prototype, events and parameters are specified later in this manual.

long *listen_sock_han - pointer to long – listening socket session handle that is returned here by HS Sockets library if HsSockTcpListen returns with success.

Returns:

- HS_SOCKET_RC_OK – success
- HS_SOCKET_RC_NOT_INIT – library not initialized
- HS_SOCKET_RC_PARAM – invalid parameter(s)
- HS_SOCKET_RC_CREATE_FAILED – underlying socket layer failed to create a socket
- HS_SOCKET_RC_FAIL – underlying socket layer failure, unspecified reason
- HS_SOCKET_RC_NO_MEM – out of memory

2.5 HsSockTcpSend

Declaration:

```
int HsSockTcpSend(long hssk_handle,      // socket layer handle
                  unsigned char *packet_buf, // buffer data
                  int length,           // buffer data length
                  int *txed_len);      // bytes actually transmitted
```

Summary:

This function is used to send data over an established TCP session. This is a non-blocking function and returns immediately. If underlying transmit buffers are full, the function returns with HS_SOCKET_RC_SEND_PENDING. The application must call the same function later to retry sending the buffer.

If HsSockTcpSend succeeds it does not necessarily mean that it has transmitted the full buffer (packet_buf) of length specified in length. HsSockTcpSend may transmit a part of the packet_buf. In that case the length of transmitted data returned in txed_len is less than the length specified in length parameter. The application must advance to the next byte address within the packet_buf and re-submit the rest of the data at a later time by calling HsSockTcpSend again.

Parameters:

long hssk_handle – socket session handle

unsigned char *packet_buf – pointer to buffer containing data to be sent

int length – size of data buffer to be sent in bytes

int *txed_len – pointer to integer variable that receives number of bytes actually sent on return from this function

Returns:

- HS_SOCKET_RC_OK – success, buffer sent, txed_len contains number of bytes sent which may be less than length parameter

- HS_SOCK_RC_NOT_INIT – library not initialized
- HS_SOCK_RC_PARAM – invalid parameter(s)
- HS_SOCK_RC_SEND_PENDING – underlying socket layer is busy or transmit buffers are full. The requested buffer cannot be sent at this time, call the same function again later to retry.
- HS_SOCK_RC_FAIL – underlying socket layer failure, buffer sending failed.

2.6 HsSockUdpOpen

Declaration:

```
int HsSockUdpOpen(long usr_handle,
                 unsigned short source_port,
                 usr_event_fn_t *event_fn,
                 long *sock_handle);
```

Summary:

This function is used to open a UDP socket session on specified port number. This is a non blocking function and returns immediately with corresponding error code. If the error code is success (HS_SOCK_RC_OK), the application must process events in event_fn callback to receive UDP data on the open UDP port and detect socket closed event. The callback function prototype, events and parameters are specified later in this manual.

Parameters:

long usr_handle – user application handle. Set it to your application context handle. HS Sockets shall associate this handle with the new socket session. This handle shall be passed as a parameter unchanged in user event callback.

unsigned short source_port – local port to bind the socket session to. The application would then be able to receive UDP data on this port. To have the underlying socket layer allocate an unused local port dynamically, set source_port to 0. The actual port used can then be retrieved by calling function HsSockGetLocalAddress.

usr_event_fn_t *event_fn – pointer to event callback function that is used by HS Sockets library to indicate network events to user application, such as UDP data received and session closed. The callback function prototype, events and parameters are specified later in this manual.

long *sock_handle – pointer to long socket session handle that is returned here by HS Sockets library if HsSockUdpOpen returns with success.

Returns:

- HS_SOCK_RC_OK – success, UDP session is open
- HS_SOCK_RC_NOT_INIT – library not initialized
- HS_SOCK_RC_PARAM – invalid parameter(s)
- HS_SOCK_RC_CREATE_FAILED – underlying socket layer failed to create socket
- HS_SOCK_RC_BIND_FAILED – underlying socket layer bind failed
- HS_SOCK_RC_NO_MEM – out of memory

2.7 HsSockUdpSendto

Declaration:

```
int HsSockUdpSendto(  
    long          hssk_handle, // socket layer handle  
    unsigned long dest_ip,     // remote end IP address  
    unsigned int  dest_port,   // remote UDP port number  
    unsigned char *packet_buf, // packet data  
    unsigned int  length);     // packet data length
```

Summary:

Use this function to send data over UDP socket session to remote UDP endpoint. This is a non-blocking function and returns immediately.

