

HS X.25 v1.3.13 User Manual

Revision: 2.4

Date: 22 December 2011

| | |
|--|-----------|
| HS X.25 V1.3.13 USER MANUAL..... | 1 |
| REVISION: 2.4..... | 1 |
| DATE: 22 DECEMBER 2011..... | 1 |
| 1 INTRODUCTION..... | 2 |
| 2 HS X.25 API..... | 3 |
| 2.1 HsX25INIT..... | 3 |
| 2.2 HsX25CONNECT..... | 7 |
| 2.3 HsX25LISTEN..... | 8 |
| 2.4 HsX25CLEAR..... | 9 |
| 2.5 HsX25DATA..... | 9 |
| 2.6 HsX25DATAEXP..... | 10 |
| 2.7 HsX25TICK..... | 10 |
| 2.8 HsX25STUTDOWN..... | 11 |
| 2.9 HsX25GETSTATS..... | 11 |
| 2.10 HsX25GETLINKSTATS..... | 12 |
| 2.11 HsX25RNR..... | 12 |
| 2.12 HsX25DECODEPKT..... | 13 |
| 2.13 HsX25GETCALLLIST..... | 14 |
| 2.14 HsX25ENABLETRACING..... | 14 |
| 2.15 XOT ROUTING TABLE FOR OUTGOING CALLS..... | 15 |
| 3 SUPPORT FOR PERMATENT VIRTUAL CIRCUITS (PVC)..... | 17 |

1 Introduction

HS X.25 is a software library in C (supplied with full source code) which implements ITU-T recommendation X.25 - Interface between Data Terminal Equipment (DTE) and Data Circuit-terminating Equipment (DCE) for terminals operating in the packet mode and connected to public data networks by dedicated circuit.

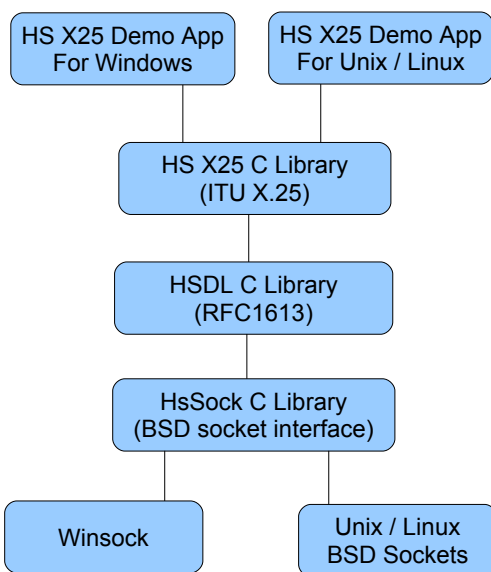
All source code components included in HS X.25 library support the following target operating systems:

- Windows (XP, Vista, 7 and server Oses)
- Linux, Solaris, HP-UX and other Unix like Oses

Individual source code components can also be ported to run in an embedded environment.

HS X.25 operates over XOT in two modes: RFC1613 Cisco Systems X.25 over TCP (XOT) or Proprietary XOT mode. XOT is part of HsDL sub-module included with the HS X.25 package.

Architecture and list of supported features:



HS X.25 internally at a lower layer interfaces to HsDL library. HsDL library is Data Link layer abstraction. When HS X.25 is ported into environment with LAPB or LAPD as data link layer, only HsDL module needs to change.

HsDL links directly to HsSock module (also included in this package) - Winsock interface component that provides reliable TCP transport services similar to LAPB / LAPD / HDLC.

HS X.25 as provided to customers may be used immediately in X.25 over TCP (XOT) solution or it may be used with traditional LAPB or LAPD in which case only HsDL will need to be modified.

When user application initialises HS X.25 library, it provides interface callbacks for the services used by HS X.25 protocol module: timer management, and event callbacks. The application then calls HS X.25 functions to establish virtual circuits, send and receive data, enforce flow control and clear calls.

| | |
|-------------------------------|--|
| DCE Operation | Yes |
| DTE Operation | Yes |
| RFC1613 XOT | Yes |
| Maximum Number of VCs | 4095 |
| Facilities Support | Yes |
| Call User Data supported | Yes |
| SVC Support | Yes |
| PVC Support | Yes, over RFC1613 XOT |
| X.25 Version | ITU-T (formerly CCITT) 10/96 |
| Outgoing Calls | Yes |
| Incoming Calls | Yes |
| Incoming Call processing | Configurable between: Match on local DTE address or Accept All Calls |
| Packet Format | Basic format, modulo 8 |
| Packet Size | Configurable in range 128,256,512,1024 with 128 default |
| Window Sizes | Configurable from 1 to 7 with default of 2 |
| A Bit | Basic format only (non TOA/NPI) addresses supported |
| Q Bit procedure | Yes |
| M Bit procedure | Yes |
| D Bit procedure | Yes |
| Logical Channel Assignment | Configurable Outgoing and Incoming range |
| Timers and Counters Supported | T10 T11 T12 T13 T20 T20 Retry T21 T22 T23 T23 Retry |
| Interrupt packets | Yes |
| Trace Function Supported | Yes |
| Flow Control (RNR) | Yes |
| Per VC statistics | Yes |

2 HS X.25 API

2.1 HsX25Init

| extern int HsX25Init (hs_x25_init_t *init); | | |
|--|-------|---|
| DESCRIPTION | | |
| <i>Initializes HS X.25 Library - this function must be called first before any other functions are called. Init structure contains function pointers which must be initialised with function addresses in user space. HS X.25 module will call these functions when it needs to manage timers or feed events back to user</i> | | |
| PARAMETERS | | |
| Type | Name | Description |
| hs_x25_init_t | *init | Pointer to initialisation structure: see next section for description |
| RETURNS | | |
| <ul style="list-style-type: none"> • <i>HS_X25_RC_OK</i> – success. HS X.25 initialised. • <i>HS_X25_RC_ALR_INIT</i> – HS X.25 library is already initialized • <i>HS_X25_RC_INV_PAR</i> – invalid parameters • <i>HS_X25_RC_INV_PKT_SIZE</i> – invalid packet size • <i>HS_X25_RC_INV_WSIZE</i> – invalid window size • <i>HS_X25_RC_DL_INIT_FAIL</i> – Data Link Layer initialisation failed | | |

