HEWLETT PACKARD

# HP 18321A X.25 Test Environment

for the HP 4954A Protocol Analyzer

Library Reference

## HP 4954A Protocol Analyzer

# HP 18321A X.25 Test Environment

## Library Reference

HEWLETT PACKARD

Manual Part Number: 18321-99502
Microfiche Part Number: 18321-99804

Printed in U.S.A. AUGUST 1989
E0889

## Notice

If your software application or hardware should fail, contact your local Hewlett-Packard Sales Office listed in the protocol analyzer operating manual.

# Contents

# Tables

# Printing History

New editions are complete revisions of the manual. Update packages are issued between editions. They contain additional and replacement pages to be merged into the manual by the customer. The dates on the title page change only when a new edition or a new update is published. No information is incorporated into a reprinting unless it appears as a prior update. The edition does not change when an update is incorporated.

Many product updates and fixes do not require manual changes and, conversely, manual corrections may be done without accompanying product changes. Therefore, do not expect a one-to-one correlation between product updates and manual updates.

| | |
|---|---|
| First Edition................................................................ | August 1989 |

# Syntax Conventions

The following symbols, abbreviations, and other conventions are used in this publication.

| Symbol | Definition |
|---|---|
| Setup Menu | A softkey. |
| Reset | A keyboard command entry. |
| CTRL U | A control character entry from the keyboard where both the CTRL key and an alphanumeric key are pressed at the same time. To enter CONTROL U press CTRL and U. |
| Shift softkey | A keyboard entry where both the Shift and a softkey are pressed at the same time to select an auxiliary softkey function. |
| FILENAME | Within menus or screens, a parameter that must be entered in the exact format shown. |
| **filename** | Within menus or screens, a user-defined parameter. |

**Warning** An operating procedure, practice, etc., which, if not correctly followed, could result in personal injury or loss of life.

**Caution** An operating procedure, practice, etc. which, if not strictly observed, could result in damage to, or destruction of, equipment or software.

**Note** Explanatory comments or supplementary instructions.

1

# Introducing X.25 Test Environment Libraries

This manual describes the functions in the X.25 Test Environment libraries. To use this material requires a knowledge of the C Language, the DataCommC Development and Run-Time environments, the X.25 protocol, and the HP 4954A Protocol Analyzer.

Publications that address these subjects include the following.

| Subject | Reference Publication |
|---|---|
| C Language | Kernighan, Brian W. and Ritchie, Dennis M., *The C Programming Language*, 1978, Prentice-Hall, Inc., Englewood Cliffs, NJ, HP Part Number 18320-99502 |
| Editing, Compiling, Linking, or Running DataCommC programs | *HP 18320A DataCommC Programming Language User's Guide*, part number 18320-99501, Hewlett-Packard CTD, February 1989. |
| DataCommC functions | *HP 18320A DataCommC Programming Language Library Reference*, part number 18320-99503, Hewlett-Packard CTD, February 1989. |
| Editing or Running X.25 Test Environment programs | *HP 18321A X.25 Test Environment User's Guide*, part number 18321-99501, Hewlett-Packard CTD, August 1989. |
| Operating the HP 4954A Protocol Analyzer | *HP 4954A Protocol Analyzer Operating Manual*, 3d edition, part number 04954-99905, Hewlett-Packard CTD, June 1989. |

## Path and File Names

Certain functions require strings containing path and file names as input arguments. These functions break the path and file name string into individual components (`drive:directory/filename.extension`) and then pass the components to the operating system of the HP 4954A Protocol Analyzer. The operating system controls the disc system directory and file structure.

### Case Sensitivity

Both the DataCommC Programming Language and the protocol analyzer operating system are *case sensitive* to the file name. If, for example, a program is saved as "Test1.program" and you try to recall the file as "test1.program" the file will not be found.

### File Extensions

References to file names usually require inclusion of the *full* file extension. However, in the DataCommC environment there are two cases where the full file extension is not required, *include* files and *csource* files. These DataCommC file extensions can be abbreviated as follows.

| Full Extension | Abbreviation |
|---|---|
| `.csource` | `.c` |
| `.include` | `.h` |

For additional information about file extensions, see the *DataCommC Programming Language User's Guide*.

## Viewing Program Results

Control is passed back to the DataCommC environment when a program has finished executing. Any program information on the display is overwritten by the Run Program menu. To retain the program information use either an endless loop (`while(1);`), or a wait-for-input (`getch( );`) function as the last routine in the program.

The following program displays the "`Hello, world`!" message until you press any key.

```
main()
{
  printf("Hello, world!\n");
  getch();
}
```

The following program displays the "`Hello, world`!" message until you press Reset.

```
main()
{
  printf("Hello, world.\n");
  while(1);
}
```

# 2

# Function Categories

This chapter is an alphabetical listing of X.25 Test Environment functions organized by their general purpose and includes the library and a brief description of the function use. The categories and their general purpose are as follows.

- **Configuration Functions**
  Configuration functions (see table 2-1) preset those emulator parameters which must be set before any emulation stage is turned ON. The functions can only be used in emulation stage OFF.

- **Set Functions**
  Set functions (see table 2-2) are used to define those emulator parameters which can be set when the emulator is ON or OFF.

- **Get Functions**
  Get functions (see table 2-3) return current parameter values. Most of these functions can be used in any emulator stage. However, when the emulator stage is OFF, get functions involving emulator stages or states will return the last, not the current value.

- **Command and Send Functions**
  These functions (see table 2-4) transmit frames and packets. The emulator commands cause the emulator to perform a specific procedure.

- **Triggering Functions**
  These functions (see table 2-5) examine emulator messages passed to the program looking for a specific frame type or packet type.

- **Formatting and Decoding Functions**
  These functions (see table 2-6) format transmitted strings or decode received frames and packets.

For detailed information about any function see chapter 3 "Library Functions."

## Table 2-1. Configuration Functions

| Function | Library | Use |
|---|---|---|
| config_clock( ) | Level 1 | Presets the Level 1 DTE clock source to either DCE or DTE. |
| config_datacode_parity( ) | Level 1 | Presets the Level 1 emulator data code and parity. |
| config_device( ) | Level 1 | Presets the Level 1 emulator to either a physical DTE or a physical DCE. |
| config_extrctrl( ) | Level 2 | Presets the Level 2 emulator for either MOD8 or MOD128 frame sequence numbering. |
| config_frame_addrs( ) | Level 2 | Presets the Level 2 emulator frame addresses. |
| config_k( ) | Level 2 | Presets the Level 2 emulator window size. |
| config_LCI_ranges( ) | Level 3 | Presets the Level 3 emulator PVC and SVC LCI ranges. |
| config_logical_device( ) | Level 2 | Presets the Level 2 emulator device as either SUBSCRIBER or NETWORK. |
| config_N1( ) | Level 2 | Presets the Level 2 emulator incoming frame size. |
| config_N2( ) | Level 2 | Presets the number of times the Level 2 emulator attempts to successfully transmit a frame. |
| config_RR_idle_L2( ) | Level 2 | Presets the idle condition of the Level 2 emulator. |
| config_SN_L3( ) | Level 3 | Presets the Level 3 sequence numbering field value. |
| config_T1( ) | Level 2 | Presets the time the Level 2 emulator waits between frame transmission and receipt of the corresponding acknowledgment. |

Table 2-2. Set Functions

| Function | Library | Use |
|---|---|---|
| clear_busy_L2( ) | Level 2 | Clears the flag that forced the LAPB emulator into the busy state. |
| set_Abit( ) | Level 3 | Implements the TOA/NPI address format. |
| set_busy_L2( ) | Level 2 | Forces the LAPB emulator into the busy state. |
| set_Dbit( ) | Level 3 | Sets the Delivery bit default value. |
| set_facil_CALL( ) | Level 3 | Initializes the Facility field default string for a transmitted CALL REQUEST or INCOMING CALL packet. |
| set_facil_CALLC( ) | Level 3 | Initializes the Facility field default string for a transmitted CALL ACCEPTED or CALL CONNECTED packet. |
| set_facil_CLEAR( ) | Level 3 | Initializes the Facility field default string for a transmitted CLEAR REQUEST or CLEAR INDICATION packet. |
| set_facil_CLEARC( ) | Level 3 | Initializes the Facility field default string for a transmitted CLEAR CONFIRMATION packet. |
| set_LCI( ) | Level 3 | Sets the LCI field default value. |
| set_Qbit( ) | Level 3 | Sets the Qualifier bit default value. |
| set_userdata_CALL( ) | Level 3 | Initializes the Called User Data field default string for a transmitted CALL REQUEST or INCOMING CALL packet. or a send_CALL routine. |
| set_userdata_CALLC( ) | Level 3 | Initializes the Called User Data field default string for a transmitted CALL ACCEPTED or CALL CONNECTED packet. |
| set_userdata_CLEAR( ) | Level 3 | Initializes the Called User Data field default string for a transmitted CLEAR REQUEST or CLEAR INDICATION packet. |
| set_windowsize_L3( ) | Level 3 | Sets the Level 3 transmit window size. |

## Table 2-3. Get Functions

| Function | Library | Use |
|---|---|---|
| get_Abit( ) | Level 3 | Implements the TOA/NPI address format. |
| get_busy_L2( ) | Level 2 | Returns emulator Level 2 busy flag value. |
| get_Dbit( ) | Level 3 | Returns Delivery bit default value. |
| get_device( ) | Level 1 | Returns the device type for the transmit mode channel. |
| get_extctrl( ) | Level 2 | Returns the value of the extended control variable. |
| get_facil_CALL( ) | Level 3 | Returns default string and string length Facility field for a transmitted CALL REQUEST or INCOMING CALL packet. |
| get_facil_CALLC( ) | Level 3 | Returns default string and string length of the Facility field for a transmitted CALL ACCEPTED or CALL CONNECTED packet. |
| get_facil_CLEAR( ) | Level 3 | Returns default string and string length of the Facility field for a transmitted CLEAR REQUEST or CLEAR INDICATION packet. |
| get_facil_CLEARC( ) | Level 3 | Returns default string and string length of the Facility field for a transmitted CLEAR CONFIRMATION packet. |
| get_frame_addrs( ) | Level 2 | Returns and stores current address variables to user specified location. |
| get_k( ) | Level 2 | Returns the window size variable. |
| get_LCI( ) | Level 3 | Returns the LCI default value. |
| get_LCI_ranges( ) | Level 3 | Returns the LCI ranges. |
| get_leads_X25( ) | Level 1 | Returns the state of the interface pod leads. |
| get_logical_device( ) | Level 2 | Returns the device configuration variable, SUBSCRIBER or NETWORK. |
| get_N1( ) | Level 2 | Returns the Level 2 frame size variable. |
| get_N2( ) | Level 2 | Returns the Level 2 "try count." |

**Table 2-3. Get Functions (Continued)**

| Function | Library | Use |
|---|---|---|
| get_NR( ) | Level 2 | Returns the receive sequence number value. |
| get_NS( ) | Level 2 | Returns the send sequence number value. |
| get_PR( ) | Level 3 | Returns the received packet sequence number value. |
| get_PS( ) | Level 3 | Returns the transmitted packet sequence number value. |
| get_Qbit( ) | Level 3 | Returns the Qualifier bit default value. |
| get_RR_idle_L2( ) | Level 2 | Returns the Level 2 RR idle value. |
| get_SN_L3( ) | Level 3 | Returns the Level 3 sequence numbering default value. |
| get_stage_X25( ) | General | Returns the current emulation stage value. |
| get_state_L2( ) | Level 2 | Returns the Level 2 emulator state. |
| get_state_L3( ) | Level 3 | Returns the user selected LCI state. |
| get_T1( ) | Level 2 | Returns the wait time between frame transmission and corresponding acknowledgment. |
| get_userdata_CALL( ) | Level 3 | Returns the Called User Data field default string for a transmitted CALL REQUEST or INCOMING CALL packet. |
| get_userdata_CALLC( ) | Level 3 | Returns the Called User Data field default string for a transmitted CALL ACCEPTED or CALL CONNECTED packet. |
| get_userdata_CLEAR( ) | Level 3 | Returns the Called User Data field default string for a transmitted CLEAR REQUEST or CLEAR INDICATION packet. |
| get_windowsize_L3( ) | Level 3 | Returns the transmitted window size for all LCIs. |

## Table 2-4. Command and Send Functions

| Function | Library | Use |
|---|---|---|
| call_clear( ) | Level 3 | Initiates the procedure to clear a call on an LCI. |
| call_establish( ) | Level 3 | Initiates the procedure to establish a call on an LCI. |
| emulate_X25( ) | General | Stages up and stages down the emulator. |
| leads_off_X25( ) | Level 1 | Turns the Level 1 control interface leads OFF. |
| leads_on_X25( ) | Level 1 | Turns the Level 1 control interface leads ON. |
| link_down_X25( ) | Level 2 | Instructs the Level 2 emulator to initiate the link disconnect procedure. |
| link_up_X25( ) | Level 2 | Instructs the Level 2 emulator to initiate the link up procedure. |
| reset_channel( ) | Level 3 | Initiates the procedure to reset an LCI. |
| restart_L3( ) | Level 3 | Initiates the procedure to restart Level 3. |
| send_CALL( ) | Level 3 | Sends a formatted CALL REQUEST or INCOMING CALL packet. |
| send_CALLC( ) | Level 3 | Sends a formatted CALL ACCEPTED or CALL CONNECTED packet. |
| send_CLEAR( ) | Level 3 | Sends a formatted CLEAR REQUEST or CLEAR INDICATION packet. |
| send_CLEARC( ) | Level 3 | Sends a formatted CLEAR CONFIRMATION packet. |
| send_DATA( ) | Level 3 | Sends a formatted DATA packet. |
| send_DIAG( ) | Level 3 | Sends a formatted DIAGNOSTIC packet. |
| send_DISC( ) | Level 2 | Sends a DISC (disconnect) frame. |
| send_DM( ) | Level 2 | Sends a DM (disconnect mode) frame. |
| send_FRMR( ) | Level 2 | Sends a FRMR (frame reject) frame. |
| send_I( ) | Level 2 | Sends an Information frame. |
| send_INT( ) | Level 3 | Sends a formatted INTERRUPT packet. |
| send_INTC( ) | Level 3 | Sends a formatted INTERRUPT CONFIRMATION packet. |
| send_REG( ) | Level 3 | Sends a formatted REGISTRATION packet on LCI 0. |

Table 2-4. Command and Send Functions (Continued)

| Function | Library | Use |
|---|---|---|
| send_REGC( ) | Level 3 | Sends a formatted REGISTRATION CONFIRMATION packet on LCI 0. |
| send_REJ_L2( ) | Level 2 | Sends a REJ (reject) frame. |
| send_REJ_L3( ) | Level 3 | Sends a formatted REJ (reject) packet. |
| send_RESET( ) | Level 3 | Sends a formatted RESET REQUEST or RESET INDICATION packet. |
| send_RESETC( ) | Level 3 | Sends a formatted RESET CONFIRMATION packet. |
| send_RESTART( ) | Level 3 | Sends a formatted RESTART REQUEST or RESTART INDICATION packet on LCI 0. |
| send_RESTARTC( ) | Level 3 | Sends a formatted RESTART CONFIRMATION packet on LCI 0. |
| send_RNR_L2( ) | Level 2 | Sends a RNR (receiver not ready) frame. |
| send_RNR_L3( ) | Level 3 | Sends a formatted RNR (receiver not ready) packet. |
| send_RR_L2( ) | Level 2 | Sends a RR (receiver ready) frame. |
| send_RR_L3( ) | Level 3 | Sends a formatted RR (receiver ready) packet. |
| send_SABM( ) | Level 2 | Sends a SABM (set asynchronous balanced mode) frame. |
| send_SABME( ) | Level 2 | Sends a SABME (set asynchronous balanced mode extended) frame. |
| send_UA( ) | Level 2 | Sends a UA (unnumbered acknowledgement) frame. |
| sendf_X25( ) | General | A general purpose send command for sending any X.25 frame using the Level 1 Emulator. |

Table 2-5. Triggering Functions

| Function | Library | Use |
|---|---|---|
| get_frame( ) | Level 2 | Returns frames from the input message queue. |
| get_packet( ) | Level 3 | Returns packets from input message queue. |

Table 2-6. Formatting and Decoding Functions

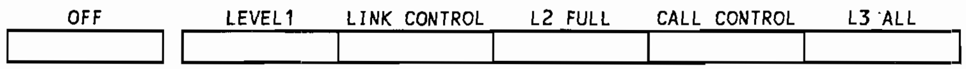
| Function | Library | Use |
|---|---|---|
| decode_frame( ) | Level 2 | Decodes a LAPB frame string and places the results in the structure pointed to by out_ptr. |
| decode_packet( ) | Level 3 | Decodes an X.25 Level 3 packet string and places the results in the structure pointed to by out_ptr. |
| format_L3_X25( ) | Level 3 | Formats any X.25 Level 3 packet. |

3

# Library Functions

This chapter is an alphabetical listing of the X.25 Test Environment functions that includes the following information.

**Emulation Stage** To use an X.25 Test Environment function the emulator is required to be in a specific emulation stage or stages. The following emulation stage graphic appears at the top of each library function. The shaded areas indicate the stages in which the function can be used.



**Format** A program fragment showing the function declaration type, and the position and declaration types of its parameters. The fragment also lists any necessary include files.

**Description** A description of what the function does, including a list of acceptable values that may be used for input parameters, and the possible values returned by output parameters.

**Return Values** A list of the values that the function may return, and short descriptions of the conditions that could cause those values to be returned.

**See Also** Related X.25 Test Environment and DataCommC functions.

**Example** A short program showing typical function use.

**Note** The examples in this section are provided for clarification only. HP assumes no responsibility for their functionality or fitness for a specific purpose.

# call_clear( )

|  |  |  |  | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
|  |  |  |  | ▨ | ▨ |

## Format

```
#include <X25.include>

int call_clear(control, args)

char *control;
char *args;
```

## Description

This Level 3 library function instructs the X.25 emulator to initiate the procedure for clearing a call on a selected logical channel.