Parameters:

long hssk_handle – socket session handle

unsigned long dest_ip – remote IP address to send the data to

unsigned int dest_port – destination port of remote UDP peer to send the data to

unsigned char *packet_buf – pointer to buffer containing data to be sent

unsigned int length – size of data buffer to be sent in bytes

Returns:

- HS_SOCKET_RC_OK – success, buffer sent
- HS_SOCKET_RC_NOT_INIT – library not initialized
- HS_SOCKET_RC_PARAM – invalid parameter(s)

2.8 HsSockClose

Declaration:

```
int HsSockClose(long sock_handle);
```

Summary:

Use this function to close a socket session. For TCP sessions, Hs Sockets will attempt graceful connection close before closing the socket and de-allocating all resources. This is a non-blocking function and returns immediately. Yet the actual socket closing and resource de-allocation happens at a later time asynchronously after the function returns.

Parameters:

long sock_handle – socket session handle

Returns:

- HS_SOCKET_RC_OK – success, session closed

- HS_SOCKET_RC_NOT_INIT – library not initialized
- HS_SOCKET_RC_PARAM – invalid parameter(s)

2.9 HsSockGetHostIpByName

Declaration:

```
int HsSockGetHostIpByName(unsigned char *hostname, unsigned char *dest_ip);
```

Summary:

Use this function to retrieve (resolve) host name string to IP address string. This call may result in underlying socket layer doing a DNS lookup / query.

Parameters:

hostname – host name string to resolve

dest_ip – result IP address string shall be copied into this buffer.

Returns:

- HS_SOCKET_RC_OK – success, IP address resolved
- HS_SOCKET_RC_NOT_INIT – library not initialized
- HS_SOCKET_RC_PARAM – invalid parameter(s)
- HS_SOCKET_RC_FAIL – lower layer failed to resolve hostname to IP address

2.10 HsSockGetDomain

Declaration:

```
int HsSockGetDomain(unsigned char *domain, int length);
```

Summary:

Use this function to obtain local peer hostname string, also known as local domain name.

Parameters:

unsigned char *domain – buffer to copy the obtained domain name into

int length – maximum size of buffer pointed to by *domain.

Returns:

- HS_SOCKET_RC_OK – success, local domain name retrieved
- HS_SOCKET_RC_NOT_INIT – library not initialized

- HS_SOCK_RC_PARAM – invalid parameter(s)
- HS_SOCK_RC_FAIL – lower layer failed to retrieve local domain name

2.11 HsSocketAddr

Declaration:

unsigned long HsSocketAddr(unsigned char *ip_addr);

Summary:

Use this function to convert a given IP address string into an unsigned long.

Parameters:

unsigned char *ip_addr – pointer to null terminated IP address string

Returns:

unsigned long 32 bit integer representing the IP address

2.12 HsSocketNtoa

Declaration:

void HsSocketNtoa(unsigned long ipl, unsigned char *ip_addr);

Summary:

Converts a given unsigned long ip address value into an IP address string

Parameters:

unsigned long ipl – 32 bit integer IP address

unsigned char *ip_addr – pointer to buffer to receive the result IP address string after conversion from 32 bit integer.

Returns:

none

2.13 HsSockGetLocalAddress

Declaration:

```
int HsSockGetLocalAddress(  
    long sock_handle, unsigned long *local_ip, unsigned short *local_port);
```

Summary:

Retrieves local IP address and local port used on a socket from socket layer.