2.1.1 Definition of hs_x25_init_t structure

| Function pointer | Prototype and Description |
|------------------|--|
| hs_x25_api_t api | <p>This structure contains X.25 API to user function pointers (these must be initialised by user to point to functions in user code):</p> <ul style="list-style-type: none"> • hs_x25_start_timer_t *start_timer – X.25 calls this function to start a timer. <ul style="list-style-type: none"> ○ Prototype: long hs_x25_start_timer_t(long handle, unsigned long secs, hs_x25_timer_callback_t *callback, int tid); ○ Parameters: handle – HS X.25 module handle secs – number of seconds to timeout after callback – pointer to function in X.25 that the user code calls when this timer expires. It has prototype: <i>void hs_x25_timer_callback_t(long handle);</i> where handle is X.25 module handle tid – timer id ○ Returns: timer handle • hs_x25_stop_timer_t *stop_timer – X.25 calls this function to stop a timer. <ul style="list-style-type: none"> ○ Prototype: void hs_x25_stop_timer_t(long handle, int tid); ○ Parameters: handle – timer handle tid – timer id • hs_x25_event_callback_t *event_cb – X.25 calls this function to feed events back to user application. This function and all events are fully specified in the following section (2.1.2 HS X.25 Event Callback) • logging_enabled – 1 = internal event logging is enabled; 0=internal event logging is disabled. Default setting is 0; Internal event reporting into application supplied event log can be enabled for debugging purposes • evlog_cb – if logging_enabled=1, this is a function pointer of event log function in user code. |

| | |
|----------------------------|---|
| | <p>HS X.25 would call this function to log an event.</p> <ul style="list-style-type: none"> o Prototype: void hs_x25_log_event_fn_t(unsigned char *str); |
| int opt | <p>X.25 protocol startup options:</p> <p>HS_X25_OPT_NORMAL =0 – normal operation HS_X25_OPT_L2_LOOP =1 – operation in loopback mode between 2 local level 2 links.</p> |
| hs_trace_cb_t *trc_fn; | <p>Trace function pointer in user code If tracing_enabled is 1, this function shall be called by HS X.25 for every X.25 packet passing through X.25 stack to present raw X.25 packets to application for further decoding, logging and / or display.</p> <ul style="list-style-type: none"> • Prototype: void hs_trace_cb_t(int is_rx, unsigned char *buf, int len, long arg); • Parameters: <ul style="list-style-type: none"> o Is_rx – indicates packet direction 1=received; 0=transmitted o Buf – raw X.25 packet data o Len - length of raw data o Arg – currently unused, reserved for future extension |
| int tracing_enabled; | Enable or Disable trace function: 1=enable; 0=disable |
| int is_rfc1613; | Specifies if RFC1613 Cisco mode XOT is to be used at Data Link layer 1 – Use RFC1613 Cisco Mode XOT 0 – Use proprietary mode XOT |
| int is_server; | <p>if proprietary XOT is used (not RFC1613), then this parameter configures data link module (HsDI) to operate as a TCP client (is_server=0) or TCP server (is_server=1).</p> <p>if RFC1613 mode is used, this parameter is ignored</p> |
| unsigned char *peer_ip; | <p>If parameter is_rfc1613 is set, peer_ip is a null terminated string containing remote IP address of XOT peer where outgoing connections are made to. The IP address string is in a dotted format, for example “192.168.1.2”.</p> <p>HS X.25 can make calls over XOT using two methods. The default mode of operation is that all outgoing calls using RFC1613 XOT go to the same destination IP address, but using separate TCP sessions. In this mode parameter *peer_ip identifies remote IP address for all X.25 calls.</p> <p>The second method is with XOT routing table. If this mode of operation is enabled with a call to API function HsX25RouteTableEnable, parameter *peer_ip is ignored and XOT routes outgoing calls by looking up destination IP address by X.25 called NUA for each call. Each route consisting of destination IP address and remote NUA, must be added individually with function calls to HsX25RouteAdd.</p> <p>In proprietary XOT mode this parameter is unused</p> |
| int is_dce; | Specifies DCE / DTE role of HS X.25. 1 - DCE 0 – DTE |
| int pktsize | Maximum default size of data payload of X.25 DATA packets. Valid values: 0 – use internal default (128), 128, 256, 512, 1024 |
| int wsize; | Default window size. Valid values: 0 – use internal default (2), 1, 2, 3, 4, 5, 6, 7. |
| int hi_incoming_lcn; | If RFC1613 XOT mode is used, this is highest logical channel number (LCN) HS X.25 will use for incoming calls. Valid values from 1 to 4095 |
| int lo_incoming_lcn; | If RFC1613 XOT mode is used, this is lowest logical channel number (LCN) HS X.25 will use for incoming calls. Valid values from 1 to 4095 |
| int support_dbit; | 0 – D bit procedure not supported, 1 – D bit procedure supported |
| void *hwnd | Handle of main application window which receives all Windows messages. |
| char local_ip[16]; | local IP address to use (Applies to XOT) If local_ip[0] == 0, default local IP address is used Cisco mode XOT endpoint, (RFC1613) will bind to port 1998 and this IP address, The IP address is specified as a dotted IP ascii null terminated string, such as “192.168.1.1” |

2.1.2 HS X.25 Event Callback

This function is called by HS X.25 to pass events about the state of calls virtual circuits, states of links, transmission and reception of data and other events.

2.1.2.1 Prototype

```

/*
 * HS X.25 event callback function
 */
typedef void hs_x25_event_callback_t
(
    // these 2 parameters below (l2addr, port) identify X.25 link
    long                l2addr, // in case data link XOT, IP address
    unsigned short int  port,   // in case data link XOT, IP port

    unsigned short int  lci,    // VC identifier (channel number 0 to 4095)
    int                 ev,     // event id
    long                arg1,   // first argument
    long                arg2    // second argument
);

```

2.1.2.2 Parameters

| Parameter | Description |
|-------------------------|--|
| long l2addr | 32 bit integer identifying data link, together with port. This is set to IP address. Data links are uniquely identified by IP address and port |
| unsigned short int port | 16 bit integer identifying datalink together with l2addr. This is set to IP port |
| unsigned short int lci | Logical channel identifier 0 to 4095 |
| int ev | Event id. All events are specified in section the next section |
| long arg1 | Event argument 1 |
| long arg2 | Event argument 2 |