## Conversion Specifications

Packet fields are defined by a control string and the associate argument list which are described in detail in chapter 8, "Frame and Packet Formatting," *HP 18321A X.25 Test Environment User's Guide*. The following table defines the conversion specifications for this function.

| Specification | Use | Default Value |
|---|---|---|
| %[n]A | Implement TOA/NPI address format. | A_Bit |
| %[n]CC | Define the cause code. | 0 |
| %[n]CDAb or<br>%[n]CDAs | Define the length and location of the called address. | Length = 0<br>Data Pointer = None |
| %[n]CGAb or<br>%[n]CGAs | Define the length and location of the calling address. | Data Pointer = None |

| Specification | Use | Default Value |
|---|---|---|
| %[n]DC | An *optional entry* that contains the diagnostic code. | |
| %[n]Fb | Define the facilities length and data for *optional* user facilities. | facil_CLEAR and facil_CLEAR_length |
| %[n]LCI | Define the logical channel to clear. | LCI_value |
| %[n]UDb or %[n]UDs | Define the length and location of user data. | userdata_CLEAR and userdata_CLEAR_length |

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -121 | ERROR_121 | Emulator is not in the correct stage to process this function. |
| -122 | ERROR_122 | Front end not configured properly. |
| -123 | ERROR_123 | An error was found in the control string. |
| -124 | ERROR_124 | One of the control string parameters has an illegal value. |
| -125 | ERROR_125 | One of the control string parameters is entered more than once. |
| -126 | ERROR_126 | Change the send string length to be between 2 bytes and 4,106 bytes. |
| -131 | ERROR_131 | Front end is not running; stop_data( ) may have been called. |
| -141 | ERROR_141 | LAPB window is full. |
| -142 | ERROR_142 | LAPB emulator is in the wrong state. |
| -144 | ERROR_144 | LCI is in the wrong state. |
| -145 | ERROR_145 | LCI is invalid. |
| -162 | ERROR_162 | Mixed use of CGAs and CGAb in the same control string. |
| -163 | ERROR_163 | Mixed use of CDAs and CDAb in the same control string. |
| -164 | ERROR_164 | String length for one or more of the following character specifications too long: CDAb, CDAs, CGAb, CGAs, or Fb. |
| -165 | ERROR_165 | One of the elements of the string pointed by CDAs or CGAs is not a digit. |

# call_clear( )

## See Also

| | | |
|---|---|---|
| call_establish( ) | config_SN_L3( ) | get_SN_L3( ) |
| get_facil_CALL( ) | get_userdata_CALL( ) | send_CALL( ) |
| set_Abit( ) | set_facil_CALL( ) | set_LCI_bit( ) |
| set_userdata_CALL( ) | | |

## Example

The following example shows how a combination of control string entries, argument list entires, and default values can be used with `call_clear( )`.

```
char *User_Data;
User_Data = "This is UserData";
call_clear("%2LCI%12UDs", User_Data);
```

## call_establish( )

### Format

```
#include <X25.include>

int call_establish(control, args)

char *control;
char *args;
```

### Description

This Level 3 library function instructs the X.25 emulator to initiate a call establishing procedure on the selected logical channel. If this routine returns SUCCESSFUL, the user will receive a USER_STATUS message with a subtype of CALL_EST or CALL_CLEAR and the LCI field will equal the LCI the call was made on.

### Conversion Specifications

Packet fields are defined by a control string and the associate argument list which are described in detail in chapter 8, "Frame and Packet Formatting," *HP 18321A X.25 Test Environment User's Guide*. The following table defines the conversion specifications for this function.

| Specification | Use | Default Value |
|---|---|---|
| %[n]A | Implement TOA/NPI address format. | A_bit |
| %[n]CDAb or<br>%[n]CDAs | Define the length and location of the called address. | Length = 0<br>Data Pointer = None |
| %[n]CGAb or<br>%[n]CGAs | Define the length and location of the calling address. | Length = 0<br>Data Pointer = None |

# call_establish( )

| Specification | Use | Default Value |
|---|---|---|
| %[n]D | Specify Level 3 delivery confirmation requirement. | 0 |
| %[n]Fb | Define the facilities length and data for *optional* user facilities. | facil_CALL and facil_CALL_length |
| %[n]LCI | Define the logical channel to clear. | LCI_value |
| %[n]UDb or %[n]UDs | Define the length and location of user data. | userdata_CALL and userdata_CALL_length |

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -121 | ERROR_121 | Emulator is not in the correct stage to process this function. |
| -122 | ERROR_122 | Front end not configured properly. |
| -123 | ERROR_123 | An error was found in the control string. |
| -124 | ERROR_124 | One of the control string parameters has an illegal value. |
| -125 | ERROR_125 | One of the control string parameters is entered more than once. |
| -126 | ERROR_126 | Change the send string length to be between 2 bytes and 4,106 bytes. |
| -131 | ERROR_131 | Front end is not running; stop_data( ) may have been called. |
| -141 | ERROR_141 | LAPB window is full. |
| -142 | ERROR_142 | LAPB emulator is in the wrong state. |
| -144 | ERROR_144 | LCI is in the wrong state. |
| -145 | ERROR_145 | LCI is invalid. |
| -146 | ERROR_146 | All LCIs have been used. |
| -162 | ERROR_162 | Mixed use of CGAs and CGAb in the same control string. |
| -163 | ERROR_163 | Mixed use of CDAs and CDAb in the same control string. |
| -164 | ERROR_164 | String length for one or more of the following character specifications is too long: CDAb, CDAs, CGAb, CGAs, or Fb. |
| -165 | ERROR_165 | One of the elements of the string pointed by CDAs or CGAs is not a digit. |

## See Also

| | | |
|---|---|---|
| call_clear( ) | config_SN_L3( ) | get_SN_L3( ) |
| get_facil_CALL( ) | get_userdata_CALL( ) | send_CALL( ) |
| set_Abit( ) | set_facil_CALL( ) | set_LCI_bit( ) |
| set_userdata_CALL( ) | | |

## Example

The following example shows how a combination of control string entries, argument list entires, and default values can be used with `call_establish( )`.

```
char *Called_Addr, *Calling_Addr;
char Facil[] = {0x42, 0x77};
Called_Addr = "5551212";
Calling_Addr = "555132";
call_establish("%0A%0D,%3LCI%CDAs%CGAs%2Fb", Called_Addr, Calling_Addr, Facil);
```

# clear_busy_L2()

|  |  |  |  | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|---|
|  |  |  |  | ▨ | ▨ | ▨ |

## Format

```
#include <X25.include>

int clear_busy_L2( )
```

## Description

This Level 2 function clears the LAPB busy flag which is used to force the LAPB emulator into the busy state. The initial value of the LAPB busy flag is `0`.

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -121 | ERROR_121 | Emulator is not in the correct stage to process this function. |

## See Also

`get_busy_L2( )` `set_busy_L2( )`

## Example

The following example sets the LAPB busy flag to `0`.

```
clear_busy_L2( );
```

## config_clock( )

OFF

### Format

```
#include <dlib.include>
#include <X25.include>

int config_clock(clock_source)

unsigned char clock_source;
```

### Description

This Level 1 library function presets the DTE clock source to be either the DCE or the DTE (values of 1 and 2, respectively). The initial value of the DTE clock source is DCE.

### Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -101 | ERROR_101 | Check parameter 1 for correct value. |
| -121 | ERROR_121 | Emulator is not in the correct stage to process this function. |

### See Also

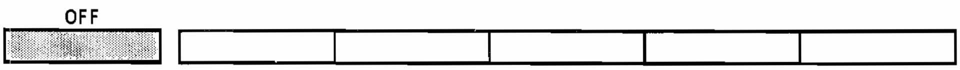
| DataCommC | X.25 |
|---|---|
| set_protocol( ) | get_clock( ); |

### Example

The following example will establish the DCE as the Level 1 emulator DTE clock source.

```
config_clock(DCE);
```

## config_datacode_parity( )



### Format

```
#include <dlib.include>
#include <X25.include>

int config_datacode_parity(datacode, parity)

unsigned int datacode, parity;
```

### Description

This Level 1 library function presets the data code and the parity used by the emulator.

The following is a list of data codes and parities, and their associated values.

| Data Code | Value | Parity | Value |
|---|---|---|---|
| ASCII8 | 1 | ODD | 1 |
| ASCII7 | 2 | EVEN | 2 |
| EBCDIC | 3 | NO PARITY | 3 |
| HEX8 | 4 | NO CHANGE | 0xFFFF |

The initial values are ASCII8 (1), and NO PARITY (3).

### Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| 101 | WARNING_101 | Cannot process parameter 1; it is assigned the default value. |
| 102 | WARNING_102 | Cannot process parameter 2; it is assigned the default value. |
| -121 | ERROR_121 | Emulator is not in the correct stage to process this function. |

### See Also

DataCommC

get_datacode( )

set_protocol( )

get_parity( )

## Example

The following example sets the data code to ASCII7 with ODD parity.

```
        config_datacode_parity(ASCII7,ODD);
```

## config_device( )

| OFF | | | | | |
|---|---|---|---|---|---|
| ▨ | | | | | |

### Format

```
#include <dlib.include>
#include <X25.include>

int config_device(device)

unsigned char device;
```

### Description

The Level 1 library function presets the Level 1 emulator to be a physical DCE or DTE (values of 1 and 2 respectively). The physical device is initially set to be DTE.

### Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -101 | ERROR_101 | Check parameter 1 for correct value. |
| -121 | ERROR_121 | Emulator is not in the correct stage to process this function. |

### See Also

| DataCommC | X.25 |
|---|---|
| set_channel_config( ) | get_device( ) |

### Example

The following example will set the Level 1 emulator to be a physical DCE when it is turned ON.

```
config_device(DCE);
```

## config_extctrl( )

OFF

### Format

```
#include <dlib.include>
#include <X25.include>

int config_extctrl(ext_ctrl)

unsigned char ext_ctrl;
```

### Description

This Level 2 library function presets the emulator to use either standard control fields (mod_8 sequence numbering), or extended control (mod_128 sequence numbering). The emulator is initially configured to use standard control fields.

| Control | Value | Constant | SN |
|---|---|---|---|
| Standard | 0 | OFF | mod_8 |
| Extended | 1 | ON | mod_128 |

### Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -101 | ERROR_101 | Check parameter 1 for correct value. |
| -121 | ERROR_121 | Emulator is not in the correct stage to process this function. |

### See Also

`get_extctrl( );`

### Example

The following example causes the Level 2 emulator to use mod 128 frame sequence numbering.

```
config_extctrl(ON);
```

# config_frame_addrs( )

| OFF | | | | | |
|---|---|---|---|---|---|
| ▨ | | | | | |

## Format

```
#include <X25.include>

int config_frame_addrs(subscriber_addr, network_addr)

unsigned char subscriber_addr, network_addr;
```

## Description

This Level 2 library function configures the frame addresses used by the emulator based on the device configuration established by `config_logical_device( )`.

When the emulator configuration is SUBSCRIBER, it uses the specified `network_addr` as the frame address in all COMMAND frames. It uses the specified `subscriber_addr` as the frame address in all RESPONSE frames.

When the emulator configuration is NETWORK, it uses the specified `subscriber_addr` as the frame address in all COMMAND frames. It uses the specified `network_addr` as the frame address in all RESPONSE frames. The `subscriber_addr` initial value is 3 and the `network_addr` initial value is 1.

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -121 | ERROR_121 | Emulator is not in the correct stage to process this function. |

## See Also

`config_logical_device( )` `get_frame_addrs( )`

## Example

The following example sets the `subscriber_addr` to 7 and the `network_addr` to 5.

```
config_frame_addrs(7,5);
```

# config_k( )

OFF

## Format

```
#include <X25.include>

int config_k(k)

unsigned int k;
```

## Description

This Level 2 library function defines the maximum number of sequentially numbered Information frames (k) on the DTE-DCE link that can be unacknowledged at any given time, the window size. The range for k is from one frame to seven frames, with an initial value of 2 frames.

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| 101 | WARNING_101 | Check process parameter 1; it is assigned the default value. |
| -121 | ERROR_121 | Emulator is not in the correct stage to process this function. |

## See Also

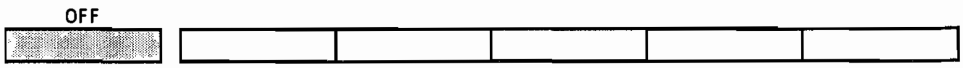
get_k( )

## Example

The following example sets the window size to 2 unacknowledged, sequentially numbered frames.

```
config_k(2);
```

## config_LCI_ranges( )



### Format

```
#include <X25.include>

int config_LCI_ranges(PVC_low, PVC_high, SVC_low, SVC_high)

unsigned int PVC_low, PVC_high, SVC_low, SVC_high;
```

### Description

This Level 3 library function presets the LCI ranges used by the emulator. These values are only used when the emulator is in stage `CALL_CONTROL` or `L3_ALL`. Channels in the range specified by `SVC_low` to `SVC_high` are all two-way channels.

In general, the ranges for each PVC and SVC parameter are from Channel 1 through Channel 4095. However the following limitations apply,

- `PVC_low <= PVC_high < SVC_low <= SVC_high.`
- `0` is only used to specify no `PVC`s or `SVC`s.
- To specify no `PVC`s, both `PVC_low` and `PVC_high` must be set to `0`.
- To specify no `SVC`s, both `SVC_low` and `SVC_high` must be set to `0`.

### Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -101 | ERROR_101 | Check parameter 1 for correct value. |
| -102 | ERROR_102 | Check parameter 2 for correct value. |
| -103 | ERROR_103 | Check parameter 3 for correct value. |
| -104 | ERROR_104 | Check parameter 4 for correct value. |
| -121 | ERROR_121 | Emulator is not in the correct stage to process this function. |

## See Also

`get_LCI_ranges( )`

## Example

The following example would set the PVC range from Channel 1 to Channel 2 and the SVC range from Channel 3 through Channel 15.

```
        config_LCI_ranges(1,2,3,15);
```

This example would set Channels 1 through 15 as SVC channels with no PVC channels.

```
        config_LCI_ranges(NO_PVC,NO_PVC,1,15);
```

## config_logical_device( )

| OFF | | | | | |
|---|---|---|---|---|---|
| ▨ | | | | | |

### Format

```
#include <X25.include>

int config_logical_device(logical_device)

unsigned char logical_device;
```

### Description

This Level 2 library function presets the LAPB emulator configuration to be either the SUBSCRIBER (0) or the NETWORK (1). The initial logical_device is SUBSCRIBER. The logical device configuration is used in conjunction with the values for subscriber address and network address to determine the LAPB frame address that is appropriate for a specific frame.

### Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -101 | ERROR_101 | Check parameter 1 for correct value. |
| -121 | ERROR_121 | Emulator is not in the correct stage to process this function. |

### See Also

config_frame_addrs( )    get_logical_device( )

### Example

The following example sets the emulator to function as the NETWORK device.

```
config_logical_device(NETWORK);
```

| OFF | | | | | |
|---|---|---|---|---|---|
| ▨ | | | | | |

## Format

```
#include <X25.include>

int config_N1(N1)

unsigned long N1;
```

## Description

This Level 2 library function defines the maximum allowable incoming LAPB frame size by specifying the total number of bytes (N1) in the frame.

N1 is the total number of bytes in the frame including the address field (1 byte), the control field (1 or 2 bytes depending whether extended control is used), the information field, and the FCS (2 bytes). N1 can be set to any value between 0 and 65,535 bytes. The initial value is 65535. The configured value of N1 has no effect on the size of frames transmitted by the emulator.

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| 101 | WARNING_101 | Cannot process parameter 1, it is assigned the default value. |
| -121 | ERROR_121 | Emulator is not in the correct stage to process this function. |

## See Also

get_N1( )

## Example

The following example sets the maximum size of received LAPB frames to 135 bytes, for a data field(packet size) of 128 bytes. This example assumes the ext_ctrl is OFF.

```
config_N1(135);
```

# config_N2( )

| OFF | | | | | |
|---|---|---|---|---|---|
| ▨ | | | | | |

## Format

```
#include <X25.include>

int config_N2(N2)

unsigned int N2;
```

## Description

This Level 2 library function defines the total number of attempts (`N2`) made by the emulator to complete the successful transmission of a frame. A time of `T1` elapses between each attempt. The range for `N2` is from `0` to `32,767` with an initial value of `20`.

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| 101 | WARNING_101 | Check process parameter 1; it is assigned the default value. |
| -121 | ERROR_121 | Emulator is not in the correct stage to process this function. |

## See Also

`config_T1( )`  `get_N2( )`

## Example

The following example configures the LAPB emulator to make `10` attempts at a successful frame transmission, rather than the initial value of `20`.

```
int config_N2(10);
```

# config_RR_idle_L2()

OFF

## Format

```
#include <dlib.include>
#include <X25.include>

int config_RR_idle_L2(rr_idles)

unsigned char rr_idles;
```

## Description

This Level 2 library function configures the Level 2 emulator to idle in one of the following conditions at emulation stage L2_FULL and above. The initial state of rr_idles is OFF.

- Idle in RR frames with poll bit set each time T1 times out during idle time. This is the rr_idles ON (1) state.
- Idle in flags, the rr_idles OFF (0) state.

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -101 | ERROR_101 | Check parameter 1 for correct value. |
| -121 | ERROR_121 | Emulator is not in the correct stage to process this function. |

## See Also

config_T1( )    get_rr_idle_L2( )

## Example

The following example sets the LAPB emulator to idle in RR frames.

```
config_RR_idle_L2(ON);
```

# config_SN_L3()

| OFF | | | | | |
|---|---|---|---|---|---|
| ▨ | | | | | |

## Format

```
#include <X25.include>

int config_SN_L3(L3mod)

unsigned char L3mod;
```

## Description

This Level 3 library function sets the transmitted packet sequencing numbering (SN) field value.

The values for L3mod are either MOD_128, or MOD_8 (the initial value).

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -101 | ERROR_101 | Check parameter 1 for correct value. |
| -121 | ERROR_121 | Emulator is not in the correct stage to process this function. |

## See Also

get_SN_L3( )

## Example

The following example sets the transmitted packet sequencing numbering field SN to MOD_128.

```
config_SN_L3(MOD_128);
```

## config_T1( )

OFF

## Format

```
#include <X25.include>

int config_T1(T1)

unsigned long T1;
```

## Description

This Level 2 library function defines the time (T1), in tenths of a second, that the transmitter waits between transmitting a frame and receiving the corresponding acknowledgment. The recovery procedure begins if T1 expires without the required acknowledgement.

The range for T1 is from 0 to 99,999/10 seconds with an initial value of 30/10 (3 seconds).

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -101 | ERROR_101 | Check parameter 1 for correct value. |
| -121 | ERROR_121 | Emulator is not in the correct stage to process this function. |

## See Also

`config_N2( )` `get_T1( )`

## Example

The following example would set T1 for a time of 6 seconds.

```
config_T1(60);
```

# decode_frame()

| OFF | | LEVEL1 | LINK CONTROL | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|---|
| ▨ | | ▨ | ▨ | ▨ | ▨ | ▨ |

## Format

```
#include <X25.include>

int decode_frame( lapb_str_length, lapb_str_ptr, out_ptr)

int lapb_str_length;
unsigned char *lapb_str_ptr;
LAPB_DECODE *out_ptr;
```

## Description

This Level 2 library function decodes the string pointed to by `lapb_str_ptr` and places the decoded results in the structure pointed to by `out_ptr`.

The user must declare a structure of the type `LAPB_DECODE` (see "Structures," in this section) and then pass the structure address in as `out_ptr`.