Parameters:

sock_handle – HS Sockets session handle

unsigned long *local_ip – pointer to buffer to receive the local IP address string used by the session.

unsigned short *local_port pointer to unsigned short variable to receive the local port number used by the session

Returns:

- HS_SOCKET_RC_OK – success, local IP address and local port retrieved
- HS_SOCKET_RC_NOT_INIT – library not initialized
- HS_SOCKET_RC_PARAM – invalid parameter(s)
- HS_SOCKET_RC_FAIL – lower layer failed to retrieve local IP address and port

2.14 HsSockSetReadSize

Declaration:

```
int HsSockSetReadSize(long hssk_handle, int readsize);
```

Summary:

Sets the read block size. This value determines how much data the library reads from a socket at a time when there is data available on a socket. By default, HS Sockets library uses a value of 2048 bytes. If necessary an application can change the read block size by calling this function.

Parameters:

sock_handle – HS Sockets session handle

int readsize – number of bytes to read from a socket every time when there is data available on a socket, set to 0 to use the default value of 2048

Returns:

- HS_SOCKET_RC_OK – success, the read block size has been set
- HS_SOCKET_RC_NOT_INIT – library not initialized
- HS_SOCKET_RC_PARAM – invalid parameter(s)

2.15HsSockSetReadEnabled

Declaration:

```
int HsSockSetReadEnabled(long sock_han, int enabled);
```

Summary:

Enables or disables HS Sockets library to read the data on a socket session. If reading is disabled, HS Sockets does not call the lower layer socket read function and does not generate HS_SOCK_EV_DATA events. Thus the received data becomes buffered in the underlying socket layer driver. This feature can be useful for application level flow control.

By default reading is always enabled. To disable reading, the application must call this function. To re-enable reading, the application must call this function again with enabled set to 1

Parameters:

sock_handle – HS Sockets session handle

int enabled – Set to 0 to disable reading data from a socket. Set to 1 to (re)-enable reading data from a socket

Returns:

- HS_SOCK_RC_OK – success
- HS_SOCK_RC_NOT_INIT – library not initialized
- HS_SOCK_RC_PARAM – invalid parameter(s)

2.16 Socket Event Callback

The event callback function is defined as follows:

```
typedef long usr_event_fn_t(long usr_handle, int ev, long arg1, long arg2, long arg3);
```

usr_handle – user defined data passed in to a call to HsSockTcpConnect, HsSockUdpOpen or returned in response to event HS_SOCKET_EV_GET_USER_CTX or passed to HsSockTcpListen

ev – socket event.

arg1, arg2, arg3 – arguments are specific to each event

Event	Arguments	Return	Description
HS_SOCKET_EV_CONNECTED	arg1 = 0 arg2 = 0 arg3 = 0	0	TCP session established This event is generated when TCP session is established after a call to HsSockTcpConnect
HS_SOCKET_EV_GET_USER_CTX	arg1 = 0 arg2 = remote IP port number arg3 = pointer to remote IP address string	- return 0 if application does not want to accept a TCP connection. - return a non zero application context handle after allocating resources for the new session.	This event is generated when there is an incoming TCP connection request, after user has called HsSockTcpListen. In response to this event, the user can: - return 0 if application does not want to accept a TCP connection. - return a non zero application context handle after allocating resources for the new session.
HS_SOCKET_EV_ACCEPTED	arg1 = socket session handle arg2 = remote IP port number arg3 = pointer to remote IP address string	0	Incoming TCP connection accepted / established. The application must store socket session handle from argument 1 for further use (it is required for all operations on a socket session)
HS_SOCKET_EV_DATA	arg1 – number of bytes of data received arg2 – remote peer port number	0	Data received on a socket session. Data has been copied into the user buffer which has been supplied to library via get buffer callback

	arg3 = remote peer IP address (32 bit integer)		function. The length of data received is passed in arg1. This event is generated for both UDP and TCP sessions.
HS_SOCK_EV_CLOSED	arg1 = 0 arg2 = 0 arg3 = 0	0	Socket session has been closed by network or by local underlying socket layer
HS_SOCK_EV_CONN_FAILED	arg1 = 0 arg2 = 0 arg3 = 0	0	TCP connection attempt failed and TCP socket session closed. The actions taken should be the same as for event HS_SOCK_EV_CLOSED

3 Secure Sockets API specification

HS Sockets library can be optionally used with OpenSSL library to establish SSL encrypted client and server connections. This section describes SSL related API

3.1 *HsSockTcpSwitchToSecureMode*

Declaration:

```
int HsSockTcpSwitchToSecureMode(  
    long hssk_handle, hs_sock_ssl_param_t *pSslParam);
```

Summary:

Use this function to negotiate SSL connection over an established TCP session. Call this function on processing HS_SOCKET_EV_CONNECTED or HS_SOCKET_EV_ACCEPTED events to switch to secure communication on a given socket session.