2.1.2.3 Events

| Event | Arg1 and Arg2 | Remarks |
|-------------------------------|---|---|
| HS_X25_EV_CONN_FAIL_PKTCLR | Arg1=0; arg2=0 | X.25 call failed, packet layer not ready |
| HS_X25_EV_PKTCLR_DOWN | Arg1=0; arg2=0 | VC cleared: Packet layer down as result of physical layer down. This event is generated for every connected VC on the affected X.25 link |
| HS_X25_EV_PKTCLR_RESTARTED | Arg1=0; arg2=0 | VC cleared: Packet layer restarted (result of X.25 RESTART packet received from the line). This event is generated for every connected VC on the affected X.25 link |
| HS_X25_EV_CALL_CONNECTED | Arg1=0; arg2 = points to conn structure (hs_x25_conn_t) which holds X.25 called and calling addresses | X.25 SVC connected. |
| HS_X25_EV_CALL_CLEAR_COMPLETE | Arg1=0; arg2=0 | X.25 SVC clearing completed |
| HS_X25_EV_CALL_CLEARED | Arg1=0; arg2=0 | X.25 SVC cleared locally |
| HS_X25_EV_DATA | Arg1= pointer to hs_x25data_t structure, which is defined as follows: unsigned char *bufp – pointer to data buffer received | X.25 data sequence received. X.25 data sequence does not always correspond to a single X.25 DATA packet. If data transmitted by remote end is larger than used packet size the data is sent by transmitter in several data packets with M |

| | | |
|-----------------------------------|---|---|
| | int len – length of data buffer arg2=0 | bit set in all packets but last. HS X.25 at the receive end will re-assemble complete data sequence before passing it to application with this event. |
| HS_X25_EV_DATA_EXP | arg1 – Pointer to start of data, arg2 – Integer length of data | X.25 Expedited data received |
| HS_X25_EV_EXP_DATA_TX_FAIL | Arg1=0; arg2=0 | Cannot send expedited data, previous exp data not acknowledged |
| HS_X25_EV_EXP_DATA_DELIVERED | Arg1=0; arg2=0 | X.25 Expedited data received |
| HS_X25_EV_CALL_CLEARED_T13 | Arg1=0; arg2=0 | X.25 SVC cleared - no answer to call indication |
| HS_X25_EV_BUF_TX_FAIL | Arg1=buffer pointer Arg2=0 | buffer transmission failed |
| HS_X25_EV_INCOMING_CALL_CONNECTED | Arg1=0; arg2 = points to conn structure (hs_x25_conn_t) which holds X.25 called and calling addresses | incoming call connected (HsX25Listen completed) <u>hs_x25_conn_t definition:</u> called – called X.25 address of the incoming call calling – calling X.25 address of the incoming call int dbit – D bit state of the incoming call 0- D bit clear, 1- D bit set |
| HS_X25_EV_LINK_RELEASED | Arg1=0; arg2=0 | X.25 link released. This event is generated when data link has released (meaning that either client or server TCP session has failed) In proprietary XOT mode, this event can be used by application to initiate data link re-establishment. |
| HS_X25_EV_LINK_ESTABLISHED | Arg1=0; arg2=0 | X.25 link established This event is generated when data link has connected and is ready to carry X.25 VCs |
| HS_X25_EV_PVC_CONNECTED | Arg1=0; arg2=pointer to PVC setup structure - hs_x25_pvc_setup_t | PVC Setup complete |
| HS_X25_EV_PVC_SETUP_FAILED | Arg1=0; arg2=0 | PVC Setup failed |
| HS_X25_EV_PVC_DISCONNECTED | Arg1=0; arg2=0 | PVC disconnected |

2.2 HsX25Connect

| Extern int HsX25Connect (hs_x25_conn_t *conn, int *lci); | | | |
|---|-------|---|---|
| DESCRIPTION | | | |
| <i>This function is called to initiate an outgoing X.25 SVC establishment</i> | | | |
| PARAMETERS | | | |
| Type | Name | Description | |
| hs_x25_conn_t | *conn | Pointer to connect structure as follows: | |
| | | Data members | Description |
| | | long l2_addr; | 4 octet level 2 address. <u>RFC1613 XOT mode:</u> - Set to unique identifier, for example 0xffff. Listens must also be submits on the same link identifier <u>Proprietary XOT mode:</u> - remote IP address |
| | | long arg; | <u>RFC1613 XOT mode:</u> - Not used <u>Proprietary XOT mode:</u> - Set to remote TCP port number |
| | | unsigned char called[HS_MAX_NUA+1]; | Called X.25 DTE address (null terminated) |
| | | unsigned char calling[HS_MAX_NUA+1]; | Calling X.25 DTE address (null terminated) |
| | | unsigned char facility_buf[HS_MAX_FAC]; | Optional X.25 facilities buffer to be included with the call request. Facilities must be encoded by application according to X.25 standard. |
| | | unsigned char facility_len; | Size of facility buffer |
| | | unsigned char call_user_data[HS_MAX_CUD]; | Call user data to be included with the call |
| | | int cud_len; | Length of call user data |
| | | int dbit; | D Bit setting for the call |
| <p>Example setting level 2 address and argument for Proprietary XOT mode: conn.l2_addr = (long)inet_addr("10.1.73.164"); conn.arg = 1261;</p> | | | |
| unsigned short int | *lci | Pointer to integer variable to receive selected logical channel number for outgoing virtual circuit | |
| RETURNS | | | |
| <ul style="list-style-type: none"> • <i>HS_X25_RC_OK</i> – success. Outgoing Call initiated. The user application is notified of call completion or call failure asynchronously via event callback depending on call procedure result • <i>HS_X25_RC_INV_PAR</i> – invalid parameters • <i>HS_X25_RC_TOO_LONG</i> – the sum of X.25 addresses, user facilities and call user data exceeds global maximum X.25 packet size • <i>HS_X25_RC_NOLINK</i> - could not allocate new Layer 2 link • <i>HS_X25_RC_NO_VC</i> - no free VC available • <i>HS_X25_RC_NOT_INIT</i> – HS X.25 Library not initialized | | | |