## Results

`out_ptr->result` indicates the decode process result as one of the following values.

| Value | Constant | Definition |
|---|---|---|
| 0 | OK | Frame successfully decoded. |
| 0X02 | ADDR_ERR | Frame address not `subscriber_addr` or `network_addr`. |
| 0X04 | FRMR_TOO_SHORT | FRMR frame not long enough. |

| | Minimum FRMR Frame Length |
|---|---|
| Modulo 8 | 5 bytes + FCS |
| Modulo 128 | 7 bytes + FCS |

| | | |
|---|---|---|
| 0X01 | INVALID_FRAME | Frame length less than allowed minimum. |

| | Minimum Frame Length |
|---|---|
| **Modulo 8** | I frame - 2 bytes + FCS<br>S frame - 2 bytes + FCS<br>U frame - 2 bytes + FCS |
| **Modulo 128** | I frame - 3 bytes + FCS<br>S frame - 3 bytes + FCS<br>U frame - 2 bytes + FCS |

| | | |
|---|---|---|
| 0X08 | UNEX_INFO | Supervisory or Unnumbered frame too long. |

| | Maximum Frame Length |
|---|---|
| **Modulo 8** | S frame - 2 bytes + FCS<br>U frame - 2 bytes + FCS |
| **Modulo 128** | S frame - 3 bytes + FCS<br>U frame - 3 bytes + FCS |

The error conditions are bit mapped and can be ORed together so more than one error condition can be reported.

| Bit Number | Error Condition |
|---|---|
| 0 | INVALID_FRAME |
| 1 | ADDR_ERR |
| 2 | FRMR_TOO_SHORT |
| 3 | UNEX_INFO |

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |

# decode_frame( )

## See Also

| | | |
|---|---|---|
| set_ext_ctrl( ) | set_logical_device( ) | set_frame_addr( ) |
| get_ext_ctrl( ) | get_logical_device( ) | get_frame_addrs( ) |
| decode_packet( ) | | |

## Example

```
HP_MESSAGE Mssg;
LAPB_DECODE Decode_Output;
read_message(&Mssg,0L);
if(Mssg.message.type = USER_DATA)
   decode_frame(Mssg.message.body.event.event_ptr->length,
   Mssg.message.body.event.event_ptr->frame,&Decode_Output);
```

## Structures

This function uses the following structures.

### LAPB_DECODE Structure

```
typedef struct
{

       unsigned char result;        /*  OK:  Frame successfully decoded. */
                                    /*  INVALID:  Frame too short. */
       long int L2_raw_length;      /*  0 when not used.  */

       unsigned char *L2_raw_ptr;   /*  NULL when not used.  */
       unsigned char addr;          /*  Address field value.  */
       unsigned char command;       /*  0 = FALSE = RESPONSE frame.
                                        1 = TRUE  = COMMAND frame.  */
       unsigned int ctrl_field;     /*  Entire control field.  */
       unsigned char ctrl_id;       /*  Frame Type identifier.
                                        unknown frame type = UNDEF  */
       unsigned char ctrl_ref;      /*  Mapped FTI used to access a table;
                                        value 1 - 10.  */
       unsigned char pf_bit;        /*  Poll or Final bit value.  */
       unsigned char ctrl_format;   /*  Information Frame:
                                        Supervisory Frame:
                                        Unnumbered Frame: */
       unsigned char NS;            /*  NS value.  */
       unsigned char NR;            /*  NR value.  */
       FRMR_INFO frmr;              /*  When ctrl_id==FRMR.structure follows  */
       unsigned int L3_raw_length;  /*  0 when not used.  */
       unsigned char *L3_raw_ptr;   /*  NULL when not used.  */
} LAPB_DECODE;
```

## FRMR_INFO Structure

```
typedef struct
{
        unsigned int ctrl;            /*  Rejected frame control field. */
        unsigned char Vr;             /*  V(r) value. */
        unsigned char Vs;             /*  V(s) value. */
        unsigned char command;        /*  0 = FALSE = RESPONSE frame
                                          1 = TRUE  = COMMAND frame.  */
        unsigned char w_bit;          /*  W bit value.
        unsigned char x_bit;          /*  X bit value. */
        unsigned char y_bit;          /*  Y bit value. */
        unsigned char z_bit;          /*  Z bit value. */
} FRMR_INFO;
```

# decode_packet( )

| OFF | LEVEL1 | LINK CONTROL | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
| ▨ | ▨ | ▨ | ▨ | ▨ | ▨ |

## Format

```
#include <stdio.include>
#include <dlib.include>
#include <X25.include>

int decode_packet(packet_length, packet_ptr, is_dce, out_ptr)

int packet_length;
unsigned char *packet_ptr;
int is_dce;
DEC_X25 *out_ptr;
```

## Description

This function decodes the string passed in and places the decoded results in the structure pointed to by `out_ptr`. The user must declare a structure of type DEC_X25 and then pass the address of that structure in as `out_ptr`.

## Results

`out_ptr->result` indicates the result of the decode process.

| Bit | Error | Definition |
|---|---|---|
| 0 | OK | Packet successfully decoded. |
| -1 | INVALID_PACKET | Packet length is less than 3 bytes. |

The following error conditions are bit mapped and can be ORed together so more than one error condition can be reported.

| Bit | Error | Definition |
|---|---|---|
| 0 | PTI_ERR | Undefined packet type. |
| 1 | USER_LEN_ERR | User data length is either greater than the remainder of the packet or greater than 128 bytes. |
| 2 | FACIL_LEN_ERR | Facilities length is either greater than the remainder of the packet or greater than 109 bytes. |

| Bit | Error | Definition |
|---|---|---|
| 3 | REGIST_LEN_ERR | Register length is either greater than the remainder of the packet or greater than 109. |
| 4 | NONBCD_ERR | Address field contains a non-BCD number. |
| 5 | ADDR_LEN_ERR | The address length is greater than the remainder of the packet. |
| 6 | LCI_ERR | LCI is not valid for the packet type. |
| 7 | SN_ERR | Sequence numbers 0 and 3 are not valid. |
| 8 | Dbit_ERR | D_bit is not valid for packet type. |
| 9 | Qbit_ERR | Q_bit is not valid for packet type. |
| 10 | PACKET_SHORT | The packet length is less than that required for the packet type. |
| 11 | PACKET_LONG | The packet length is too long for the packet type. |

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -101 | ERROR_101 | Check parameter 1 for correct value. |
| -103 | ERROR_103 | Check parameter 3 for correct value. |

## See Also

decode_frame( )

## Example

The following example decodes a DCE packet and stores the decoded results into the structure outX25.

```
int packet_length;
unsigned char *packet_ptr;
int device;
DEC_X25 outX25;

decode_packet(packet_length, packet_ptr, DCE_FRAME, &outX25);
```

# decode_packet()

## Structures

This function uses the following structures.

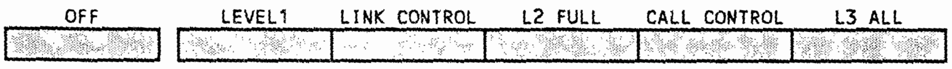
### DEC_X25 Structure

```
/*******************************************************************************
*                                                                              *
*  The DEC_X25 structure is used to store the decoded Level 3 packet infor-    *
*  mation.  Appropriate error conditions are in the result field.              *
*                                                                              *
*******************************************************************************/

typedef struct
{
        int result;       /* error conditions */
        int is_dec;
        int packet_len;
        unsigned char *packet_ptr;
        int l4_raw_len;
        unsigned char *l4_raw_ptr;

        unsigned char Qbit;
        unsigned char Dbit;
        unsigned char SN;
        unsigned char LCGN;
        unsigned char LCN;
        int LCI;
        unsigned char PTI:
        unsigned char PR;
        FACILITIES *facil_struct;  /* pointer to facility struct */
        X25PACKETS Packet_Type;
} DEC_X25;
```

See chapter 4 "Include Files", of this manual for additional details on structures.

## decode_packet( )

# emulate_X25()



## Format

```
#include <system.include>
#include <dlib.include>
#include <X25.include>

int emulate_X25(stage)

unsigned int stage;
```

## Description

This function turns on the emulator at the specified stage. There are six emulator stages, and they are:

| Emulation Level | Stage |
|---|---|
| Level 3 (X.25) | L3_ALL<br>CALL_CONTROL |
| Level 2 (LAPB) | L2_FULL<br>LINK_CONTROL |
| Level 1 | LEVEL1 |
| OFF | OFF |

The stage specified by the stage parameter must be lower than the current emulator stage, or an error is returned. From the OFF stage, any emulator stage can be turned ON.

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| 121 | WARNING_121 | Emulator is already in the specified stage; no action taken. |
| -101 | ERROR_101 | Check parameter 1 for correct value. |
| -121 | ERROR_121 | Emulator is not in the correct stage to process this function. |
| -128 | ERROR_128 | The emulator can only be staged down; use `emulate_X25(OFF)` and then `emulate_X25(stage)` to stage up. |

## See Also

`get_stage_X25( )`

## Example

The following example shows how to change the stage of the emulator.

```
emulate_X25(OFF);
emulate_X25(L2_FULL);
emulate_X25(OFF);
emulate_X25(LEVEL1);
emulate_X25(OFF);
emulate_X25(L3_ALL);
```

# format_L3_X25()

| OFF | LEVEL1 | LINK CONTROL | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
| ▨ | ▨ | ▨ | ▨ | ▨ | ▨ |

## Format

```
#include <X25.include>

int format_L3_X25(packet_ptr_ptr, packet_length_ptr, control, args)

char **packet_ptr_ptr;
int *packet_length_ptr;
```

## Description

This Level 3 library function is a general purpose formatting function used to format *any* X.25 Level 3 packet. It has two fixed parameters -- `packet_ptr_ptr` and `packet_length_ptr` -- and the variable parameters described in the conversion specification table.

| | |
|---|---|
| `packet_ptr_ptr` | Places the address of the formatted packet at the address `*packet_ptr_ptr`. |
| `packet_length_ptr` | This is a pointer to an integer that defines the packet length. |

## Conversion Specifications

Packet fields are defined by a control string and the associate argument list which are described in detail in chapter 8, "Frame and Packet Formatting," *HP 18321A X.25 Test Environment User's Guide*. The following table defines the conversion specifications for this function.

| Specification | Use | Default Value |
|---|---|---|
| %[n]A | Implement TOA/NPI address format. | A_bit value |
| %[n]CC | cause code | 0 |
| %[n]CDAb or<br>%[n]CDAs | Define the length and location of the called address data or the calling address data. | Length = 0<br>Data Pointer = None |
| %[n]CGAb or<br>%[n]CGAs | Define the length and location of the called address data or the calling address data. | Length = 0<br>Data Pointer = None |
| %[n]D | Specify Level 3 delivery confirmation requirement. | D_bit |
| %[n]DC | Level 3 Diagnostic Code | 0 or None |
| %[n]DEb | Define the length and location of the Level 3 Diagnostic Explanation data. | None |
| %[n]Fb | Define the length and location of the Level 3 Facilities data. | **Information Frame**<br>Packet type dependent.<br>**Supervisory Frame**<br>Length = 0<br>Data Location = None<br>**Unnumbered Frame**<br>Length = 0<br>Data Location = None |
| %[n]LCI | Define the logical channel. | LCI_value |
| %[n]M | Signify requirement for another DATA packet. | 0 (OFF) |
| %[n]PR or<br>%[n]PS | Define Level 3 receive (PR) or send (PS) sequence number. | 0 |
| %[n]PT | Define the Level 3 Packet type. | DATA |
| %[n]Q | Specify Level 3 qualification requirement. | Q_bit value |

# format_L3_X25()

| Specification | Use | Default Value |
|---|---|---|
| %[n]Rb | Define the length and location of Level 3 Registration data. | Length = 0<br>Data Location = None |
| %[n]SN | Define Level 3 packet sequence numbering scheme as either modulo 8 or 128. | SN_value |
| %[n]UDb or<br>%[n]UDs | Define the length and location of Level 3 user data. | None when FDb, or FDs<br>Otherwise, frame type dependent.<br>**Information Frame**<br>Packet type dependent<br>**Supervisory Frame**<br>Length = 0<br>Data Location = None<br>**Unnumbered Frame**<br>Length = 0<br>Data Location = None |

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -123 | ERROR_123 | An error was found in the control string. |
| -124 | ERROR_124 | One of the control string parameters has an illegal value. |
| -125 | ERROR_125 | One of the control string parameters entered more than once. |
| -126 | ERROR_126 | Change the send string length to be between 2 bytes and 4,106 bytes. |
| -162 | ERROR_162 | Mixed use of CGAs and CGAb in the same control string. |
| -163 | ERROR_163 | Mixed use of CDAs and CDAb in the same control string. |
| -164 | ERROR_164 | String length for one or more of the following character specifications is too long: CDAb, CDAs, CGAb, CGAs, or Fb. |
| -165 | ERROR_165 | One of the elements of the string pointed by CDAs or CGAs is not a digit. |
| -167 | ERROR_167 | %Q, which is illegal when PT is CALL, CALLC, CLEAR, CLEARC, REG, or REGC is entered. |
| -168 | ERROR_168 | %A is entered when PT is set to be one of the packets other than CALL, CALLC, CLEAR, CLEARC, REG, or REGC. |

## See Also

| | | |
|---|---|---|
| call_clear( ) | call_establish( ) | config_LCI_ranges( ) |
| config_SN_L3( ) | decode_packet( ) | get_facil_CALL( ) |
| get_Abit( ) | get_facil_CLEAR( ) | get_facil_CLEARC( ) |
| get_facil_CALLC( ) | get_LCI( ) | get_LCI_ranges( ) |
| get_packet( ) | get_PR( ) | get_PS( ) |
| get_Qbit( ) | get_SN_L3( ) | get_state_L3( ) |
| get_userdata_CALL( ) | get_userdata_CALLC( ) | get_userdata_CLEAR( ) |
| get_windowsize_L3( ) | reset_channel( ) | restart_L3( ) |
| send_CALL( ) | send_CALLC( ) | send_CLEAR( ) |
| send_CLEARC( ) | send_DATA( ) | send_DIAG( ) |
| send_INT( ) | send_INTC( ) | send_REG( ) |
| send_REGC( ) | send_REJ_L3( ) | send_RESET( ) |
| send_RESETC( ) | send_RESTART( ) | send_RESTARTC( ) |
| send_RNR_L3( ) | send_RR_L3( ) | set_Abit( ) |
| set_Dbit( ) | set_facil_CALL( ) | set_facil_CALLC( ) |
| set_facil_CLEAR( ) | set_facil_CLEARC( ) | set_LCI( ) |
| set_Qbit( ) | set_userdata_CALL( ) | set_userdata_CALLC( ) |
| set_userdata_CLEAR( ) | set_windowsize_L3( ) | |

## Example

The following example shows how a combination of control string entries, argument list entires, and default values can be used with format_L3_X25( ).

```
char *packet_ptr_ptr;
int packet_length;
char *User_Data, *Called_Addr, *Calling_Addr;
char Facil[] = {0x42, 0x77};
Called_Addr = "5551212";
Calling Addr = "555132";

User_Data = "This is UserData";
format_L3_X25(&packet_ptr_ptr, &packet_length,
              "%PT%Add%D%LCI%CC%DC%CDAs%CGAs%*Fb",
             CLEAR, 0, 1, 3, 0, 0, Called_Addr, 2, Facil);
```

# get_Abit( )

| OFF | LEVEL1 | LINK CONTROL | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
| ▨ | ▨ | ▨ | ▨ | ▨ | ▨ |

## Format

```
#include <X25.include>

int get_Abit( )
```

## Description

This Level 3 function gets the default value that is placed in the TOA/NPI address format subfield of a transmitted packet. The default value is used in functions which have not passed the TOA/NPI value as a parameter. The initial value of the default Abit is `0`. It can be changed using `set_Abit( )`.

## Return Values

The TOA/NPI address format default bit is the return value.

## See Also

`set_Abit( )`

## Example

This example assigns the value of the default Abit parameter to the variable `abit`.

```
int abit;

abit = get_Abit( );
```

## get_busy_L2( )

| OFF | | LEVEL1 | LINK CONTROL | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|---|
| ▨ | | ▨ | ▨ | ▨ | ▨ | ▨ |

### Format

```
#include <dlib.include>
#include <X25.include>

int get_busy_L2( )
```

### Description

This Level 2 function returns the current LAPB busy flag value. When the LAPB busy flag is set, the emulator is forced into the busy state. The initial value of the busy flag is `0`. The flag can be set and cleared using `set_busy_L2( )` and `clear_busy_L2( )`.

### Return Values

| Value | Constant |
|---|---|
| 0 | OFF |
| 1 | ON |

### See Also

`clear_busy_L2( )`    `set_busy_L2( )`

### Example

The following example assigns the current value of the Level 2 busy flag to variable `busy_flag`.

```
int busy_flag;

busy_flag = get_busy_L2( );
```

# get_Dbit( )

| OFF | LEVEL1 | LINK_CONTROL | L2_FULL | CALL_CONTROL | L3_ALL |
|---|---|---|---|---|---|
| ▨ | ▨ | ▨ | ▨ | ▨ | ▨ |

## Format

```
#include <X25.include>

int get_Dbit( )
```

## Description

This Level 3 function gets the default value used for the Delivery bit field of a transmitted packet. The default value is only used when the Delivery bit parameter has not been passed to a function. The initial value of the default Dbit is `0`. The variable can be set and cleared using `set_Dbit( )`.

## Return Values

The Delivery bit field default is the return value.

## See Also

`set_Dbit( )`

## Example

This example assigns the current value of the default Delivery bit to variable `default_dbit`.

```
int default_dbit;

default_dbit = get_Dbit( );
```

## get_device( )

| OFF | LEVEL1 | LINK CONTROL | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
| ▨ | ▨ | ▨ | ▨ | ▨ | ▨ |

### Format

```
#include <dlib.include>
#include <X25.include>

int get_device( )
```

### Description

This Level 1 library function determines which channel is in the transmit mode and returns the device type (i.e., `DTE`, `DCE`, or `NOT_CONFIGURED`) of that channel. This function can be used at any emulation stage. The transmit device is initially set to be DTE. The device can be changed to DCE using `config_device( )`, or monitor using `DataCommC function set_channel_config( )`.