After this function succeeds, any data exchanged via normal API (HsSockTcpSend and HS_SOCKET_EV_DATA) will be SSL encrypted

Parameters:

sock_handle – HS Sockets session handle

hs_sock_ssl_param_t *pSslParam – SSL parameters, defined as follows:

```
int          ssl_meth;           // SSL method  
int          is_server;         // is SSL server?  
unsigned char *pCaFile;         // certificate authority list filename  
unsigned char *pCertFile;       // server certificate file  
unsigned char *pKeyFile;        // private key file  
unsigned char *passw;           // private key password  
int          verify_cert;       // Verify server certificate
```

ssl_meth – SSL method, one of the following:

```
HSSOCK_SSLv2   - use SSLv2 protocol  
HSSOCK_SSLv3   - use SSLv3 protocol  
HSSOCK_TLSv1   - use TLSv1 protocol  
HSSOCK_SSLv23  - use SSL 2,3 or TLS 1 protocol
```

is_server – set to 0 is connecting to SSL server, set to 1 if accepting a connection from SSL client

*pCaFile – if acting as SSL server, this is an optional certificate authority filename

*pCertFile – if acting as SSL server, this is the certificate filename, mandatory for SSL server

unsigned char *pKeyFile – if acting as SSL server, this is private key file, mandatory for SSL server

unsigned char *passw – if acting as SSL server, this is private key password, mandatory for SSL server

verify_cert – if acting as SSL client, this controls whether to validate server certificate or not, set to 1 to verify, set to 0 to not verify

Returns:

- HS_SOCK_RC_OK – success
- HS_SOCK_RC_NOT_INIT – library not initialized
- HS_SOCK_RC_PARAM – invalid parameter(s)
- HS_SOCK_RC_SSL_NOT_ENABLED – SSL support is not enabled
- HS_SOCK_RC_SSL_CTX – failed to create SSL context

3.2 HsSockTcpCheckSecureHandshakeDone

Declaration:

```
extern int HsSockTcpCheckSecureHandshakeDone(  
    long hssk_handle,           // socket handle  
    int *is_done,              // returns if SSL negotiation complete or not  
    int *is_encryption_on,     // returns true if encryption has been negotiated  
    unsigned char *pCipherDesc); // returns descriptive string of the used cipher
```

Summary:

This is an optional function that can be used to determine if SSL handshake has completed. It also returns if encryption has been negotiated during SSL handshake and returns the negotiate cipher description string.

Parameters:

sock_handle – HS Sockets session handle

*is_done – 0, SSL not yet negotiated, 1=SSL negotiation complete

*is_encryption_on – 0 = encryption not negotiated, 1=encryption negotiated

*pCipherDesc – if is_encryption_on – returns cipher description string.

Returns:

- HS_SOCK_RC_OK – success
- HS_SOCK_RC_NOT_INIT – library not initialized
- HS_SOCK_RC_PARAM – invalid parameter(s)
- HS_SOCK_RC_SSL_NOT_ENABLED – SSL support is not enabled

3.3 HsSockTcpCheckPeerCertSimple

Declaration:

```
int HsSockTcpCheckPeerCertSimple(long hssk_handle, unsigned char *pHostName,  
    unsigned char *errstr)
```

Summary:

This is an optional function that can be used to validate the server SSL certificate after SSL connection has been negotiated.

The server hostname passed to the function is validated / compared against the server name contained in the server certificate.

Parameters:

sock_handle – HS Sockets session handle

*pHostName – the server hostname null terminated string

*errstr – pointer to buffer to receive the error string as a result of validation

Returns:

TRUE (1) – peer certificate validation passed successfully, the hostname is the same as contained in the peer certificate

FALSE (0) - peer certificate validation failed