2.3 HsX25Listen

```
int HsX25Listen(hs_x25_listen_t *listen)
```

DESCRIPTION

This function is called to listen for an incoming X.25 call

PARAMETERS

| Type | Name | Description | | | | | | | | | | | | | | |
|------------------------------------|--|--|--------------|--|----------|---|-------------------|--|-----------------------------------|---|-----------------|---------------------------------|------------------------------------|--|------------------|-----------------------------------|
| hs_x25_listen_t | *listen | <p>Pointer to listen structure as follows:</p> <table border="1"> <tr> <td>Long l2_addr</td> <td> <p>RFC1613 XOT mode: Set to unique link identifier, for example 0xffff. Outgoing calls must also be done on this link identifier</p> <p>Proprietary XOT mode: set to local IP address which remote end will be connecting to.</p> <p>If layer 2 address not found, a new layer 2 link will be allocated.</p> </td> </tr> <tr> <td>Long arg</td> <td> <p>RFC1613 XOT mode: Not used</p> <p>Proprietary XOT mode: Set to local TCP port to listen on</p> </td> </tr> <tr> <td>int take_any_call</td> <td> <p>1- accept all incoming calls, regardless of X.25 called address,</p> <p>0 – check X.25 called and calling addresses of incoming call against calling and called parameters of hs_x25_listen_t structure</p> </td> </tr> <tr> <td>unsigned char called[HS_MAX_NUA];</td> <td>When processing incoming call, compare the called X.25 address of the incoming call packet with this NUA.</td> </tr> <tr> <td>int called_len;</td> <td>Number of called address digits</td> </tr> <tr> <td>unsigned char calling[HS_MAX_NUA];</td> <td>When processing incoming call, compare the calling X.25 address of the incoming call packet with this NUA.</td> </tr> <tr> <td>int calling_len;</td> <td>Number of calling address digits.</td> </tr> </table> | Long l2_addr | <p>RFC1613 XOT mode: Set to unique link identifier, for example 0xffff. Outgoing calls must also be done on this link identifier</p> <p>Proprietary XOT mode: set to local IP address which remote end will be connecting to.</p> <p>If layer 2 address not found, a new layer 2 link will be allocated.</p> | Long arg | <p>RFC1613 XOT mode: Not used</p> <p>Proprietary XOT mode: Set to local TCP port to listen on</p> | int take_any_call | <p>1- accept all incoming calls, regardless of X.25 called address,</p> <p>0 – check X.25 called and calling addresses of incoming call against calling and called parameters of hs_x25_listen_t structure</p> | unsigned char called[HS_MAX_NUA]; | When processing incoming call, compare the called X.25 address of the incoming call packet with this NUA. | int called_len; | Number of called address digits | unsigned char calling[HS_MAX_NUA]; | When processing incoming call, compare the calling X.25 address of the incoming call packet with this NUA. | int calling_len; | Number of calling address digits. |
| Long l2_addr | <p>RFC1613 XOT mode: Set to unique link identifier, for example 0xffff. Outgoing calls must also be done on this link identifier</p> <p>Proprietary XOT mode: set to local IP address which remote end will be connecting to.</p> <p>If layer 2 address not found, a new layer 2 link will be allocated.</p> | | | | | | | | | | | | | | | |
| Long arg | <p>RFC1613 XOT mode: Not used</p> <p>Proprietary XOT mode: Set to local TCP port to listen on</p> | | | | | | | | | | | | | | | |
| int take_any_call | <p>1- accept all incoming calls, regardless of X.25 called address,</p> <p>0 – check X.25 called and calling addresses of incoming call against calling and called parameters of hs_x25_listen_t structure</p> | | | | | | | | | | | | | | | |
| unsigned char called[HS_MAX_NUA]; | When processing incoming call, compare the called X.25 address of the incoming call packet with this NUA. | | | | | | | | | | | | | | | |
| int called_len; | Number of called address digits | | | | | | | | | | | | | | | |
| unsigned char calling[HS_MAX_NUA]; | When processing incoming call, compare the calling X.25 address of the incoming call packet with this NUA. | | | | | | | | | | | | | | | |
| int calling_len; | Number of calling address digits. | | | | | | | | | | | | | | | |

RETURNS

- *HS_X25_RC_OK- success. Listen request accepted. Incoming call establishment is notified asynchronously via event callback*
- *HS_X25_RC_INV_PAR – invalid parameters*
- *HS_X25_RC_DL_LIS_FAIL – Listen failed due to Data Link layer error*
- *HS_X25_RC_NOLINK – no free link available*
- *HS_X25_RC_NOT_INIT – HS X.25 Library not initialize*

2.4 HsX25Clear

| int HsX25Clear(long l2_addr, unsigned short int port, unsigned short int lci) | | |
|---|---------|---|
| DESCRIPTION | | |
| <i>This function is called to Clear an X.25 SVC</i> | | |
| PARAMETERS | | |
| Type | Name | Description |
| long | l2_addr | IP address of X.25 link (4 octets) |
| unsigned short int | port | IP port of X.25 link (2 octets) |
| unsigned short int | lci | Logical channel number of the VC to clear |
| RETURNS | | |
| <ul style="list-style-type: none"> • <i>HS_X25_RC_OK – success. Clearing procedure initiated</i> • <i>HS_X25_RC_NOT_INIT - X.25 Library not initialized</i> • <i>HS_X25_RC_NOLINK – X.25 link not found</i> • <i>HS_X25_RC_NO_VC – X.25 VC is not allocated</i> • <i>HS_X25_RC_INV_PAR – LCI is out of range</i> | | |

2.5 HsX25Data

| int HsX25Data(long l2_addr, unsigned short int port, unsigned short int lci, unsigned char *buf, int len, int qbit); | | |
|---|---------|---|
| DESCRIPTION | | |
| <i>This function is called to send data over established X.25 VC</i> | | |
| PARAMETERS | | |
| Type | Name | Description |
| long | l2_addr | IP address of X.25 link (4 octets) |
| unsigned short int | port | IP port of X.25 link (2 octets) |
| unsigned short int | lci | Logical channel number of the VC |
| unsigned char | *buf | Pointer to data buffer to send |
| int | len | Length of data to send |
| int | qbit | Q Bit setting to use for the DATA packets carrying the specified data buffer. |
| RETURNS | | |
| <ul style="list-style-type: none"> • <i>HS_X25_RC_OK – success, data queued by HS X.25 for sending over specified VC</i> • <i>HS_X25_INV_PAR – invalid parameters</i> • <i>HS_X25_BUF_TOO_BIG – buffer is too big</i> • <i>HS_X25_TXQ_FULL – transmit queue is full, try again later.</i> • <i>HS_X25_RC_NOT_INIT - X.25 Library not initialized</i> • <i>HS_X25_RC_NOLINK – X.25 link not found</i> • <i>HS_X25_RC_NO_VC – X.25 VC is not allocated</i> | | |

2.6 HsX25DataExp

```
extern int HsX25DataExp(
    long l2_addr,           // layer 2 address (IP address)
    unsigned short int port, // IP port
    unsigned short int lci, // VC channel number
    unsigned char *buf, int len); // buffer to send, length to send
```

DESCRIPTION

This function is called to send expedited data over established X.25 VC. This function sends X.25 interrupt packet. An interrupt packet bypasses normal flow control mechanisms. Data is sent regardless of the state of the transmit window and no protocol variables are updated.