### Return Values

| Value | Constant |
|---|---|
| 0 | NOT_CONFIGURED |
| 1 | DCE |
| 2 | DTE |

### See Also

| DataCommC | X.25 |
|---|---|
| get_channel_config( ) | config_device( ) |
| set_channel_config( ) | |

### Example

In the following example assigns the device type configured for transmission into the variable `device`.

```
int device;

device = get_device( );
```

# get_extctrl( )

| OFF | LEVEL1 | LINK CONTROL | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
| ▨ | ▨ | ▨ | ▨ | ▨ | ▨ |

## Format

```
#include <dlib.include>
#include <X25.include>

int get_extctrl( )
```

## Description

This Level 2 function returns the current value of the extended control (`ext_ctrl`) variable. When extended control is `ON`, the return value is `1`; when it is `OFF`, the value is `0`. The initial value of the `ext_ctrl` variable is `OFF` (`0`). The value of the `ext_ctrl` can be changed using `config_extctrl( )`.

## Return Values

| Value | Constant |
|---|---|
| 0 | OFF |
| 1 | ON |

## See Also

`config_extctrl( )` `send_SABM( )` `send_SABME( )`

## Example

The following example assigns the current value of the extended control configuration variable to the variable `extended_control`.

```
int extended_control;

extended_control = get_extctrl( );
```

## get_extctrl( )

# get_facil_CALL()

| OFF | LEVEL1 | LINK CONTROL | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
| ▨ | ▨ | ▨ | ▨ | ▨ | ▨ |

## Format

```
#include <X25.include>

int get_facil_CALL(length,string_ptr)

unsigned int *length;
char **string_ptr;
```

## Description

This Level 3 function gets the default string length and pointer, and uses them in the facility field of CALL REQUEST or INCOMING CALL packets which are transmitted by the emulator when `call_establish( )` or `send_CALL( )` are used.

The values passed in for the `length` and `string_ptr` parameters are the addresses of an unsigned integer and a character pointer respectively. When this function executes, the character pointer will point to the current default facility field string and the unsigned integer will have the length of the string which can range between `0` and `255`.

The default facility field string for CALL REQUEST and INCOMING CALL packets is initially a null string.

`set_facil_CALL( )` can be used to set up the facility field default for CALL REQUEST and INCOMING CALL packets to the desired value.

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |

## See Also

`set_facil_CALL( )`

## Example

This example returns the facility field string and its length.

```
unsigned int length;
char *string;

get_facil_CALL(&length,&string);
```

## get_facil_CALLC( )

| OFF | LEVEL1 | LINK CONTROL | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
| ▨ | ▨ | ▨ | ▨ | ▨ | ▨ |

### Format

```
#include <X25.include>

int get_facil_CALLC(length,string_ptr)

unsigned int *length;
char **string_ptr;
```

### Description

This Level 3 function gets the default string length and pointer, and places them in the facility field of CALL ACCEPTED or CALL CONNECTED packets which are transmitted automatically by the emulator or when `send_CALLC( )` is used.

The values passed in for the `length` and `string_ptr` parameters are the addresses of an unsigned integer and a character pointer respectively. When this function executes, the character pointer will point to the current default facility field string and the unsigned integer will have the length of the string which can range between `0` and `255`.

The default facility field string for CALL ACCEPTED and CALL CONNECTED packets is initially a null string.

`set_facil_CALLC( )` can be used to set the facility field default for CALL ACCEPTED and CALL CONNECTED packets to the desired value.

### Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |

### See Also

`set_facil_CALLC( )`

## Example

This example returns the facility field string and its length.

```
unsigned int length;
char *string;

get_facil_CALLC(&length,&string);
```

## get_facil_CLEAR( )

| OFF | LEVEL1 | LINK CONTROL | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
| ▨ | ▨ | ▨ | ▨ | ▨ | ▨ |

### Format

```
#include <X25.include>

int get_facil_CLEAR(length,string_ptr)

unsigned int *length;
char **string_ptr;
```

### Description

This Level 3 function gets the default string length and pointer, and places them in the facility field of CLEAR REQUEST or CLEAR INDICATION packets which are transmitted automatically by the emulator or when `call_CLEAR( )` or `send_CLEAR( )` are used.

The values passed in for the `length` and `string_ptr` parameters are the addresses of an unsigned integer and a character pointer respectively. When this function executes, the character pointer will point to the current default facility field string and the unsigned integer will have the length of the string which can range between 0 and 255.

The default facility field string for CLEAR REQUEST or CLEAR INDICATION packets is initially a null string.

`set_facil_CLEAR( )` can be used to set the facility field default for CLEAR REQUEST and CLEAR INDICATION packets to the desired value.

### Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |

### See Also

`set_facil_CLEAR( )`

## Example

This example returns the facility field string and its length.

```
unsigned int length;
char *string;

get_facil_CLEAR(&length,&string);
```

# get_facil_CLEARC( )

| OFF | LEVEL1 | LINK CONTROL | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
| ▨ | ▨ | ▨ | ▨ | ▨ | ▨ |

## Format

```
#include <X25.include>

int get_facil_CLEARC(length,string_ptr)

unsigned int *length;
char **string_ptr;
```

## Description

This Level 3 function gets the default string length and pointer, and places them in the facility field of a CLEAR CONFIRMATION packet which is transmitted automatically by the emulator or when `send_CLEAR( )` is used.

The values passed in for the `length` and `string_ptr` parameters are the addresses of an unsigned integer and a character pointer respectively. When this function executes, the character pointer will point to the current default Facility field string and the unsigned integer will have the length of the string which can range between 0 and 255.

The default Facility field string for a CLEAR CONFIRMATION packet is initially a null string.

`set_facil_CLEAR( )` can be used to set the Facility field default for a CLEAR CONFIRMATION packet to the desired value.

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |

## See Also

`set_facil_CLEARC( )`

## Example

This example returns the facility field string and its length.

```
unsigned int length;
char *string;

get_facil_CLEARC(&length,&string);
```

## get_frame()

| | LEVEL1 | LINK CONTROL | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
| | ▨ | ▨ | ▨ | ▨ | ▨ |

### Format

```
#include <X25.include>

unsigned int get_frame(datacomm_subtype, message, timeout, frame_num, frame_types)

int datacomm_subtype;
HP_MESSAGE  *message;
unsigned long timeout;
int frame_num;
unsigned int frame_types;
```

### Description

This Level 2 Library function searches the message queue and returns the first frame that matches one of the frame types specified in the `frame_type` parameter.

Five different frame types can be listed in the `frame_types` parameter, but only one frame is returned. If there are more `frame_types` listed than `frame_num`, the rest of the frame types are ignored.

If no matching frames are found during the period specified by the `timeout` parameter, an error message is returned.

As the function searches the message queue, it discards any messages containing non-matching frame types, and also releases them from the event buffer. If no match is found, all messages read during the period defined by the `timeout` parameter are discarded from the queue and released from the event buffer.

After the function returns with the desired frame, you must release an event from the event buffer using the DataCommC function `release_event`.

## Return Values

| Value | Constant | Definition |
|---|---|---|
| | FRAME_TYPE | |
| -101 | ERROR_101 | Check parameter 1 for correct value. |
| -103 | ERROR_103 | Check parameter 3 for correct value. |
| -104 | ERROR_104 | Check parameter 4 for correct value. |
| -105 | ERROR_105 | Check parameter 5 for correct value. |
| -121 | ERROR_121 | Emulator is not in the correct stage to process this function. |
| -161 | ERROR_161 | Selected frame is not found during timeout. |
| -162 | ERROR_162 | No memory available for timer, or maximum number of timers (32,767) active. |

## See Also

get_packet( )

## Example

The following example returns either a UA or a SABM frame providing the frame is found within 1000 tenths of a second (100 seconds), otherwise it returns an error.

```
unsigned int  frame_type;
HP_MESSAGE  message;

if ((frame_type = get_frame(DCE_FRAME, &message, 1000l, 2, UA, SABM)) == UA)
   {
   printf("UA is found \n");
   release_event(UDATA_EVENT_PTR(message));
   }
else if (frame_type == SABM)
   {
   printf("SABM is found \n");
   release_event(UDATA_EVENT_PTR(message));
   }
else
   printf("ERROR:  %d \n",  frame_type);
```

## get_frame_addrs( )

| OFF | LEVEL1 | LINK CONTROL | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
| ▨ | ▨ | ▨ | ▨ | ▨ | ▨ |

### Format

```
#include <X25.include>

int get_frame_addrs(subscriber_addr, network_addr)

unsigned char *subscriber_addr, *network_addr;
```

### Description

This Level 2 library function returns the current address variables (subscriber_addr, network_addr) to the user specified addresses. The initial value for subscriber_addr is 3 and for network_addr is 1. These values can be changed using config_frame_addrs( ).

### Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |

### See Also

config_frame_addrs( )

### Example

The following example sets subscriber_addr and network_addr to the values used for the LAPB subscriber's address and network's address.

```
unsigned char subscriber_addr, network_addr;

get_frame_addrs(&subscriber_addr, &network_addr);
```

## get_k( )

| OFF | LEVEL1 | LINK CONTROL | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
| ▨ | ▨ | ▨ | ▨ | ▨ | ▨ |

### Format

```
#include <X25.include>

int get_k( )
```

### Description

This Level 2 library function returns the LAPB window size variable `k`. This function can be used at any emulation stage. `k` is used when the emulator is in stage `L2_FULL` or above.

The initial value of `k` is `7`. `k` can be changed using `config_k( )`.

### Return Values

The return value is the value of k, which is in the range of 1 through 7.

### See Also

`config_k( )`

### Example

The following example assigns the current value of `k` to the variable `window_size`.

```
int window_size;

window_size = get_k( );
```

## get_LCI()

| OFF | LEVEL1 | LINK CONTROL | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
| ▨ | ▨ | ▨ | ▨ | ▨ | ▨ |

### Format

```
#include <X25.include>

int get_LCI( )
```

### Description

This Level 3 function gets the default value used for most transmitted packets when no LCI parameter has been passed to the send function. The default LCI value is not used for RESTART REQUEST, RESTART CONFIRMATION, DIAGNOSTIC, REGISTRATION REQUEST or REGISTRATION CONFIRMATION packets. The initial default LCI is 1. The value can be changed using `set_LCI( )`.

The LCI default value can range from 1 to 4095.

### Return Values

The default LCI value is the return value.

### See Also

`set_LCI( )`

### Example

This example assigns the current value of the default LCI parameter to the variable `lci`.

```
int lci;

lci = get_LCI( );
```

# get_LCI_ranges( )

| OFF | LEVEL1 | LINK CONTROL | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
| ▨ | ▨ | ▨ | ▨ | ▨ | ▨ |

## Format

```
#include <X25.include>

int get_LCI_ranges(PVC_low, PVC_high, SVC_low, SVC_high)

unsigned int *PVC_low, *PVC_high, *SVC_low, *SVC_high;
```

## Description

This Level 3 library function gets the LCI ranges used by the Level 3 emulator. These values are only used when the emulator is in stage CALL_CONTROL or L3_ALL.

Channels in the range specified by SVC_low and SVC_high are all two-way channels.

The ranges for each PVC and SVC parameter are from Channel 0 through Channel 4095. However, the following limitations apply:

- PVC_low <= PVC_high < SVC_low <= SVC_high.
- 0 is only used to specify no PVCs or SVCs.
- To specify no PVCs, both PVC_low and PVC_high must be set to 0.
- To specify no SVCs, both SVC_low and SVC_high must be set to 0.

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |

## See Also

config_LCI_ranges( )

## Example

The following example gets the current PVC and SVC LCI ranges.

```
unsigned int PVC_low;
unsigned int PVC_high;
unsigned int SVC_low;
unsigned int SVC_high:

get_LCI_ranges(&PVC_low, &PVC_high, &SVC_low, &SVC_high);
```

# get_leads_X25()

LEVEL1

| | ▨ | | | | |
|---|---|---|---|---|---|

## Format

```
int get_leads_X25(lead_states)

int *lead_states;
```

## Description

This Level 1 library function returns the current status of the interface leads, bit-mapped into an integer value.

## Return Values

As shown below, a bit map is used to indicate lead status and is dependent on the interface pod type. The return value is 1 when the lead is ON, and 0 when the lead is OFF.

**Lead Status Bit-map**

| | Bit Number | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Interface Pod** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** |
| RS-232C, MIL-188C | RTS | CTS | DSR | DTR | RI | CD | SQ | DRS | SRS | SCS | SCD | | | | |
| V.35 | RS | CS | DSR | DTR | RI | CD | LT | | | | | | | | |
| RS-449 | RS | CS | DM | TR | IC | RR | SQ | SI | SRS | SCS | SRR | IS | SF | RL | SS |

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -121 | ERROR_121 | Emulator is not in the correct stage to process this function. |

## See Also

`leads_off_X25( )` `leads_on_X25( )`

## Example

The following example gets the current lead status based upon the interface pod type.

```
int lead_states;

get_leads_X25(&lead_states);
```

# get_logical_device( )

| OFF | LEVEL1 | LINK CONTROL | L2 FULL | CALL_CONTROL | L3 ALL |
|---|---|---|---|---|---|
| ▨ | ▨ | ▨ | ▨ | ▨ | ▨ |

## Format

```
#include <X25.include>

int get_logical_device( )
```

## Description

This Level 2 function returns the `logical_dev` variable that is the logical device configuration. This value is either `SUBSCRIBER` (0) or `NETWORK`(1).

The logical device is initially set to be `SUBSCRIBER` and can be changed using `config_logical_device( )`.

## Return Values

| Value | Constant |
|---|---|
| 0 | SUBSCRIBER |
| 1 | NETWORK |

## See Also

`config_logical_device( )`

## Example

The following example assigns the current logical device configuration to the variable `logical_device`.

```
int logical_device;

logical_device = get_logical_device( );
```

## get_N1( )

| OFF | LEVEL1 | LINK_CONTROL | L2_FULL | CALL_CONTROL | L3_ALL |
|---|---|---|---|---|---|
| ▨ | ▨ | ▨ | ▨ | ▨ | ▨ |

### Format

```
#include <X25.include>

long get_N1( )
```

### Description

This Level 2 library function returns the value of the LAPB frame size variable `N1`. The initial value of `N1` is 65535. `N1` can be changed using `config_N1( )`.

### Return Values

The return value is the `N1` value.

### See Also

`config_N1( )`

### Example

The following example assigns the value of `N1` to the variable `frame_size`.

```
long frame_size;

frame_size = get_N1( );
```

## get_N2()

| OFF | LEVEL1 | LINK CONTROL | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
| ▨ | ▨ | ▨ | ▨ | ▨ | ▨ |

### Format

```
#include <X25.include>

int get_N2( )
```

### Description

This Level 2 library function returns the total number of times (N2) that the emulator will attempt to successfully complete the transmission of a frame. This "try count" is set with `config_N2( )` and includes the initial transmission. N2 has an initial value of 20.

There is no default value.

### Return Values

The return value is the N2 value.

### See Also

`config_N2( )`

### Example

The following example assigns the value of N2 to the variable `try_count` .

```
int try_count;

try_count = get_N2( );
```

## get_NR( )

| OFF | LEVEL1 | LINK_CONTROL | L2_FULL | CALL_CONTROL | L3_ALL |
|---|---|---|---|---|---|
| ▨ | ▨ | ░ | ░ | ░ | ░ |

### Format

```
#include <X25.include>

int get_NR( )
```

### Description

This Level 2 library function returns the current receive sequence number value `Vr`. `Vr` is only used by the emulator when it is in stage `L2_FULL` and above. If the emulator is in stage `LINK_CONTROL` or below, and the emulator was not previously in `L2_FULL` or above, `Vr` is `0`. If the emulator has been staged down to `LINK_CONTROL` or below, `get_NR( )` returns the last value of `Vr` before the emulator was staged down.

### Return Values

The return value is `Vr`.

### See Also

None

### Example

The following example assigns the current receive sequence number value `Vr` to the variable `nr_value`.

```
int nr_value;

nr_value = get_NR( );
```

# get_NS()

| OFF | | LEVEL1 | LINK CONTROL | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|---|
| ▨ | | ▨ | ▨ | ▨ | ▨ | ▨ |

## Format

```
#include <X25.include>

int get_NS( )
```

## Description

This Level 3 library function returns the current send sequence number value `Vs`. `Vs` is only used by the emulator when it is in stage `L2_FULL` and above. If the emulator is in stage `LINK_CONTROL` or below, and the emulator was not previously in `L2_FULL` or above, `Vs` is `0`. If the emulator has been staged down to `LINK_CONTROL` or below, `get_NS( )` returns the last value of `Vs` before the emulator was staged down.

## Return Values

The return value is `Vs`.

## See Also

None.

## Example

The following example assigns the current receive sequence number value `Vs` to the variable `ns_value`.

```
int ns_value;

ns_value = get_NS( );
```

## get_NS( )

# get_packet( )

|  |  | LINK CONTROL | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
|  |  | ▨ | ▨ | ▨ | ▨ |

## Format

```
#include <X25.include>

unsigned int get_packet(datacomm_subtype, message, timeout, packet_num,
                        packet_types)

int datacomm_subtype;
HP_MESSAGE  *message;
unsigned long timeout;
int packet_num;
unsigned int packet_types;
```

## Description

This Level 3 Library function searches the message queue and returns the first packet that matches one of the packet types specified in the `packet_types` parameter.

Five different packet types can be listed in the `packet_types` parameter, but only one packet is returned. If there are more `packet_types` listed than `packet_num`, the rest of the packet types are ignored.

If no matching packets are found during the period specified by the `timeout` parameter, an error message is returned.

As the function searches the message queue, it discards any messages containing non-matching packet types, and also releases them from the event buffer. If no match is found, all messages read during the period defined by the `timeout` parameter are discarded from the queue and released from the event buffer.

After the function returns with the desired packet, you must release an event from the event buffer using the DataCommC function `release_event`.

## Return Values

| Value | Constant | Definition |
|---|---|---|
| | packet type | |
| -101 | ERROR_101 | Check parameter 1 for correct value. |
| -103 | ERROR_103 | Check parameter 3 for correct value. |
| -104 | ERROR_104 | Check parameter 4 for correct value. |
| -105 | ERROR_105 | Check parameter 5 for correct value. |
| -121 | ERROR_121 | Emulator is not in the correct stage to process this function. |
| -161 | ERROR_161 | Selected packet not found during `timeout`. |
| -162 | ERROR_162 | No memory available for timer, or maximum number of timers (32,767) active. |

## See Also

`get_frame( )`

## Example

The following example returns a RESTART packet providing the packet is found within 1000 tenths of a second (100 sec), otherwise it returns an error.

```
unsigned int packet_type;
HP_MESSAGE  message;

if  ((packet_type = get_packet(DCE_FRAME, &message, 1000l, 1, RESTART) == RESTART)
       /*  Release the event from the buffer.  */
       release_event(UDATA_EVENT_PTR(message));
else
       /*  Desired packet type was not found during the timeout period.  */
       printf("ERROR : %d\n", packet_type);
```

# get_PR()

| OFF | LEVEL1 | LINK CONTROL | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
| ▨ | ▨ | ▨ | ▨ | ▨ | ▨ |

## Format

```
#include <X25.include>

int get_PR(lci)

unsigned int lci;
```

## Description

This Level 3 library function returns the current emulator packet received sequence number value for the specified logical channel (`lci`).