PARAMETERS

| Type | Name | Description |
|--------------------|---------|---|
| long | l2_addr | IP address of X.25 link (4 octets) |
| unsigned short int | port | IP port of X.25 link (2 octets) |
| unsigned short int | lci | Logical channel number of the VC |
| unsigned char | *buf | Pointer to data buffer. The user must leave a space at the start of the buffer for X.25 header (recommended 10 bytes), example call to HsX25Data: #define HDR_SPACE 10 static unsigned char testbuf2[HDR_SPACE + 8]; HsX25DataExp(x25vc_han, &testbuf2[HDR_SPACE], 8); |
| int | len | Length of data to send |

RETURNS

- *HS_X25_RC_OK* – success. Expedited data request accepted.
- *HS_X25_RC_NOT_INIT* - X.25 Library not initialized
- *HS_X25_RC_NOLINK* – X.25 link not found
- *HS_X25_RC_NO_VC* – X.25 VC is not allocated
- *HS_X25_INV_PAR* – invalid parameters

2.7 HsX25Tick

```
int HsX25Tick(void)
```

DESCRIPTION

This function must be called by user code as often as possible (from main loop or based on a fast timer). This function drives internal HS X.25 timers and services transmit queue.

PARAMETERS

| Type | Name | Description |
|------|------|---------------|
| | | No parameters |

RETURNS

- *HS_X25_RC_OK* – success
- *HS_X25_RC_NOT_INIT* - X.25 Library not initialized

2.8 HsX25StutDown

| Extern void HsX25StutDown(hs_x25_shutdown_callback_fn_t *fn) | | |
|--|------|--|
| DESCRIPTION | | |
| <i>This function is used to close all X.25 links and release all resources used by HS X.25</i> | | |
| PARAMETERS | | |
| Type | Name | Description |
| hs_x25_shutdown_callback_fn_t void hs_x25_shutdown_callback_fn_t(void); | *fn | Function callback for HS X.25 to call for every cleared VC during shutdown procedure |
| RETURNS | | |
| <ul style="list-style-type: none"> • None | | |

2.9 HsX25GetStats

| void HsX25GetStats(long l2_addr, unsigned short int port, unsigned short int lci, hs_x25_stats_t *stats) | | | | | | | | | | | | | | | | | |
|--|----------|--|-----|-----|------------------------|---------|----|---------------------|---------|----|--------------------|----------|----------|-----------------------|-----|---|----------------|
| DESCRIPTION | | | | | | | | | | | | | | | | | |
| <i>This function is used to get X.25 VC statistics</i> | | | | | | | | | | | | | | | | | |
| PARAMETERS | | | | | | | | | | | | | | | | | |
| Type | Name | Description | | | | | | | | | | | | | | | |
| long | l2_addr | IP address of X.25 link (4 octets) | | | | | | | | | | | | | | | |
| unsigned short int | port | IP port of X.25 link (2 octets) | | | | | | | | | | | | | | | |
| unsigned short int | lci | Logical channel number of the VC | | | | | | | | | | | | | | | |
| hs_x25_stats_t | *stats | Pointer to stats structure where X.25 will copy statistics information hs_x25_stats_t format: <table border="1" style="margin-left: 20px;"> <tr> <td>Int</td> <td>lci</td> <td>Logical channel number</td> </tr> <tr> <td>tx_rx_t</td> <td>tx</td> <td>Transmit statistics</td> </tr> <tr> <td>tx_rx_t</td> <td>rx</td> <td>Receive statistics</td> </tr> <tr> <td>unsigned</td> <td>char *st</td> <td>Name of current state</td> </tr> <tr> <td>int</td> <td>w</td> <td>Current Window</td> </tr> </table> Tx_rx_t definition: long data – DATA packet count long rr; - RR packet count long rnr – RNR packet count long reset – RESET packet count long intr – INTERRUPT packet count | Int | lci | Logical channel number | tx_rx_t | tx | Transmit statistics | tx_rx_t | rx | Receive statistics | unsigned | char *st | Name of current state | int | w | Current Window |
| Int | lci | Logical channel number | | | | | | | | | | | | | | | |
| tx_rx_t | tx | Transmit statistics | | | | | | | | | | | | | | | |
| tx_rx_t | rx | Receive statistics | | | | | | | | | | | | | | | |
| unsigned | char *st | Name of current state | | | | | | | | | | | | | | | |
| int | w | Current Window | | | | | | | | | | | | | | | |
| RETURNS | | | | | | | | | | | | | | | | | |
| <ul style="list-style-type: none"> • None | | | | | | | | | | | | | | | | | |

2.10 HsX25GetLinkStats

| extern int HsX25GetLinkStats(link_stats_t *pStats); | | | |
|---|-------------------------|--|--|
| DESCRIPTION | | | |
| <i>This function is used to get X.25 Link statistics</i> | | | |
| PARAMETERS | | | |
| Type | Name | Description | |
| link_stats_t | *pStats | Pointer to stats structure where X.25 will copy statistics information | |
| | | link_stats_t format: | |
| | | long l2addr | Link IP address |
| | | unsigned short int port | Link IP port |
| | | int num_listens | Number of queued listen requests |
| | | long dl_handle | Lower layer handle |
| | | int dl_state | Data link connection state |
| | | char pkt_lyr_state_str[40] | Packet layer state name (for example "R0", "R1" etc) |
| int num_alloc_vc | Number of allocated VCs | | |
| RETURNS | | | |
| <ul style="list-style-type: none"> • <i>HS_X25_RC_OK</i> – success • <i>HS_X25_RC_INV_PAR</i> – invalid parameters specified • <i>HS_X25_RC_NOT_INIT</i> - X.25 Library not initialized • <i>HS_X25_RC_NOLINK</i> – X.25 link not found | | | |

2.11 HsX25Rnr

| extern int HsX25Rnr(long l2_addr, unsigned short int port, unsigned short int lci, int on) | | |
|--|---------|--|
| DESCRIPTION | | |
| <i>This function is used set flow control condition which shall result in X.25 stack sending RNR or RR. Flow control condition can be set to recover temporarily low receive buffer resources.</i> | | |
| PARAMETERS | | |
| Type | Name | Description |
| long | l2_addr | IP address of X.25 link (4 octets) |
| unsigned short int | port | IP port of X.25 link (2 octets) |
| unsigned short int | lci | Logical channel number of the VC |
| int | on | 1= set flow control condition 0= clear flow control condition |
| RETURNS | | |
| <ul style="list-style-type: none"> • <i>HS_X25_RC_OK</i> – success • <i>HS_X25_RC_INV_PAR</i> – invalid parameters specified • <i>HS_X25_RC_NOT_INIT</i> - X.25 Library not initialized • <i>HS_X25_RC_NOLINK</i> – X.25 link not found • <i>HS_X25_RC_NO_VC</i> – X.25 VC is not allocated | | |

2.12 HsX25DecodePkt

```
int HsX25DecodePkt(unsigned char *pkt, int len, int *ev, hs_x25_pkt_info_t *info, int need_name);
```

DESCRIPTION

This function may be used for debugging or diagnostics to decode the content of an X.25 packet

PARAMETERS

| Type | Name | Description | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----------------------------|---|----------------------|----------------------------|-----------------|------------------------|---------------------|---------------------------|--------------------|-----------------|-----------|----------------|------------|--------------------|------------------------|----------------|------------------|---------------|------------------|---------------|----------------|-------------------|----------------|-------------------|------------------|---------------|------------------|---------------|-----------------|-----------|----------------|----------|
| unsigned char | *pkt | Pointer to X.25 packet | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| int | len | Length of X.25 packet | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| int | *ev | Pointer to integer that will receive decoded PTI (packet type identifier), one of the following: <table border="1" data-bbox="667 638 1468 1048"> <tbody> <tr><td>HS_X25_CALL_ACCEPTED</td><td>Call accept</td></tr> <tr><td>HS_X25_CALL_REQ</td><td>Call request</td></tr> <tr><td>HS_X25_RESTART_REQ,</td><td>Restart request</td></tr> <tr><td>HS_X25_RESTART_CNF</td><td>Restart confirm</td></tr> <tr><td>HS_X25_RR</td><td>Receiver ready</td></tr> <tr><td>HS_X25_RNR</td><td>Receiver not ready</td></tr> <tr><td>HS_X25_DAT</td><td>DATA</td></tr> <tr><td>HS_X25_CLEAR_CNF</td><td>Clear confirm</td></tr> <tr><td>HS_X25_CLEAR_REQ</td><td>Clear request</td></tr> <tr><td>HS_X25_INT_CNF</td><td>Interrupt confirm</td></tr> <tr><td>HS_X25_INT_REQ</td><td>Interrupt request</td></tr> <tr><td>HS_X25_RESET_REQ</td><td>Reset request</td></tr> <tr><td>HS_X25_RESET_CNF</td><td>Reset confirm</td></tr> <tr><td>HS_X25_PVCSETUP</td><td>PVC Setup</td></tr> <tr><td>HS_X25_PVCKILL</td><td>PVC Kill</td></tr> </tbody> </table> | HS_X25_CALL_ACCEPTED | Call accept | HS_X25_CALL_REQ | Call request | HS_X25_RESTART_REQ, | Restart request | HS_X25_RESTART_CNF | Restart confirm | HS_X25_RR | Receiver ready | HS_X25_RNR | Receiver not ready | HS_X25_DAT | DATA | HS_X25_CLEAR_CNF | Clear confirm | HS_X25_CLEAR_REQ | Clear request | HS_X25_INT_CNF | Interrupt confirm | HS_X25_INT_REQ | Interrupt request | HS_X25_RESET_REQ | Reset request | HS_X25_RESET_CNF | Reset confirm | HS_X25_PVCSETUP | PVC Setup | HS_X25_PVCKILL | PVC Kill |
| HS_X25_CALL_ACCEPTED | Call accept | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| HS_X25_CALL_REQ | Call request | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| HS_X25_RESTART_REQ, | Restart request | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| HS_X25_RESTART_CNF | Restart confirm | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| HS_X25_RR | Receiver ready | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| HS_X25_RNR | Receiver not ready | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| HS_X25_DAT | DATA | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| HS_X25_CLEAR_CNF | Clear confirm | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| HS_X25_CLEAR_REQ | Clear request | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| HS_X25_INT_CNF | Interrupt confirm | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| HS_X25_INT_REQ | Interrupt request | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| HS_X25_RESET_REQ | Reset request | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| HS_X25_RESET_CNF | Reset confirm | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| HS_X25_PVCSETUP | PVC Setup | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| HS_X25_PVCKILL | PVC Kill | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| hs_x25_pkt_info_t | *info | Pointer to structure in user code that receives decoded packet information. <p>Structure definition:</p> <table border="1" data-bbox="667 1234 1468 1503"> <tbody> <tr><td>Int lci</td><td>Logical channel identifier</td></tr> <tr><td>Int pti</td><td>Packet type identifier</td></tr> <tr><td>unsigned char *pkt</td><td>Points to start of packet</td></tr> <tr><td>Int len</td><td>Packet length</td></tr> <tr><td>Int Pr</td><td>PR value</td></tr> <tr><td>Int Ps</td><td>PS value</td></tr> <tr><td>unsigned char *pktname</td><td>Name of packet</td></tr> </tbody> </table> | Int lci | Logical channel identifier | Int pti | Packet type identifier | unsigned char *pkt | Points to start of packet | Int len | Packet length | Int Pr | PR value | Int Ps | PS value | unsigned char *pktname | Name of packet | | | | | | | | | | | | | | | | |
| Int lci | Logical channel identifier | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Int pti | Packet type identifier | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| unsigned char *pkt | Points to start of packet | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Int len | Packet length | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Int Pr | PR value | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Int Ps | PS value | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| unsigned char *pktname | Name of packet | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| int | need_name | 1= fill in packet name 0= do not fill in packet name | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