The range for the `lci` variable is `1` through `4095`. LCI 0 is reserved for packet level information.

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0-127 | | Current `PR` value. |
| -132 | ERROR_132 | Inactive LCI; LCI is a PVC with no activity or there is not an active call on the LCI. |
| -133 | ERROR_133 | Invalid LCI; LCI is not in the PVC or SVC ranges specified by `config_LCI_ranges( )`. |

## See Also

None.

## Example

The following example assigns the current `PR` value for `lci 7` to the variable `pr`.

```
int pr;

pr = get_PR(7);
```

# get_PS( )

| OFF | LEVEL1 | LINK CONTROL | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
| ▨ | ▨ | ▨ | ▨ | ▨ | ▨ |

## Format

```
#include <X25.include>

int get_PS(lci)

unsigned int lci;
```

## Description

This Level 3 library function returns the current emulator packet sent sequence number value for the selected logical channel (`lci`).

The range for the `lci` variable is `0` through `4095`. LCI 0 is reserved for packet level information.

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0-127 | | Current `PS` value. |
| -132 | ERROR_132 | Inactive LCI; LCI is a PVC with no activity or there is not an active call on the LCI. |
| -133 | ERROR_133 | Invalid LCI; LCI is not in the PVC or SVC ranges specified by `config_LCI_ranges( )`. |

## See Also

None.

## Example

The following example assigns the current `PS` value for `lci 16` to the variable `ps`.

```
int ps;

ps = get_PS(16);
```

# get_Qbit( )

| OFF | LEVEL1 | LINK CONTROL | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
| ▨ | ▨ | ▨ | ▨ | ▨ | ▨ |

## Format

```
#include <X25.include>

int get_Qbit( )
```

## Description

This Level 3 function returns the default value that is placed in the Qualifier bit field of a transmitted packet when the Qbit parameter has not been passed to a send function.

The initial value of the default Qbit is `0` and can be changed using `set_Qbit( )`.

## Return Values

The return value is the default Qualifier bit.

## See Also

`set_Qbit( )`

## Example

This example assigns the default Qbit value to the variable `qbit`.

```
int qbit;

qbit = get_Qbit( );
```

## get_RR_idle_L2( )

| OFF | LEVEL1 | LINK CONTROL | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
| ▓ | ▓ | ░ | ░ | ░ | ░ |

### Format

```
#include <dlib.include>
#include <X25.include>

int get_RR_idle_L2( )
```

### Description

This Level 2 library function returns the current emulator Level 2 `RR_idle` flag value. When the `RR_idle` flag is `ON`, the emulator sends an RR frame with the poll bit set, every `T1` seconds of idle time.

### Return Values

| **Value** | **Idle** | **Frame** |
|---|---|---|
| 0 | `RR_idles OFF` | Send flags during idle time. |
| 1 | `RR_idles ON` | Send `RR` frames with the poll bit set each `T1` seconds of idle time. |

### See Also

`config_RR_idle_L2( )`

### Example

The following example assigns the value of the `RR_idle` flag to the variable `idle_in_RRs`.

```
int idle_in_RRs;

idle_in_RRs = get_RR_idle_L2( );
```

## get_SN_L3( )

| OFF | LEVEL1 | LINK_CONTROL | L2_FULL | CALL_CONTROL | L3_ALL |
|---|---|---|---|---|---|
| ▨ | ▨ | ▨ | ▨ | ▨ | ▨ |

### Format

```
#include <X25.include>

int get_SN_L3( )
```

### Description

This Level 3 library function returns the transmitted packet sequence number (SN) field default value. The SN field default is used by any of the Level 3 send functions which allow a SN control string input and for which the SN parameter has been omitted.

The initial SN field default value is 1 and can be changed using config_SN_L3( ).

### Return Values

| Value | Constant | Definition |
|---|---|---|
| 1 | mod_8 | Sequence Number range 0 to 7. |
| 2 | mod_128 | Sequence Number range 0 to 127. |

### See Also

config_SN_L3( )

### Example

The following example assigns the value of the sequence number variable L3mod to the variable seq_num.

```
int seq_num;

seq_num = get_SN_L3( );
```

# get_stage_X25( )

OFF
LEVEL1
LINK CONTROL
L2 FULL
CALL CONTROL
L3 ALL

## Format

```
#include <dlib.include>
#include <X25.include>

int get_stage_X25( )
```

## Description

This general library function returns the current emulator stage.

## Return Values

| Value | Emulator Stage |
|---|---|
| 0 | OFF |
| 0x11 | LEVEL1 |
| 0x22 | LINK_CONTROL |
| 0x32 | L2_FULL |
| 0x43 | CALL_CONTROL |
| 0x53 | L3_ALL |

## See Also

emulate_X25( )

## Example

The following example assigns the current emulator stage to the variable `stage`.

```
int  stage;

stage = get_stage_X25( );
```

# get_state_L2()

| OFF | LEVEL1 | LINK_CONTROL | L2_FULL | CALL_CONTROL | L3_ALL |
|---|---|---|---|---|---|
| ▨ | ▨ | ▨ | ▨ | ▨ | ▨ |

## Format

```
#include <X25.include>

int get_state_L2( )
```

## Description

This Level 2 function returns the current Level 2 emulator state when in LINK_CONTROL through L3_ALL emulation stages. When the emulator is in the LEVEL1 or OFF stage, the return value is the *last* emulator state.

## Return Values

All emulator stages given below are minimum and above.

| Value | Min. Stage | Emulator State | Description |
|---|---|---|---|
| 1 | LINK_CONTROL | DISC_PHASE | Disconnect Phase (DP) |
| 2 | LINK_CONTROL | LINK_DISC | Link Disconnected (LD) |
| 3 | LINK_CONTROL | LINK_SETUP | Link Setup (LS) |
| 4 | LINK_CONTROL | FRMR_STATE | FRMR sent (FR) |
| 5 | LINK_CONTROL | NORMAL_DT | Normal Information Transfer (N) |
| 6 | LINK_CONTROL | LOCAL_BUSY | Local Station Busy (LSB) |
| 7 | L2_FULL | REMOTE_BUSY | Remote Station Busy (RSB) |

| Value | Min. Stage | Emulator State | Description |
|---|---|---|---|
| 8 | L2_FULL | BOTH_BUSY | Both Stations Busy (BB) |
| 9 | L2_FULL | WAIT_ACK | Waiting for Acknowledgment (WA) |
| 10 | L2_FULL | WAIT_ACK_LB | Waiting for Acknowledgment and Local Station Busy (WA/LB) |
| 11 | L2_FULL | WAIT_ACK_RB | Waiting for Acknowledgment and Remote Station Busy (WA/RB) |
| 12 | L2_FULL | WAIT_ACK_BB | Waiting for Acknowledgment and Both Stations Busy (WA/BB) |
| 13 | L2_FULL | REJ_SENT | REJ sent (RJ) |
| 14 | L2_FULL | REJ_SENT_LB | REJ sent and Local Station Busy (RJ/LB) |
| 15 | L2_FULL | REJ_SENT_RB | REJ sent and Remote Station Busy (RJ/RB) |
| 16 | L2_FULL | REJ_SENT_BB | REJ sent and Both Stations Busy (RJ/BB) |

## See Also

None.

## Example

The following example assigns the current state of the LAPB emulator to the variable `l2_state`.

```
int l2_state;

l2_state = get_state_L2( );
```

# get_state_L3()

| OFF | LEVEL1 | LINK CONTROL | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
| ▨ | ▨ | ▨ | ▨ | ▨ | ▨ |

## Format

```
#include <X25.include>

int get_state_L3(lci)

unsigned int lci;
```

## Description

This Level 3 function returns the current state of the channel having the specified `lci` when the emulator is in the `CALL_CONTROL` or `L3_ALL` stage. When the emulator is in stage `L2_FULL` or below, the return value is the *last* emulator state.

To return the state of the packet layer, specify LCI `0`.

## Return Values

### LCI 0 Packet Level State

| | |
|---|---|
| 1 | PACKET_LEVEL_READY |
| 2 | DTE_RESTART_REQ |
| 3 | DCE_RESTART_IND |

### LCI 1 - 4095 Packet Level State

| | |
|---|---|
| 4 | READY |
| 5 | DTE_CALL_REQ |
| 6 | DCE_CALL_IND |
| 7 | CALL_COLLISION |
| 8 | DTE_CLEAR_REQ |
| 9 | DEC_CLEAR_IND |
| 10 | DTE_RESET_REQ |
| 11 | DCE_RESET_IND |
| 12 | DT_REMOTE_LCI_NOT_BUSY |
| 13 | DT_REMOTE_LCI_BUSY |

| Value | Constant | Definition |
|---|---|---|
| 1-13 | | Current state of specific LCI. |
| -133 | ERROR_133 | Invalid LCI; LCI is not in the PVC or SVC ranges specified by config_LCI_ranges( ). |

## See Also

config_LCI_ranges( )

## Example

The following example assigns the current state of logical channel 2 to the variable lci_3_state.

```
int lci_3_state;

lci_3_state = get_state_L3(2);
```

# get_T1( )

| OFF | LEVEL1 | LINK CONTROL | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
| ▨ | ▨ | ▨ | ▨ | ▨ | ▨ |

## Format

```
#include <X25.include>

long get_T1( )
```

## Description

This Level 2 library function returns the time (`T1`), in tenths of a second, that the Level 2 emulator waits after a frame is transmitted, without a corresponding acknowledgment before recovery procedures begin. `T1` has an initial value of `30/10` (3 seconds) and can be set to a different value using `config_T1( )`.

## Return Values

The return value is `T1`.

## See Also

`config_T1( )`

## Example

The following example assigns the value of `T1` to the variable `t1_timer`.

```
long t1_timer;

t1_timer = get_T1( );
```

## get_T1( )

## get_userdata_CALL( )

| OFF | LEVEL1 | LINK CONTROL | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
| ▨ | ▨ | ░ | ░ | ▨ | ▨ |

### Format

```
#include <X25.include>

int get_userdata_CALL(length,string_ptr)

unsigned int *length;
char **string_ptr;
```

### Description

This Level 3 function gets the default string length and pointer, and places them in the Call User Data field of CALL REQUEST or INCOMING CALL packets which are transmitted by the emulator when `call_establish( )` or `send_CALL( )` are used.

The values passed in for the `length` and `string_ptr` parameters are the addresses of an unsigned integer and a character pointer respectively. When this function executes, the character pointer will point to the current default Call User Data field string and the unsigned integer will have the length of the string which can range between 0 and 255.

The default Call User Data field string for CALL REQUEST and INCOMING CALL packets is initially a null string.

### Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |

### See Also

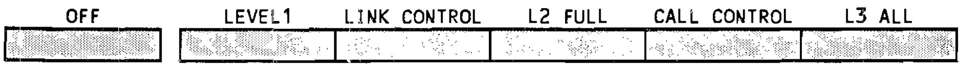
`set_userdata_CALL( )`

## Example

This example returns the Call User Data string pointer and its length.

```
int length;
char *string;

get_userdata_CALL(&length,&string);
```

# get_userdata_CALLC( )



## Format

```
#include <X25.include>

int get_userdata_CALLC(length,string_ptr)

unsigned int *length;
char **string_ptr;
```

## Description

This Level 3 function gets the default string length and pointer, and places them in the Called User Data field of CALL ACCEPTED or CALL CONNECTED packets which are transmitted automatically by the emulator or when `send_CALLC( )` is used.

The values passed in for the `length` and `string_ptr` parameters are the addresses of an unsigned integer and a character pointer respectively. When this function executes, the character pointer will point to the current default Call User Data field string and the unsigned integer will have the length of the string which can range between 0 and 255.

The default Call User Data field string for CALL ACCEPTED or CALL CONNECTED packets is initially a null string.

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |

## See Also

`set_userdata_CALLC( )`

## Example

This example returns the Call User Data string pointer and its length.

```
int length;
char *string;

get_userdata_CALLC(&length,&string);
```

# get_userdata_CLEAR( )

| OFF | LEVEL1 | LINK CONTROL | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
| | | | | | |

## Format

```
#include <X25.include>

int get_userdata_CLEAR(length,string_ptr)

unsigned int *length;
char **string_ptr;
```

## Description

This Level 3 function gets the default string length and pointer, and places them in the Clear User Data field of CLEAR REQUEST or CLEAR INDICATION packets which are transmitted by the emulator or when `call_clear( )` or `send_CLEAR( )` are used.

The values passed in for the `length` and `string_ptr` parameters are the addresses of an unsigned integer and a character pointer respectively. When this function executes, the character pointer will point to the current default Clear User Data field string and the unsigned integer will have the length of the string which can range between 0 and 255.

The default Clear User Data field string for CLEAR REQUEST or CLEAR INDICATION packets is initially a null string.

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |

## See Also

`set_userdata_CLEAR( )`

## Example

This example returns the Clear User Data string pointer and its length.

```
int length;
char *string;

get_userdata_CLEAR(&length,&string);
```

# get_windowsize_L3()

| OFF | LEVEL1 | LINK CONTROL | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
| ▨ | ▨ | ▨ | ▨ | ▨ | ▨ |

## Format

```
#include <X25.include>

int get_windowsize_L3( )
```

## Description

This Level 3 function returns the current transmit window size used for all LCIs. The transmit window size has an initial value of 2.

## Return Values

The Level 3 transmit window size is the return value.

## See Also

`set_windowsize_L3( )`

## Example

This example assigns the Level 3 transmit window size to the variable `transmit_window`.

```
int transmit_window;

transmit_window = get_windowsize_L3( );
```

# leads_off_X25( )

| | LEVEL1 | LINK CONTROL | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
| | ▨ | ▨ | ▨ | ▨ | ▨ |

## Format

```
#include <X25.include>

int leads_off_X25( )
```

## Description

This Level 1 library function turns the Level 1 control interface leads OFF.

| RS232/MIL188C | | RS449 | | V.35 | | X.21 | |
|---|---|---|---|---|---|---|---|
| DTE | DCE | DTE | DCE | DTE | DCE | DTE | DCE |
| DTR | CD | RS | CS | CS | DSR | C | I |
| RTS | CTS | TR | DM | DTR | RLSD | | |
| | DSR | | RR | RS | | | |

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -121 | ERROR_121 | Emulator is not in the correct stage to process this function. |
| -122 | ERROR_122 | Front end is not configured properly. |
| -129 | ERROR_129 | Incorrect pod attached. |

## See Also

`config_device( )`    `leads_on_X25( )`

## Example

The following example sets the appropriate Level 1 control leads OFF.

```
leads_off_X25( );
```

# leads_on_X25()

| | LEVEL1 | LINK CONTROL | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
| | ▨ | ▨ | ▨ | ▨ | ▨ |

## Format

```
#include <X.25.include>

int leads_on_X25( )
```

## Description

This Level 1 library function turns the Level 1 control interface leads ON.

| RS232/MIL188C | | RS449 | | V.35 | | X.21 | |
|---|---|---|---|---|---|---|---|
| DTE | DCE | DTE | DCE | DTE | DCE | DTE | DCE |
| DTR | CD | RS | CS | CS | DSR | C | I |
| RTS | CTS | TR | DM | DTR | RLSD | | |
| | DSR | | RR | RS | | | |

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -121 | ERROR_121 | Emulator is not in the correct stage to process this function. |
| -122 | ERROR_122 | Front end is not configured properly. |
| -129 | ERROR_129 | Incorrect pod attached. |

## See Also

`config_device( )`  `leads_off_X25( )`

## Example

The following example sets the appropriate Level 1 control leads ON.

```
leads_on_X25( );
```

## link_down_X25()

| | | | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
| | | | ▨ | ▨ | ▨ |

### Format

```
        int link_down_X25( )
```

### Description

This Level 2 library function instructs the Level 2 emulator to initiate the link disconnect procedure.

`link_down_X25( )` does not check to ensure that the link has been brought down. A SUCCESSFUL return value only indicates that the link disconnect procedure has been started.

### Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -121 | ERROR_121 | Emulator is not in the correct stage to process this function. |

### See Also

`link_up_X25( )`

### Example

The following examples illustrate three ways to use `link_down_X25( )`.

This example simply starts the link disconnect procedure and no check is made to ensure that the link has been brought down. Any processing done immediately following the call to `link_down_X25( )` should not require the LAPB emulator to be in the Disconnected Phase state.

```
        link_down_X25( );
```

Although the following example does not include a check to ensure that the link has been brought down, `wait( )` is called to allow some time to disconnect the link before additional processing takes place.

```
        link_down_X25( );
        wait(get_T1());
```

In the following example, a check is made to ensure that the emulator is in the Disconnected Phase state before the program continues.

```
      HP_MESSAGE message;

      link_down_X25();
      set_timer(get_T1() * get_N2());
      do
      {
          read_message(&message,WAIT,0L);

      } while((X25_TYPE(message) != TIMER) &&
              (get_state_L2() != DISC_PHASE));
```

# link_up_X25()

|  |  |  | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
|  |  |  | ▨ | ▨ | ▨ |

## Format

```
int link_up_X25( )
```

## Description

This Level 2 library function instructs the Level 2 emulator to initiate the link set up procedure.

`link_up_X25( )` does not check to ensure that the link has been brought up. A SUCCESSFUL return value only indicates that the link up procedure has been started.

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -121 | ERROR_121 | Emulator is not in the correct stage to process this function. |

## See Also

`get_state_L2( )`  `link_down_X25( )`

## Example

The following examples illustrate three ways to use `link_up_X25( )`. If an INFORMATION frame is to be sent immediately after the link is established, it is advisable to use either the second or third example as a model.

This example simply starts the link up procedure. No check is made to ensure that the link has been brought up and no INFORMATION frames are sent.

```
link_up_X25( );
```

Although a check is not included in the following example to ensure that the link has been brought up, `wait( )` is called to allow some time to establish the link before the INFORMATION frame is transmitted. This method can be used in cases where the link is expected to come up with no problems.

```
        link_up_X25( );
        wait(get_T1());
        send_RESTART("");
```

In the following example, a check is made to ensure that the emulator is in the Normal INFORMATION Transfer state before an INFORMATION frame is transmitted.

```
      HP_MESSAGE message;

      link_up_X25();
      set_timer(get_T1() * get_N2());
      do
      {
          read_message(&message,WAIT,0L);

      } while((X25_TYPE(message) != TIMER) &&
              (get_state_L2() != NORMAL_DT));

      if (get_state_L2() == NORMAL_DT)
          send_RESTART("");
```

## reset_channel( )

|  |  |  |  | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
|  |  |  |  | ▨ | ▨ |

### Format

```
#include <X25.include>

int reset_channel(control, args)
```

### Description

This Level 3 library function instructs the X.25 emulator to initiate the procedure for resetting a channel.