RETURNS

- *HS_X25_RC_DECODE_SHORT* – invalid packet – too short
- *HS_X25_RC_DECODE_LONG* – invalid packet – too long
- *HS_X25_RC_DECODE_LCI_UNASSIGN* – packet on unassigned logical channel
- *HS_X25_RC_DECODE_UNSPEC* – decoded error, unspecified
- *HS_X25_RC_DECODE_INV_GFI* – invalid general format identifier
- *HS_X25_RC_DECODE_OK* – packet is good and is correctly decoded

2.13 HsX25GetCallList

| extern void HsX25GetCallList(call_list_fn_t *fn); | | |
|---|------|--|
| DESCRIPTION | | |
| <i>This function is used to retrieve a list of connected VCs.</i> | | |
| PARAMETERS | | |
| Type | Name | Description |
| call_list_fn_t | *fn | Function pointer HS X.25 shall call for every connected VC Definition: void call_list_fn_t(unsigned short int lci, hs_x25_conn_t *pConn); lci – channel number of the VC pConn points to hs_x25_conn_t structure and contains X.25 called and calling DTE addresses and link identification information (IP address and port number) |
| RETURNS | | |
| <ul style="list-style-type: none"> • none | | |

2.14 HsX25EnableTracing

| extern void HsX25EnableTracing(int enable); | | |
|--|--------|---------------------------------------|
| DESCRIPTION | | |
| <i>This function is used to enable or disable tracing function</i> | | |
| PARAMETERS | | |
| Type | Name | Description |
| int | enable | 1=enable tracing 0=disable tracing |
| RETURNS | | |
| <ul style="list-style-type: none"> • none | | |

2.15 XOT Routing Table for Outgoing Calls

HS X.25 can make calls over XOT using to methods. The default mode of operation is that all outgoing calls using RFC1613 XOT go to the same destination IP address, but using separate TCP sessions. In this mode parameter `*rfc1613_peer_ip` in the X.25 stack initialization structure `hs_x25_init_t` identifies remote IP address for all X.25 calls.

The second method is with XOT routing table. If this mode of operation is enabled with a call to API function `HsX25RouteTableEnable`, parameter `*rfc1613_peer_ip` is ignored and XOT routes outgoing calls by looking up destination IP address by X.25 called NUA for each call. Each route consisting of destination IP address and remote NUA, must be added individually with function calls to `HsX25RouteAdd`.

Routes can be added, modified and removed at any after HS X.25 Library was initialized with a call to `HsX25Init`.

2.15.1 HsX25RouteTableEnable

| void HsX25RouteTableEnable(int enable) | | |
|--|--------|---------------------|
| DESCRIPTION | | |
| <i>This function is used to enable XOT routing table for making outgoing calls. If XOT routing table is enabled parameter <code>*rfc1613_peer_ip</code> is ignored and XOT routes outgoing calls by looking up destination IP address by X.25 called NUA for each call. Each route consisting of destination IP address and remote NUA, must be added individually with function calls to <code>HsX25RouteAdd</code></i> | | |
| PARAMETERS | | |
| Type | Name | Description |
| int | enable | 1=enable; 0=disable |
| RETURNS | | |
| <ul style="list-style-type: none"> • <i>None</i> | | |

2.15.1 HsX25RouteAdd

| int HsX25RouteAdd(unsigned char *remote_nua, unsigned char *remote_ip, int *route_index); | | |
|---|--------------|---|
| DESCRIPTION | | |
| <i>This function is used to add an outgoing XOT route to XOT routing table. A route consists of destination X.25 NUA and remote IP address. This function may be called at any time. Maximum supported number of routes is 4095 in the current version of HS X25 Library.</i> | | |
| PARAMETERS | | |
| Type | Name | Description |
| unsigned char | *remote_nua | Pointer to null terminated string – X.25 called DTE address, also called remote NUA. Maximum length of string not including terminating zero is 15 digits. |
| unsigned char | *remote_ip | Pointer to null terminated string – remote IP address where the XOT will connect TCP session to before sending X.25 call request. Remote IP address is in dotted IP format. “n.n.n.n”. Maximum length of string not including zero terminating character is 15 bytes. |
| int | *route_index | Pointer to integer variable that will receive route index on completion of the function. This route index can then be used to modify or delete the route using functions <code>HsX25RouteSet</code> and <code>HsX25RouteDelete</code> |
| RETURNS | | |
| <ul style="list-style-type: none"> • <i>HS_X25_RC_FAIL</i> – adding route failed • <i>HS_X25_RC_OK</i> – success, route added | | |

2.15.2 HsX25RouteSet

| int HsX25RouteAdd(int route_index, unsigned char *remote_nua, unsigned char *remote_ip); | | |
|---|-------------|---|
| DESCRIPTION | | |
| <i>This function is used to modify an existing XOT route of XOT routing table.</i> | | |
| PARAMETERS | | |
| Type | Name | Description |
| int | route_index | Index of route to modify. Index is obtained with a call to HsX25RouteAdd |
| unsigned char | *remote_nua | Pointer to null terminated string – X.25 called DTE address, also called remote NUA. Maximum length of string not including terminating zero is 15 digits. |
| unsigned char | *remote_ip | Pointer to null terminated string – remote IP address where the XOT will connect TCP session to before sending X.25 call request. Remote IP address is in dotted IP format. “n.n.n.n”. Maximum length of string not including zero terminating character is 15 bytes. |
| RETURNS | | |
| <ul style="list-style-type: none"> • <i>HS_X25_RC_FAIL – changing the route failed</i> • <i>HS_X25_RC_OK – success, the route changed</i> | | |

2.15.1 HsX25RouteDelete

| int HsX25RouteDelete(int route_index); | | |
|---|-------------|---|
| DESCRIPTION | | |
| <i>This function is used to delete an existing XOT route from XOT routing table.</i> | | |
| PARAMETERS | | |
| Type | Name | Description |
| int | route_index | Index of route to delete. This index is obtained with a call to HsX25RouteAdd |
| RETURNS | | |
| <ul style="list-style-type: none"> • <i>HS_X25_RC_FAIL – deleting the route failed</i> • <i>HS_X25_RC_OK – success, route deleted</i> | | |