### Conversion Specifications

Packet fields are defined by a control string and the associate argument list which are described in detail in chapter 8, "Frame and Packet Formatting," *HP 18321A X.25 Test Environment User's Guide*. The following table defines the conversion specifications for this function.

| Specification | Use | Default Value |
|---|---|---|
| `%[n]CC` | Define the reason for resetting the channel. | `0` |
| `%[n]DC` | An *optional entry* that contains more information about the reason for resetting the channel. | None |
| `%[n]LCI` | Defines the logical channel to reset. | `LCI_value` |

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -121 | ERROR_121 | Emulator is not in the correct stage to process this function. |
| -122 | ERROR_122 | Front end is not configured properly. |
| -123 | ERROR_123 | An error was found in the control string. |
| -124 | ERROR_124 | One of the control string parameters has an illegal value. |
| -125 | ERROR_125 | One of the control string parameters was entered more than once. |
| -131 | ERROR_131 | Front end is not running; `stop_data( )` may have been called. |
| -141 | ERROR_141 | LAPB window is full. |
| -142 | ERROR_142 | LAPB emulator is in the wrong state. |
| -144 | ERROR_144 | LCI is wrong state. |
| -145 | ERROR_145 | LCI is invalid. |

## See Also

`config_SN_L3( )` `get_SN_L3( )` `set_Abit( )` `set_LCI_bit( )`

## Example

The following example shows how a combination of control string entries, arguments, and default values can be used with `reset_channel( )`.

```
reset_channel("%LCI%CC%1DC", 3, 2);
```

# restart_L3( )

CALL CONTROL　L3 ALL

## Format

```
#include <X25.include>

int restart_L3(control, args)
```

## Description

This Level 3 library function instructs the X.25 emulator to restart Level 3.

## Conversion Specifications

Packet fields are defined by a control string and the associate argument list which are described in detail in chapter 8, "Frame and Packet Formatting," *HP 18321A X.25 Test Environment User's Guide*. The following table defines the conversion specifications for this function.

| Specification | Use | Default Value |
|---|---|---|
| `%[n]CC` | Define the reason for resetting the channel. | 0 |
| `%[n]DC` | An *optional entry* that contains more information about the reason for resetting the channel. | None |

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -121 | ERROR_121 | Emulator is not in the correct stage to process this function. |
| -122 | ERROR_122 | Front end is not configured properly. |
| -123 | ERROR_123 | An error was found in the control string. |
| -124 | ERROR_124 | One of the control string parameters has an illegal value. |
| -125 | ERROR_125 | One of the control string parameters was entered more than once. |
| -131 | ERROR_131 | Front end is not running; `stop_data( )` may have been called. |
| -141 | ERROR_141 | LAPB window is full. |
| -142 | ERROR_142 | LAPB emulator is in the wrong state. |
| -144 | ERROR_144 | LCI is in the wrong state. |

## See Also

`config_SN_L3( )`　　`get_SN_L3( )`

## Example

The following example shows how a combination of control string entries, arguments, and default values can be used with `restart_L3( )`.

```
restart_L3("%CC%0DC", 3);
```

# send_CALL()

## Format

```
#include <X25.include>

int send_CALL(control, args)

char *control;
char *args;
```

## Description

This Level 3 function sends a formatted Level 3 CALL REQUEST or INCOMING CALL packet.

## Conversion Specifications

Packet fields are defined by a control string and the associate argument list which are described in detail in chapter 8, "Frame and Packet Formatting," *HP 18321A X.25 Test Environment User's Guide*. The following table defines the conversion specifications for this function.

| Specification | Use | Default Value |
|---|---|---|
| `%[n]A` | Implement TOA/NPI address format. | `A_Bit value` |
| `%[n]CDAb` or<br>`%[n]CDAs` | Define the length (0-15 or 0-255) and location of the called DTE address data. | Length = 0<br>Data = None |
| `%[n]CGAb` or<br>`%[n]CGAs` | Define the length (0-15 or 0-255) and location of the calling DTE address data. | Length = 0<br>Data = None |
| `%[n]D` | Specify Level 3 delivery confirmation requirement. (0 or 1; OFF or ON). | 0 |

| Specification | Use | Default Value |
|---|---|---|
| %[n]Fb | Define the length (0-255) and location of the Level 3 facilities data. | Length = facil_CALL_length<br>Data = facil_CALL |
| %[n]LCI | Select a logical channel (1-4095). | LCI_value |
| %[n]UDb or<br>%[n]UDs | Define the length and location of Level 3 user data. | Length = userdata_CALL_length<br>Data = userdata_CALL |

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -121 | ERROR_121 | Emulator is not in the correct stage to process this function. |
| -122 | ERROR_122 | Front end is not configured properly. |
| -123 | ERROR_123 | An error was found in the control string. |
| -124 | ERROR_124 | One of the control string parameters has an illegal value. |
| -125 | ERROR_125 | One of the control string parameters was entered more than once. |
| -126 | ERROR_126 | Change the send string length to be between 2 bytes and 4,106 bytes. |
| -131 | ERROR_131 | Front end is not running; stop_data( ) may have been called. |
| -131 | ERROR_141 | LAPB window is full. |
| -142 | ERROR_142 | LAPB emulator is in the wrong state. |
| -143 | ERROR_143 | LCI window full. |
| -144 | ERROR_144 | LCI in wrong state. |
| -145 | ERROR_145 | LCI is invalid. |
| -162 | ERROR_162 | Mixed use of CGAs and CGAb in the same control string. |
| -163 | ERROR_163 | Mixed use of CDAs and CDAb in the same control string. |
| -164 | ERROR_164 | String length for one or more of the following character specifications too long: CDAb, CDAs, CGAb, CGAs, or Fb. |
| -165 | ERROR_165 | One of the elements of the string pointed to by CDAs or CGAs is not a digit. |
| -167 | ERROR_167 | %Q, which is illegal when PT is CALL, CALLC, CLEAR, CLEARC, REGR or REGC, was entered. |

## send_CALL( )

**See Also** send_CALLC( ) , sendf_X25( )

**Example**

```
send_CALL("%4LCI%CDAs%CGAs", "5551212", "4441313");
```

# send_CALLC()

L2 FULL

## Format

```
#include <X25.include>

int send_CALLC(control, args)

char *control;
char *args;
```

## Description

This Level 3 function sends a formatted Level 3 CALL ACCEPTED or CALL CONNECTED packet.

## Conversion Specifications

Packet fields are defined by a control string and the associate argument list which are described in detail in chapter 8, "Frame and Packet Formatting," *HP 18321A X.25 Test Environment User's Guide*. The following table defines the conversion specifications for this function.

| Specification | Use | Default Value |
|---|---|---|
| `%[n]A` | Implement TOA/NPI address format. | `A_Bit value` |
| `%[n]CDAb` or<br>`%[n]CDAs` | Define the length (0-15 or 0-255) and location of the called DTE address data. | Length = 0<br>Data = None |
| `%[n]CGAb` or<br>`%[n]CGAs` | Define the length (0-15 or 0-255) and location of the calling DTE address data. | Length = 0<br>Data = None |
| `%[n]D` | Specify Level 3 delivery confirmation requirement. (0 or 1; OFF or ON). | 0 |

# send_CALLC( )

| Specification | Use | Default Value |
|---|---|---|
| %[n]Fb | Define the length (0-255) and location of the Level 3 facilities data. | Length = facil_CALLC_length<br>Data = facil_CALLC |
| %[n]LCI | Select a logical channel (1-4095). | LCI_value |
| %[n]UDb or<br>%[n]UDs | Define the length and location of Level 3 user data. | Length = userdata_CALLC_length<br>Data = userdata_CALLC |

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -121 | ERROR_121 | Emulator is not in the correct stage to process this function. |
| -122 | ERROR_122 | Front end is not configured properly. |
| -123 | ERROR_123 | An error was found in the control string. |
| -124 | ERROR_124 | One of the control string parameters has an illegal value. |
| -125 | ERROR_125 | One of the control string parameters was entered more than once. |
| -126 | ERROR_126 | Change the send string length to be between 2 bytes and 4,106 bytes. |
| -131 | ERROR_131 | Front end is not running; stop_data( ) may have been called. |
| -141 | ERROR_141 | LAPB window full. |
| -142 | ERROR_142 | LAPB emulator in the wrong state. |
| -162 | ERROR_162 | Mixed use of CGAs and CGAb in the same control string. |
| -163 | ERROR_163 | Mixed use of CDAs and CDAb in the same control string. |
| -164 | ERROR_164 | String length for one or more of the following character specifications too long: CDAb, CDAs, CGAb, CGAs, or Fb. |
| -165 | ERROR_165 | One of the elements of the string pointed to by CDAs or CGAs is not a digit. |
| -167 | ERROR_167 | %Q, which is illegal when PT is CALL, CALLC, CLEAR, CLEARC, REGR or REGC, was entered. |

## See Also

send_CALL( )

## Example

```
send_CALLC("%4LCI");
```

# send_CLEAR()

| | | | | L2 FULL | | |
|---|---|---|---|---|---|---|
| | | | | ▨ | | |

## Format

```
#include <X25.include>

int send_CLEAR(control, args)

char *control;
char *args;
```

## Description

This Level 3 function sends a formatted Level 3 CLEAR REQUEST or CLEAR INDICATION packet.

## Conversion Specifications

Packet fields are defined by a control string and the associate argument list which are described in detail in chapter 8, "Frame and Packet Formatting," *HP 18321A X.25 Test Environment User's Guide*. The following table defines the conversion specifications for this function.

| Specification | Use | Default Value |
|---|---|---|
| `%[n]A` | Implement TOA/NPI address format. | `A_Bit value` |
| `%[n]CC` | Reason for clearing the call (0-255). | 0 |
| `%[n]CDAb` or `%[n]CDAs` | Define the length (0-15 or 0-255) and location of the called DTE address data. | Length = 0<br>Data = None |
| `%[n]CGAb` or `%[n]CGAs` | Define the length (0-15 or 0-255) and location of the calling DTE address data. | Length = 0<br>Data = None |

| Specification | Use | Default Value |
|---|---|---|
| %[n]DC | Level 3 Diagnostic Code (0-255). | None |
| %[n]Fb | Define the length (0-255) and location of the Level 3 facilities data. | Length = facil_CLEAR_length<br>Data = facil_CLEAR |
| %[n]LCI | Select a logical channel (1-4095). | LCI_value |
| %[n]UDb or<br>%[n]UDs | Define the length and location of Level 3 user data. | Length = userdata_CLEAR_length<br>Data = userdata_CLEAR |

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -121 | ERROR_121 | Emulator is not in the correct stage to process this function. |
| -122 | ERROR_122 | Front end is not configured properly. |
| -123 | ERROR_123 | An error was found in the control string. |
| -124 | ERROR_124 | One of the control string parameters has an illegal value. |
| -125 | ERROR_125 | One of the control string parameters was entered more than once. |
| -126 | ERROR_126 | Change the send string length to be between 2 bytes and 4,106 bytes. |
| -131 | ERROR_131 | Front end is not running; stop_data( ) may have been called |
| -141 | ERROR_141 | LAPB window full. |
| -142 | ERROR_142 | LAPB emulator in the wrong state. |
| -162 | ERROR_162 | Mixed use of CGAs and CGAb in the same control string. |
| -163 | ERROR_163 | Mixed use of CDAs and CDAb in the same control string. |
| -164 | ERROR_164 | String length for one or more of the following character specifications too long: CDAb, CDAs, CGAb, CGAs, or Fb. |
| -165 | ERROR_165 | One of the elements of the string pointed to by CDAs or CGAs is not a digit. |
| -167 | ERROR_167 | %Q, which is illegal when PT is CALL, CALLC, CLEAR, CLEARC, REGR or REGC, was entered. |

## send_CLEAR( )

**See Also** `send_CLEARC( )`

**Example**

```
send_CLEAR("%4LCI");
```

## send_CLEAR()

## send_CLEARC()

L2 FULL

### Format

```
#include <X25.include>

int send_CLEARC(control, args)

char *control;
char *args;
```

### Description

This Level 3 function sends a formatted Level 3 CLEAR CONFIRMATION packet.

### Conversion Specifications

Packet fields are defined by a control string and the associate argument list which are described in detail in chapter 8, "Frame and Packet Formatting," *HP 18321A X.25 Test Environment User's Guide*. The following table defines the conversion specifications for this function.

| Specification | Use | Default Value |
|---|---|---|
| `%[n]A` | Implement TOA/NPI address format. | `A_Bit value` |
| `%[n]CDAb` or `%[n]CDAs` | Define the length (0-15 or 0-255) and location of the called DTE address data. | Length = 0<br>Data = None |
| `%[n]CGAb` or `%[n]CGAs` | Define the length (0-15 or 0-255) and location of the calling DTE address data. | Length = 0<br>Data = None |
| `%[n]Fb` | Define the length (0-255) and location of the Level 3 facilities data. | Length = `facil_CLEARC_length`<br>Data = `facil_CLEARC` |
| `%[n]LCI` | Select a logical channel (1-4095). | `LCI_value` |

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -121 | ERROR_121 | Emulator is not in the correct stage to process this function. |
| -122 | ERROR_122 | Front end is not configured properly. |
| -123 | ERROR_123 | An error was found in the control string. |
| -124 | ERROR_124 | One of the control string parameters has an illegal value. |
| -125 | ERROR_125 | One of the control string parameters was entered more than once. |
| -126 | ERROR_126 | Change the send string length to be between 2 bytes and 4,106 bytes. |
| -131 | ERROR_131 | Front end is not running; `stop_data( )` may have been called. |
| -141 | ERROR_141 | LAPB window is full. |
| -142 | ERROR_142 | LAPB emulator is in the wrong state. |
| -162 | ERROR_162 | Mixed use of `CGAs` and `CGAb` in the same control string. |
| -163 | ERROR_163 | Mixed use of `CDAs` and `CDAb` in the same control string. |
| -164 | ERROR_164 | String length for one or more of the following character specifications too long: `CDAb`, `CDAs`, `CGAb`, `CGAs`, or `Fb`. |
| -165 | ERROR_165 | Either CDAs or CGAs points to a string containing a non-character. |

## See Also

`send_CLEAR( )`

## Example

```
send_CLEARC("%4LCI");
```

# send_DATA( )

|  |  |  | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
|  |  |  | ▨ | ▨ | ▨ |

## Format

```
#include <X25.include>

int send_DATA(PR, PS, control, args)

int PR, PS;
char *control;
char *args;
```

## Description

This Level 3 function sends a formatted Level 3 DATA packet. send_DATA( ) sends the packet to the Level 3 emulator if it is on; otherwise it sends the packet to the Level 2 emulator.

The first two parameters, PR and PS, represent the Pr and Ps counts, respectively. PR and PS are integer variables which may contain the following values:

|  |  |
|---|---|
| 0-7 | Mod8 sequence number range. |
| 0-127 | Mod128 sequence number range |

See config_SN_L3( ) for more information on sequence number configurations.

---

**Note** When the emulator is in the L3_ALL stage, the emulator automatically sets PR and PS. PR and PS should be set to UNSPECIFIED.

When in either L2_FULL or CALL_CONTROL, the constant UNSPECIFIED sets PR or PS to the default value 0.

---

## Conversion Specifications

Packet fields are defined by a control string and the associate argument list which are described in detail in chapter 8, "Frame and Packet Formatting," *HP 18321A X.25 Test Environment User's Guide*. The following table defines the conversion specifications for this function.

| Specification | Use | Default Value |
|---|---|---|
| `%[n]D` | Specify Level 3 delivery confirmation requirement. (0 or 1; OFF or ON). | `D_bit` |
| `%[n]LCI` | Select a logical channel (1-4095). | `LCI_value` |
| `%[n]M` | Signify requirement for another `DATA` packet (0 or 1; OFF or ON). | `0 (OFF)` |
| `%[n]PR` or<br>`%[n]PS` | Define Level 3 receive (`PR`) or send (`PS`) sequence number count (0-7 or 0-127). | `0` |
| `%[n]Q` | Specify Level 3 qualification requirement (0 or 1; OFF or ON). | `Q_bit` value |
| `%[n]UDb` or<br>`%[n]UDs` | Define the length and location of Level 3 user data. | Length = 0<br>Data = None |

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -101 | ERROR_101 | Check parameter 1 for correct value. |
| -102 | ERROR_102 | Check parameter 2 for correct value. |
| -121 | ERROR_121 | Emulator is not in the correct stage to process this function. |
| -122 | ERROR_122 | Front end is not configured properly. |
| -123 | ERROR_123 | An error was found in the control string. |
| -124 | ERROR_124 | One of the control string parameters has an illegal value. |
| -125 | ERROR_125 | One of the control string parameters was entered more than once. |
| -126 | ERROR_126 | Change the send string length to be between 2 bytes and 4,106 bytes. |
| -131 | ERROR_131 | Front end is not running; `stop_data( )` may have been called. |
| -141 | ERROR_141 | LAPB window is full. |
| -142 | ERROR_142 | LAPB emulator is in the wrong state. |
| -143 | ERROR_143 | LCI window is full. |
| -144 | ERROR_144 | LCI in wrong state. |
| -145 | ERROR_145 | LCI is invalid. |
| -161 | ERROR_161 | Extended control is ON. |

## See Also

None.

## Example

```
send_DATA(5,10,"%Q%0D%3LCI%M%UDs",1,1,"Hello");
```

## send_DATA( )

# send_DIAG()

|  |  |  | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
|  |  |  | ▨ | ▨ | ▨ |

## Format

```
#include <X25.include>

int send_DIAG(control, args)

char *control;
char *args;
```

## Description

This Level 3 function sends a formatted Level 3 DIAGNOSTIC packet.