3 Support for permatent virtual circuits (PVC)

In the current version of HS X.25 PVCs are supported over XOT RFC1613 only

3.1.1 HsX25CreateLink

| <code>int HsX25CreateLink (hs_x25_link_cfg_t *pLinkCfg);</code> | | |
|--|-----------------------|---|
| DESCRIPTION | | |
| <p><i>This function creates an X.25 link and configures it.</i></p> <p><i>This is an optional function which may be called before calling HsX25Connect or HsX25Listen.</i></p> <p><i>This function must be called if the user intends to use PVC functionality of HS X.25 library</i></p> | | |
| PARAMETERS | | |
| Type | Name | Description |
| <code>hs_x25_link_cfg_t</code> | <code>pLinkCfg</code> | <p>Pointer to link configuration structure, defined as follows:</p> <pre>// link configuration typedef struct { long l2_addr; // unique link identifier unsigned short int arg; int num_pvcs; // number of VCs reserved for PVCs int pvc_base_lcn; // 1st LCN number reserved for PVCs } hs_x25_link_cfg_t;</pre> <p><code>l2addr</code> – link identifier, use <code>RFC1613_LINK_ADDR</code> constant <code>arg</code> – second part of link identifier, use <code>RFC1613_LINK_PORT</code></p> <p><code>num_pvcs</code> – number of VCs to be allocated for PVCs <code>pvc_base_lcn</code> – LCN of first PVC.</p> |
| RETURNS | | |
| <ul style="list-style-type: none"> • <i>HS_X25_RC_NOT_INIT</i> – HS X.25 library not initialized • <i>HS_X25_RC_INV_PAR</i> – invlaid parameters • <i>HS_X25_RC_LINK_EXISTS</i> – link already exists (lookup is done on link identifier) • <i>HS_X25_RC_NOLINK</i> – Failed to allocated new link • <i>HS_X25_RC_NO_VC</i> – failed to allocate PVC • <i>HS_X25_RC_OK</i> – success | | |

3.1.1 HsX25ConnectPvc

```
int HsX25ConnectPvc (hs_x25_pvc_setup_t *conn);
```

DESCRIPTION

*This function initiates PVC setup over XOT RFC1616
The remote side must be listening for PVC setup*

PARAMETERS

| Type | Name | Description |
|--------------------|------|--|
| hs_x25_pvc_setup_t | conn | <p>Pointer to PVC setup configuration structure, defined as follows:</p> <p>l2_addr - link identifier, use RFC1613_LINK_ADDR constant arg – second part of link identifier, use RFC1613_LINK_PORT</p> <p>local_lcn – local VC logical channel number remote_lcn - remote VC logical channel number</p> <p>local_name - local interface name (null terminated string, max 64 bytes) remote_name - remote interface name (null terminated string, max 64 bytes)</p> <p>int local_incoming_window - local incoming window size valid values are 1 to 7</p> <p>int local_outgoing_window - local outgoing window size valid values are 1 to 7</p> <p>int local_incoming_max_pktsize - local incoming max packet size Values from 4 to 10, see note below</p> <p>int local_outgoing_max_pktsize - local outgoing max packet size Values from 4 to 10, see note below</p> <p>The values for the packet sizes are encoded as follows: 4: 16 bytes 5: 32 bytes 6: 64 bytes 7: 128 bytes 8: 256 bytes 9: 512 bytes 10: 1024 bytes</p> |

RETURNS

- *HS_X25_RC_NOT_INIT* – HS X.25 library not initialized
- *HS_X25_RC_INV_PAR* – invlaid parameters
- *HS_X25_RC_NOLINK* – Failed to allocated new link
- *HS_X25_RC_NO_VC* – PVC not found
- *HS_X25_RC_OK* – success, PVC setup initiated

3.1.1 HsX25ListenPvc

```
int HsX25ListenPvc (hs_x25_pvc_setup_t *conn);
```

DESCRIPTION

This function initiates listening for PVC setup over XOT RFC1616

PARAMETERS

| Type | Name | Description |
|--------------------|------|---|
| hs_x25_pvc_setup_t | conn | <p>Pointer to PVC setup configuration structure, defined as follows:</p> <p>l2_addr - link identifier, use RFC1613_LINK_ADDR constant arg – second part of link identifier, use RFC1613_LINK_PORT</p> <p>local_lcn – local VC logical channel number remote_lcn - remote VC logical channel number</p> <p>local_name - local interface name (null terminated string max 64 bytes) remote_name - remote interface name (null terminated string, max 64 bytes)</p> <p>int local_incoming_window - local incoming window size valid values are 1 to 7</p> <p>int local_outgoing_window - local outgoing window size valid values are 1 to 7</p> <p>int local_incoming_max_pktsize - local incoming max packet size Values from 4 to 10, see note below</p> <p>int local_outgoing_max_pktsize - local outgoing max packet size Values from 4 to 10, see note below</p> <p>The values for the packet sizes are encoded as follows: 4: 16 bytes 5: 32 bytes 6: 64 bytes 7: 128 bytes 8: 256 bytes 9: 512 bytes 10: 1024 bytes</p> |

RETURNS

- *HS_X25_RC_NOT_INIT* – HS X.25 library not initialized
- *HS_X25_RC_INV_PAR* – invlaid parameters
- *HS_X25_RC_NOLINK* – Failed to allocated new link
- *HS_X25_RC_NO_VC* – PVC not found
- *HS_X25_RC_OK* – success, PVC setup initiated

3.1.1 HsX25DisconnectPvc

int HsX25DisconnectPvc (long l2_addr, unsigned short int port, unsigned short int lcn);

DESCRIPTION

This function disconnects an existing PVC

PARAMETERS

| Type | Name | Description |
|-------|---------|---|
| long | l2_addr | Link identifier, use RFC1613_LINK_ADDR constant |
| long | arg | second part of link identifier, use RFC1613_LINK_PORT |
| short | lcn | PVC logical channel number |

RETURNS

- *HS_X25_RC_NOT_INIT* – HS X.25 library not initialized
- *HS_X25_RC_INV_PAR* – invlaid parameters
- *HS_X25_RC_NOLINK* – Failed to allocated new link
- *HS_X25_RC_NOT_VC* – VC is not a PVC
- *HS_X25_RC_OK* – success