## Conversion Specifications

Packet fields are defined by a control string and the associate argument list which are described in detail in chapter 8, "Frame and Packet Formatting," *HP 18321A X.25 Test Environment User's Guide*. The following table defines the conversion specifications for this function.

| Specification | Use | Default Value |
|---|---|---|
| `%[n]DC` | Level 3 Diagnostic Code (0-255). | 0 |
| `%[n]DEb` | Define the length (0-255) and location of the Level 3 Diagnostic Explanation data. | Length = 0<br>Data = None |

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -121 | ERROR_121 | Emulator is not in the correct stage to process this function. |
| -122 | ERROR_122 | Front end is not configured properly. |
| -123 | ERROR_123 | An error was found in the control string. |
| -124 | ERROR_124 | One of the control string parameters has an illegal value. |
| -125 | ERROR_125 | One of the control string parameters was entered more than once. |
| -131 | ERROR_131 | Front end is not running; `stop_data( )` may have been called. |
| -141 | ERROR_141 | LAPB window is full. |
| -142 | ERROR_142 | LAPB emulator is in the wrong state. |
| -143 | ERROR_143 | LCI window full. |
| -144 | ERROR_144 | LCI in wrong state. |
| -145 | ERROR_145 | LCI is invalid. |

## See Also

None

## Example

```
send_DIAG("%16DC");
```

# send_DISC( )

LEVEL1

## Format

```
#include <X25.include>

int send_DISC(poll)
int poll;
```

## Description

This Level 2 library function instructs the emulator to send a DISC (disconnect) frame. The PF bit is set to the value defined by poll. The values for poll are as follows. There is no default value.

|  |  |
|---|---|
| 0 = POLL_0 | 1 = POLL_1 |

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -101 | ERROR_101 | Check parameter 1 for correct value. |
| -121 | ERROR_121 | Emulator is not in the correct stage to process this function. |
| -122 | ERROR_122 | Front end is not configured properly. |
| -131 | ERROR_131 | Front end is not running; stop_data( ) may have been called. |

## See Also

config_extctrl( )    get_extctrl( )

## Example

The following example sends a DISC frame, with the poll bit set, when the emulation stage is LEVEL1.

```
if(get_stage_X25( ) == LEVEL1)
     send_DISC(POLL_1);
```

## send_DM( )

LEVEL1

### Format

```
#include <X25.include>

int send_DM(final)
int final;
```

### Description

This Level 2 library function instructs the emulator to send a DM (disconnect) frame. The PF bit is set to the value defined by `final`. The values for `final` are as follows. There is no default value.

```
0 = FINAL_0                1 = FINAL_1
```

### Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -101 | ERROR_101 | Check parameter 1 for correct value. |
| -121 | ERROR_121 | Emulator is not in the correct stage to process this function. |
| -122 | ERROR_122 | Front end is not configured properly. |
| -131 | ERROR_131 | Front end is not running; `stop_data( )` may have been called. |

### See Also

`config_extctrl( )`  `get_extctrl( )`

### Example

The following example sends a DM frame, with the `final` bit set, when the emulation stage is LEVEL1.

```
if(get_stage_X25( ) == LEVEL1)
    send_DM(FINAL_1);
```

# send_FRMR()

LEVEL1

| | | | | | |
|---|---|---|---|---|---|
| | ▨ | | | | |

## Format

```
#include <X25.include>

int send_FRMR(final, rej_ctrl, VR, VS, CR, W, X, Y, Z)

int final, rej_ctrl, VR, VS;
unsigned char CR, W, X, Y, Z;
```

## Description

This Level 2 library function instructs the emulator to send a FRMR frame using the following parameters. None of these parameters has a default value.

| Parameter | Function | Values |
|---|---|---|
| final | The final bit. | 0 = FINAL_0<br>1 = FINAL_1 |
| rej_ctrl | Rejected Frame Control Field | |
| | Normal Control Field | 0 - 0xFF |
| | Extended Control Field | 0 - FFFF |
| VR | Receive State Variable | |
| | Normal Control Field | 0 - 7 |
| | Extended Control Field | 0 - 127 |
| VS | Send State Variable | |
| | Normal Control Field | 0 - 7 |
| | Extended Control Field | 0 - 127 |
| CR | COMMAND bit | 256 |
| | RESPONSE bit | 257 |
| W | W bit | 0 or 1 |
| X | X bit | 0 or 1 |
| Y | Y bit | 0 or 1 |
| Z | Z bit | 0 or 1 |

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -101 | ERROR_101 | Check parameter 1 for correct value. |
| -102 | ERROR_102 | Check parameter 2 for correct value. |
| -103 | ERROR_103 | Check parameter 3 for correct value. |
| -104 | ERROR_104 | Check parameter 4 for correct value. |
| -105 | ERROR_105 | Check parameter 5 for correct value. |
| -106 | ERROR_106 | Check parameter 6 for correct value. |
| -107 | ERROR_107 | Check parameter 7 for correct value. |
| -108 | ERROR_108 | Check parameter 8 for correct value. |
| -109 | ERROR_109 | Check parameter 9 for correct value. |
| -121 | ERROR_121 | Emulator is not in the correct stage to process this function. |
| -122 | ERROR_122 | Front end is not configured properly. |
| -131 | ERROR_131 | Front end is not running; stop_data( ) may have been called. |

## See Also

config_extctrl( )　get_extctrl( )　get_NR( )
get_NS( )

## Example

The following example sends a FRMR frame when the emulation stage is LEVEL1.

```
if(get_stageX25( ) == LEVEL1)
   send_FRMR(0, 0, 0XFF, 0XFF);
   send_FRMR(0, 0, 0XFF, 0XFF, 1, 0, 0, 0, 1);
```

# send_I( )

|  | LEVEL1 | LINK_CONTROL | L2_FULL |  |  |
|---|---|---|---|---|---|
|  | ▨ | ▨ | ▨ |  |  |

## Format

```
#include <X25.include>

int send_I(poll, NR, NS, packet_ptr, packet_length, fcs_type)

int poll, NR, NS;
unsigned char *packet_ptr;
int packet_length, fcs_type;
```

## Description

This Level 2 library function instructs the Level 2 emulator to send an INFORMATION frame. If the Level 2 emulator is *off*, the Level 1 Transmitter can send the frame.

The other parameters are entered in one of the following ways.

- The values for `poll`, `NR`, `NS`, or `fcs_type` are specified in the `send_I( )` function.
- When the emulator is in the `L2_FULL` stage, the emulator writes the proper values. If you entered a value for any of these parameters other than `UNSPECIFIED (0xFF)`, `WARNING_161` is issued and the emulator overwrites your input.
- When the emulator is in either `LEVEL1` or `LINK_CONTROL` stage, parameters `NR`, `NS`, or `poll` containing the value `UNSPECIFIED (0xFF)` default to `0`.

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| 161 | WARNING_161 | When the Level 2 FULL emulator is ON, the values of poll, NR, and NS cannot be specified; their values are determined by the Level 2 emulator. |
| -101 | ERROR_101 | Check parameter 1 for correct value. |
| -102 | ERROR_102 | Check parameter 2 for correct value. |
| -103 | ERROR_103 | Check parameter 3 for correct value. |
| -106 | ERROR_106 | Check parameter 6 for correct value. |
| -121 | ERROR_121 | Emulator is not in correct stage to process this function. |
| -122 | ERROR_122 | Front end is not configured properly. |
| -126 | ERROR_126 | Change the send string length to be between 2 bytes and 4,106 bytes. |
| -131 | ERROR_131 | Front end is not running; stop_data( ) may have been called. |
| -141 | ERROR_141 | LAPB window is full. |
| -142 | ERROR_142 | LAPB emulator is in the wrong state. |

## See Also

config_extctrl( )    get_extctrl( )    get_NR( )
get_NS( )    format_L3_X25( )

## Example

```
char *packet_ptr, *Called_Addr, *Calling_Addr;
int packet_length;
char Facil[] = {0x42, 0x77};
Called_Addr = "5551212";
Calling_Addr = "555132";

format_L3_X25(&packet_ptr,&packet_length,
            "%PT%0A%0D%3LCI%0CC%DC%CDAs%CGAs%2Fb",
            CALL, 0, Called_Addr, Calling_Addr, Facil);

send_I(POLL_1, 1, 1, packet_ptr, packet_length, GOOD_FCS);
```

# send_INT( )

| | | | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
| | | | ▨ | ▨ | ▨ |

## Format

```
#include <X25.include>

int send_INT(control, args)

char *control;
char *args;
```

## Description

This Level 3 function sends a formatted Level 3 INTERRUPT packet.

## Character Specifications

Packet fields are defined by a control string and the associate argument list which are described in detail in chapter 8, "Frame and Packet Formatting," *HP 18321A X.25 Test Environment User's Guide*. The following table defines the conversion specification for this function.

| Specification | Use | Default Value |
|---|---|---|
| %[n]LCI | Select a logical channel (1-4095). | LCI_value |
| %[n]UDb or<br>%[n]UDs | Define the length and location of Level 3 user data. | Length = 0<br>Data = None |

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -121 | ERROR_121 | Emulator is not in correct stage to process this function. |
| -122 | ERROR_122 | Front end is not configured properly. |
| -123 | ERROR_123 | An error was found in the control string. |
| -124 | ERROR_124 | One of the control string parameters has an illegal value. |
| -125 | ERROR_125 | One of the control string parameters was entered more than once. |
| -126 | ERROR_126 | Change the send string length to be between 2 bytes and 4,106 bytes. |
| -131 | ERROR_131 | Front end is not running; stop_data( ) may have been called. |
| -141 | ERROR_141 | LAPB window is full. |
| -142 | ERROR_142 | LAPB emulator is in the wrong state. |
| -143 | ERROR_143 | LCI window full. |
| -144 | ERROR_144 | LCI in wrong state. |
| -145 | ERROR_145 | LCI is invalid. |

## See Also

send_INTC( )

## Example

```
send_INT("%2LCI%5UDs","Hello");
```

# send_INTC( )

|  |  |  |  | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|---|
|  |  |  |  | ▨ | ▨ | ▨ |

## Format

```
#include <X25.include>

int send_INTC(control, args)

char *control;
char *args;
```

## Description

This Level 3 function sends a formatted Level 3 INTERRUPT CONFIRMATION packet.

## Character Specifications

Packet fields are defined by a control string and the associate argument list which are described in detail in chapter 8, "Frame and Packet Formatting," *HP 18321A X.25 Test Environment User's Guide*. The following table defines the conversion specification for this function.

| Specification | Use | Default Value |
|---|---|---|
| `%[n]LCI` | Select a logical channel (1-4095). | `LCI_value` |

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -121 | ERROR_121 | Emulator is not in correct stage to process this function. |
| -122 | ERROR_122 | Front end is not configured properly. |
| -123 | ERROR_123 | An error was found in the control string. |
| -124 | ERROR_124 | One of the control string parameters has an illegal value. |
| -125 | ERROR_125 | One of the control string parameters was entered more than once. |
| -131 | ERROR_131 | Front end is not running; stop_data( ) may have been called. |
| -141 | ERROR_141 | LAPB window is full. |
| -142 | ERROR_142 | LAPB emulator is in the wrong state. |
| -143 | ERROR_143 | LCI window full. |
| -144 | ERROR_144 | LCI in wrong state. |
| -145 | ERROR_145 | LCI is invalid. |

## See Also

send_INT( )

## Example

```
send_INTC("%2LCI");
```

# send_REG( )

|  |  |  | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
|  |  |  | ▨ | ▨ | ▨ |

## Format

```
#include <X25.include>

int send_REG(control, args)

char *control;
char *args;
```

## Description

This Level 3 function sends a formatted Level 3 REGISTRATION packet.

## Character Specifications

Packet fields are defined by a control string and the associate argument list which are described in detail in chapter 8, "Frame and Packet Formatting," *HP 18321A X.25 Test Environment User's Guide*. The following table defines the conversion specification for this function.

| Specification | Use | Default Value |
|---|---|---|
| `%[n]A` | Implement TOA/NPI address format. | `A_Bit value` |
| `%[n]CDAb` or<br>`%[n]CDAs` | Define the length (0-15 or 0-255) and location of the called DTE address data. | Length = 0<br>Data = None |
| `%[n]CGAb` or<br>`%[n]CGAs` | Define the length (0-15 or 0-255) and location of the calling DTE address data. | Length = 0<br>Data = None |
| `%[n]Rb` | Define the length (0-255) and location of Level 3 Registration data. | Length = 0<br>Data = None |

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -121 | ERROR_121 | Emulator is not in correct stage to process this function. |
| -122 | ERROR_122 | Front end is not configured properly. |
| -123 | ERROR_123 | An error was found in the control string. |
| -124 | ERROR_124 | One of the control string parameters has an illegal value. |
| -125 | ERROR_125 | One of the control string parameters was entered more than once. |
| -131 | ERROR_131 | Front end is not running; `stop_data( )` may have been called. |
| -141 | ERROR_141 | LAPB window is full. |
| -142 | ERROR_142 | LAPB emulator is in the wrong state. |
| -143 | ERROR_143 | LCI window full. |
| -144 | ERROR_144 | LCI in wrong state. |
| -145 | ERROR_145 | LCI is invalid. |
| -161 | ERROR_161 | Extended control is `ON`. |
| -162 | ERROR_162 | Mixed use of `CGAs` and `CGAb` in the same control string. |
| -163 | ERROR_163 | Mixed use of `CDAs` and `CDAb` in the same control string. |
| -164 | ERROR_164 | String length for one or more of the following conversion specification too long: `CDAb`, `CDAs`, `CGAb`, `CGAs`, or `Fb`. |
| -165 | ERROR_165 | Either `CDAs` or `CGAs` points to a string containing a non-digit character. |

## See Also

`send_REGC( )`

## Example

```
send_REG("%0A");
```

# send_REGC( )

L2 FULL CALL CONTROL L3 ALL

## Format

```
#include <X25.include>

int send_REGC(control, args)

char *control;
char *args;
```

## Description

This Level 3 function sends a formatted Level 3 REGISTRATION CONFIRMATION packet.

## Character Specifications

Packet fields are defined by a control string and the associate argument list which are described in detail in chapter 8, "Frame and Packet Formatting," *HP 18321A X.25 Test Environment User's Guide*. The following table defines the conversion specification for this function.

| Specification | Use | Default Value |
|---|---|---|
| %[n]A | Implement TOA/NPI address format. | A_Bit value |
| %[n]CC | Cause code. | 0 |
| %[n]CDAb or<br>%[n]CDAs | Define the length (0-15 or 0-255) and location of the called DTE address data. | Length = 0<br>Data = None |
| %[n]CGAb or<br>%[n]CGAs | Define the length (0-15 or 0-255) and location of the calling DTE address data. | Length = 0<br>Data = None |
| %[n]DC | Level 3 Diagnostic Code (0-255). | 0 |

# send_REGC( )

| Specification | Use | Default Value |
|---|---|---|
| %[n]Rb | Define the length (0-255) and location of Level 3 Registration data. | Length = 0<br>Data = None |

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -121 | ERROR_121 | Emulator is not in correct stage to process this function. |
| -122 | ERROR_122 | Front end is not configured properly. |
| -123 | ERROR_123 | An error was found in the control string. |
| -124 | ERROR_124 | One of the control string parameters has an illegal value. |
| -125 | ERROR_125 | One of the control string parameters was entered more than once. |
| -131 | ERROR_131 | Front end is not running; stop_data( ) may have been called. |
| -141 | ERROR_141 | LAPB window is full. |
| -142 | ERROR_142 | LAPB emulator is in the wrong state. |
| -143 | ERROR_143 | LCI window full. |
| -144 | ERROR_144 | LCI in wrong state. |
| -145 | ERROR_145 | LCI is invalid. |
| -162 | ERROR_162 | Mixed use of CGAs and CGAb in the same control string. |
| -163 | ERROR_163 | Mixed use of CDAs and CDAb in the same control string. |
| -164 | ERROR_164 | String length for one or more of the following conversion specification too long: CDAb, CDAs, CGAb, CGAs, or Fb. |

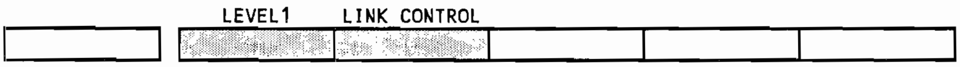
## See Also

send_REG( )

## Example

```
send_REGC("%1A");
```

# send_REJ_L2( )



## Format

```
#include <X25.include>

int send_REJ_L2(cmd, poll, NR)

int cmd, poll, NR;
```

## Description

This Level 2 library function instructs the emulator to send a Level 2 REJECT frame. There are no default values for the parameters used by this function.

| **Parameter** | **Function** | **Values** |
|---|---|---|
| cmd | COMMAND bit<br>RESPONSE bit | 256<br>257 |
| poll | The poll bit. | 0 = POLL_0<br>1 = POLL_1. |
| NR | Receive frame number<br>Normal Control Field<br>Extended Control Field | <br>0 to 7<br>0 to 127 |

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -101 | ERROR_101 | Check parameter 1 for correct value. |
| -102 | ERROR_102 | Check parameter 2 for correct value. |
| -103 | ERROR_103 | Check parameter 3 for correct value. |
| -121 | ERROR_121 | Emulator is not in correct stage to process this function. |
| -122 | ERROR_122 | Front end is not configured properly. |
| -131 | ERROR_131 | Front end is not running; stop_data( ) may have been called. |
| -141 | ERROR_141 | LAPB window is full. |
| -142 | ERROR_142 | LAPB emulator is in the wrong state. |

## See Also

config_extctrl( )    get_extctrl( )    get_NR( )

## Example

The following example sends a REJECT frame when the emulation stage is either LEVEL1 or LINK_CONTROL. It will be a RESPONSE frame with the final bit off (0) and an NR of 1.

```
if(get_stage_X25( ) == LEVEL1 || get_stage_X25( ) == LINK_CONTROL)
   send_REJ_L2(RESPONSE, FINAL_0, 1);
```

# send_REJ_L3( )

| | | | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
| | | | ▨ | ▨ | ▨ |

## Format

```
#include <X25.include>

int send_REJ_L3(PR, control, args)

int PR;
char *control;
char *args;
```

## Description

This Level 3 function sends a formatted Level 3 REJECT packet to the line.

The first parameter, `PR`, represents the Pr count and is an integer variable with the following values:

| | |
|---|---|
| 0-7 | `Mod8` sequence number range. |
| 0-127 | `Mod128` sequence number range |

See `config_SN_L3( )` for more information on sequence number configurations.

## Character Specifications

Packet fields are defined by a control string and the associate argument list which are described in detail in chapter 8, "Frame and Packet Formatting," *HP 18321A X.25 Test Environment User's Guide*. The following table defines the conversion specification for this function.

| Specification | Use | Default Value |
|---|---|---|
| %[n]LCI | Select a logical channel (1-4095). | LCI_value |
| %[n]PR | Define Level 3 receive (PR) sequence number count (0-7 or 0-127). | 0 |

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -101 | ERROR_101 | Check parameter 1 for correct value. |
| -121 | ERROR_121 | Emulator is not in correct stage to process this function. |
| -122 | ERROR_122 | Front end is not configured properly. |
| -123 | ERROR_123 | An error was found in the control string. |
| -124 | ERROR_124 | One of the control string parameters has an illegal value. |
| -125 | ERROR_125 | One of the control string parameters was entered more than once. |
| -131 | ERROR_131 | Front end is not running; stop_data( ) may have been called. |
| -141 | ERROR_141 | LAPB window is full. |
| -142 | ERROR_142 | LAPB emulator is in the wrong state. |
| -143 | ERROR_143 | LCI window full. |
| -144 | ERROR_144 | LCI in wrong state. |
| -145 | ERROR_145 | LCI is invalid. |

## See Also

send_REJ_L2( )

## Example

```
send_REJ_L3(get_PR(2), "%2LCI");
```

# send_RESET( )

| | | | L2 FULL | CALL_CONTROL | L3_ALL |
|---|---|---|---|---|---|
| | | | ▨ | ▨ | ▨ |

## Format

```
#include <X25.include>

int send_RESET(control, args)

char *control;
char *args;
```

## Description

This Level 3 function sends a formatted Level 3 RESET REQUEST or RESET INDICATION packet.

## Character Specifications

Packet fields are defined by a control string and the associate argument list which are described in detail in chapter 8, "Frame and Packet Formatting," *HP 18321A X.25 Test Environment User's Guide*. The following table defines the conversion specification for this function.

| Specification | Use | Default Value |
|---|---|---|
| %[n]CC | Cause code. | 0 |
| %[n]DC | Level 3 Diagnostic Code | None |
| %[n]LCI | Select a logical channel (1-4095). | LCI_value |

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -121 | ERROR_121 | Emulator is not in correct stage to process this function. |
| -122 | ERROR_122 | Front end is not configured properly. |
| -123 | ERROR_123 | An error was found in the control string. |
| -124 | ERROR_124 | One of the control string parameters has an illegal value. |
| -125 | ERROR_125 | One of the control string parameters was entered more than once. |
| -131 | ERROR_131 | Front end is not running; stop_data( ) may have been called. |
| -141 | ERROR_141 | LAPB window is full. |
| -142 | ERROR_142 | LAPB emulator is in the wrong state. |

## See Also

send_RESETC( )

## Example

```
send_RESET("%2LCI");
```

## send_RESETC( )

### Format

```
#include <X25.include>

int send_RESETC(control, args)

char *control;
char *args;
```

### Description

This Level 3 function sends a formatted Level 3 RESET CONFIRMATION packet.

### Character Specifications

Packet fields are defined by a control string and the associate argument list which are described in detail in chapter 8, "Frame and Packet Formatting," *HP 18321A X.25 Test Environment User's Guide*. The following table defines the conversion specification for this function.

| Specification | Use | Default Value |
|---|---|---|
| %[n]LCI | Select a logical channel (1-4095). | LCI_value |

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -121 | ERROR_121 | Emulator is not in correct stage to process this function. |
| -122 | ERROR_122 | Front end is not configured properly. |
| -123 | ERROR_123 | An error was found in the control string. |
| -124 | ERROR_124 | One of the control string parameters has an illegal value. |
| -125 | ERROR_125 | One of the control string parameters was entered more than once. |
| -131 | ERROR_131 | Front end is not running; `stop_data( )` may have been called. |
| -141 | ERROR_141 | LAPB window is full. |
| -142 | ERROR_142 | LAPB emulator is in the wrong state. |

## See Also

`send_RESET( )`

## Example

```
send_RESETC("%2LCI");
```

## send_RESTART( )

L2_FULL

### Format

```
#include <X25.include>

int send_RESTART(control, args)

char *control;
char *args;
```

### Description

This Level 3 function sends a formatted Level 3 RESTART REQUEST or RESTART INDICATION packet.

### Character Specifications

Packet fields are defined by a control string and the associate argument list which are described in detail in chapter 8, "Frame and Packet Formatting," *HP 18321A X.25 Test Environment User's Guide*. The following table defines the conversion specification for this function.

| Specification | Use | Default Value |
|---|---|---|
| %[n]CC | Cause code. | 0 |
| %[n]DC | Level 3 Diagnostic Code (0-255). | None |

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -121 | ERROR_121 | Emulator is not in correct stage to process this function. |
| -122 | ERROR_122 | Front end is not configured properly. |
| -123 | ERROR_123 | An error was found in the control string. |
| -124 | ERROR_124 | One of the control string parameters has an illegal value. |
| -125 | ERROR_125 | One of the control string parameters was entered more than once. |
| -131 | ERROR_131 | Front end is not running; stop_data( ) may have been called. |
| -141 | ERROR_141 | LAPB window is full. |
| -142 | ERROR_142 | LAPB emulator is in the wrong state. |

## See Also

send_RESTARTC( )

## Example

```
send_RESTART("");
```

## send_RESTARTC( )

### Format

```
#include <X25.include>

int send_RESTARTC( )
```

### Description

This Level 3 function sends a formatted Level 3 RESTART CONFIRMATION packet.

### Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -121 | ERROR_121 | Emulator is not in correct stage to process this function. |
| -122 | ERROR_122 | Front end is not configured properly. |
| -131 | ERROR_131 | Front end is not running; `stop_data( )` may have been called. |
| -141 | ERROR_141 | LAPB window is full. |
| -142 | ERROR_142 | LAPB emulator is in the wrong state. |

### See Also

`send_RESTART( )`

### Example

```
send_RESTARTC( );
```

# send_RNR_L2( )

| | LEVEL1 | LINK CONTROL | | | |
|---|---|---|---|---|---|
| | ▨ | ▨ | | | |

## Format

```
#include <X25.include>

int send_RNR_L2(cmd, poll, NR)

int cmd, poll, NR;
```

## Description

This Level 2 library function instructs the emulator to send a Level 2 RNR frame. There are no default values for the parameters used by this function. The expected frame sequence number used is passed in by parameter NR.

| Parameter | Function | Values |
|---|---|---|
| cmd | COMMAND bit<br>RESPONSE bit | 256<br>257 |
| poll | The poll bit. | 0 = POLL_0<br>1 = POLL_1 |
| NR | Receive frame number<br>Normal Control Field<br>Extended Control Field | <br>0 to 7<br>0 to 127 |

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -101 | ERROR_101 | Check parameter 1 for correct value. |
| -102 | ERROR_102 | Check parameter 2 for correct value. |
| -103 | ERROR_103 | Check parameter 3 for correct value. |
| -121 | ERROR_121 | Emulator is not in correct stage to process this function. |
| -122 | ERROR_122 | Front end is not configured properly. |
| -131 | ERROR_131 | Front end is not running; stop_data( ) may have been called. |
| -141 | ERROR_141 | LAPB window is full. |
| -142 | ERROR_142 | LAPB emulator is in the wrong state. |

## See Also

config_extctrl( )    get_extctrl( )    get_NR( )

## Example

The following example sends a RNR frame when the emulation stage is either LEVEL1 or LINK_CONTROL. It is a RESPONSE frame with the poll bit OFF (0) and an NR of 1.

```
if(get_stageX25( ) == LEVEL1 || get_stage_X25( ) == LINK_CONTROL)
    send_RNR_L2(RESPONSE, POLL_0, 1);
```

# send_RNR_L3( )

| | | | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
| | | | ▨ | ▨ | ▨ |

## Format

```
#include <X25.include>

int send_RNR_L3(PR, control, args)

int PR;
char *control;
char *args;
```

## Description

This Level 3 function sends a formatted Level 3 RNR packet.

The first parameter, `PR`, represents the Pr count and is an integer variable with the following values:

| | |
|---|---|
| 0-7 | `Mod8` sequence number range |
| 0-127 | `Mod128` sequence number range |

See `config_SN_L3( )` for more information on sequence number configurations.

## Character Specifications

Packet fields are defined by a control string and the associate argument list which are described in detail in chapter 8, "Frame and Packet Formatting," *HP 18321A X.25 Test Environment User's Guide*. The following table defines the conversion specification for this function.

## send_RNR_L3( )

| Specification | Use | Default Value |
|---|---|---|
| %[n]LCI | Select a logical channel (1-4095). | LCI_value |
| %[n]PR | Define Level 3 receive (PR) sequence number count (0-7 or 0-127). | 0 |

### Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -101 | ERROR_101 | Check parameter 1 for correct value. |
| -121 | ERROR_121 | Emulator is not in correct stage to process this function. |
| -122 | ERROR_122 | Front end is not configured properly. |
| -123 | ERROR_123 | An error was found in the control string. |
| -124 | ERROR_124 | One of the control string parameters has an illegal value. |
| -125 | ERROR_125 | One of the control string parameters was entered more than once. |
| -131 | ERROR_131 | Front end is not running; stop_data( ) may have been called. |
| -141 | ERROR_141 | LAPB window is full. |
| -142 | ERROR_142 | LAPB emulator is in the wrong state. |
| -143 | ERROR_143 | LCI window full. |
| -144 | ERROR_144 | LCI in wrong state. |
| -145 | ERROR_145 | LCI is invalid. |

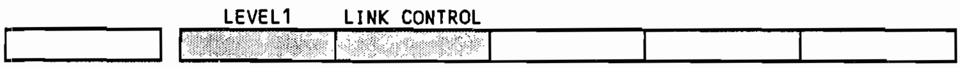
### See Also

send_RNR_L2( )

### Example

```
send_RNR_L3(get_PR(7),"%2LCI");
```

# send_RR_L2( )



## Format

```
#include <X25.include>

int send_RR_L2(cmd, poll, NR)

int cmd, poll, NR;
```

## Description

This Level 2 library function instructs the emulator to send a Level 2 RR frame. There are no default values for the parameters used by this function.

| Parameter | Function | Values |
|---|---|---|
| cmd | COMMAND bit<br>RESPONSE bit | 256<br>257 |
| poll | The poll bit. | 0 = POLL_0<br>1 = POLL_1 |
| NR | Receive frame number<br>Normal Control Field<br>Extended Control Field | <br>0 to 7<br>0 to 127 |

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -101 | ERROR_101 | Check parameter 1 for correct value. |
| -102 | ERROR_102 | Check parameter 2 for correct value. |
| -103 | ERROR_103 | Check parameter 3 for correct value. |
| -121 | ERROR_121 | Emulator is not in correct stage to process this function. |
| -122 | ERROR_122 | Front end is not configured properly. |
| -131 | ERROR_131 | Front end is not running; stop_data( ) may have been called. |
| -141 | ERROR_141 | LAPB window is full. |
| -142 | ERROR_142 | LAPB emulator is in the wrong state. |

## See Also

config_extctrl( )   get_extctrl( )   get_NR( )

## Example

The following example sends a RR frame when the emulation stage is either LEVEL1 or LINK_CONTROL. It is a RESPONSE frame with the poll bit OFF (0) and an NR of 1.

```
if(get_stageX25( ) == LEVEL1 || get_stage_X25( ) == LINK_CONTROL)
    send_RR_L2(RESPONSE, POLL_0, 1);
```

# send_RR_L3( )

|  |  |  | L2 FULL | CALL CONTROL |  |
|---|---|---|---|---|---|
|  |  |  | ▨ | ▨ |  |

## Format

```
#include <X25.include>

int send_RR_L3(PR, control, args)

int PR;
char *control;
char *args;
```

## Description

This Level 3 function sends a formatted Level 3 RR packet.

The first parameter, PR, represents the Pr count and is an integer variable with the following values:

|  |  |
|---|---|
| 0-7 | Mod8 sequence number range. |
| 0-127 | Mod128 sequence number range |

See config_SN_L3( ) for more information on sequence number configurations.

## Character Specifications

The following table defines the conversion specification for this function. For a detailed description of conversion specification see chapter 8, "Frame and Packet Formatting," *HP 18321A X.25 Test Environment User's Guide.*

# send_RR_L3( )

| Specification | Use | Default Value |
|---|---|---|
| %[n]LCI | Select a logical channel (1-4095). | LCI_value |
| %[n]PR | Define Level 3 receive (PR) sequence number count (0-7 or 0-127). | 0 |

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -101 | ERROR_101 | Check parameter 1 for correct value. |
| -121 | ERROR_121 | Emulator is not in correct stage to process this function. |
| -122 | ERROR_122 | Front end is not configured properly. |
| -123 | ERROR_123 | An error was found in the control string. |
| -124 | ERROR_124 | One of the control string parameters has an illegal value. |
| -125 | ERROR_125 | One of the control string parameters was entered more than once. |
| -131 | ERROR_131 | Front end is not running; stop_data( ) may have been called. |
| -141 | ERROR_141 | LAPB window is full. |
| -142 | ERROR_142 | LAPB emulator is in the wrong state. |

## See Also

send_RR_L2( )

## Example

```
send_RR_L3(get_PR(2), "%2LCI");
```

# send_SABM( )

LEVEL1

| | LEVEL1 | | | | |
|---|---|---|---|---|---|
| | ▨ | | | | |

## Format

```
#include <X25.include>

int send_SABM(poll)

int poll;
```

## Description

This Level 2 library function instructs the emulator to send a Level 2 SABM frame. The poll bit is set depending on the value of the poll parameter. Values for poll are as follows.

```
0 = POLL_0                    1 = POLL_1
```

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -101 | ERROR_101 | Check parameter 1 for correct value. |
| -121 | ERROR_121 | Emulator is not in correct stage to process this function. |
| -122 | ERROR_122 | Front end is not configured properly. |
| -131 | ERROR_131 | Front end is not running; stop_data( ) may have been called. |
| -161 | ERROR_161 | Extended control is ON. |

## See Also

config_extctrl( )     get_extctrl( )

## Example

The following example sends a SABM frame with the poll bit set (1) when extended control is OFF.

```
if(config_extctrl(OFF) == SUCCESSFUL)
    send_SABM(POLL_1);
```

# send_SABME( )

|  | LEVEL1 |  |  |  |  |
|---|---|---|---|---|---|
|  | ▨ |  |  |  |  |

## Format

```
#include <X25.include>

int send_SABME(poll)

int poll;
```

## Description

This Level 2 library function instructs the emulator to send a Level 2 SABME frame. The poll bit is set depending on the value of the `poll` parameter. Values for `poll` are as follows.

```
0 = POLL_0                    1 = POLL_1
```

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -101 | ERROR_101 | Check parameter 1 for correct value. |
| -121 | ERROR_121 | Emulator is not in correct stage to process this function. |
| -122 | ERROR_122 | Front end is not configured properly. |
| -131 | ERROR_131 | Front end is not running; `stop_data( )` may have been called. |

## See Also

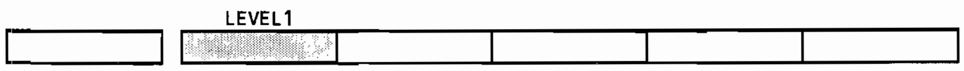
`config_extctrl( )`        `get_extctrl( )`

## Example

The following example sends a SABME frame with the poll bit set `(1)` when extended control is `ON`.

```
if(config_extctrl(ON) == SUCCESSFUL)
    send_SABME(POLL_1);
```

# send_UA( )



## Format

```
#include <X25.include>

int send_UA(final)

int final;
```

## Description

This Level 2 library function instructs the emulator to send a Level 2 UA frame. The `final` bit is set depending on the value of the `final` parameter. Values for `final` are as follows.

```
0 = FINAL_0                  1 = FINAL_1
```

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -101 | ERROR_101 | Check parameter 1 for correct value. |
| -121 | ERROR_121 | Emulator is not in correct stage to process this function. |
| -122 | ERROR_122 | Front end is not configured properly. |
| -131 | ERROR_131 | Front end is not running; `stop_data( )` may have been called. |

## See Also

`config_extctrl( )`     `get_extctrl( )`

## Example

The following example sends a UA frame with the final bit set `(1)` when the emulation stage is `LEVEL1`.

```
if(get_stage_X25( ) == LEVEL1)
    send_UA(FINAL_1);
```

## sendf_X25( )

|  | LEVEL1 | LINK CONTROL | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
|  | ▨ | ▨ | ▨ | ▨ | ▨ |

### Format

```
#include <X25.include>
#include <dlib.include>

int sendf_X25(control, args)

char *control;
char *args;
```

### Description:

This function sends a LAPB-Level 2/X.25-Level 3 formatted string to the Level 1 emulator, bypassing other upper level emulators.

`sendf_X25( )` performs formatting functions similar to those of the C programming language `printf( )` and `scanf( )` functions and the DataCommC `sendf( )` function (see *HP 18320A DataCommC Library Reference Manual*). The `sendf_X25( )` control string and argument list determine frame and packet formatting (see chapter 8, "Frame and Packet Formatting," *HP 18321A X.25 Test Environment User's Guide*).

The following are the primary uses for `sendf_X25( )`.

- Specifying the Level 2 field values such as LAPB Address field and frame type.
- Specifying standard CCITT frame types.
- Building non-standard frames using character strings.

- Specifying the Level 3 fields values such as LCI and packet type.
- Specifying standard CCITT packet types.
- Building non-standard packets using character strings.

The control string contains conversion specifications. These specifications are introduced by the character `%` and have a dimension field `[n]`, a conversion character, and possibly a pointer. As a minimum a conversion specification contains only the character `%` and the conversion character. The argument list parameters `(args)` are separated by commas.

# sendf_X25( )

In the following example, the sendf_X25( ) control string and the argument list are used to define the conversion specifications. Note that when a conversion specification is omitted it is assigned a default value.

| control string | argument list |
|---|---|
| `sendf_X25("%Add%2NR%*NS%PF%Q%D%UDs",` | `1, 3, ON, OFF, OFF, "Hello world");` |

The following tables shows the contents of the resulting frame and packet.

| | Field | Value |
|---|---|---|
| %Add | Address | 1 |
| %2NR | Receive Sequence Number Count | 2 |
| %*NS | Send Sequence Number Count | 3 |
| %PF | Poll Final | 1 |
| %Q | Q Bit | 0 |
| %D | D Bit | 0 |
| %UDs | User Data | "Hello world" |

## Conversion Specifications

sendf_X25( ) is the only function that can use all the conversion specifications which are listed in the following table. For additional information about these characters see chapter 8, "Frame and Packet Formatting," *HP 18321A X.25 Test Environment User's Guide*.

The majority of sendf_X25( ) conversion specifications define standard Level 2 and Level 3 fields. For Level 2 they define fields such as frame type (FT) and frame specific fields (for example, NS and NR). For Level 3 they define packet specific fields (for example, the Cause Code field CC).

Some conversion specifications specify strings inserted either after a Level 2 or Level 3 field or that overwrite, that is, replace a field. These are the conversion specifications FDb, FDs, PDb, PDs, UDb, and UDs. Other conversion specifications, such as CDAb, use strings for standard Level 3 fields, in this case the Called Address fields.

| Specification | Use | Default Value |
|---|---|---|
| %[n]A | Implement TOA/NPI address format. | A_Bit value |
| %[n]Add | Define Level 2 address field. | Determined by logical_dev value based on frame type (COMMAND or RESPONSE). |
| %[n]CC | Reason for clearing the call. | 0 |
| %[n]CDAb or<br>%[n]CDAs | Define the length and location of the called address data. | Length = 0<br>Data = None |
| %[n]CGAb or<br>%[n]CGAs | Define the length and location of the calling address data. | Length = 0<br>Data = None |
| %[n]D | Specify Level 3 delivery confirmation requirement. | D_bit |
| %[n]DC | Level 3 Diagnostic Code | 0 |
| %[n]DEb | Define the length and location of the Level 3 Diagnostic Explanation data. | None |
| %[n]EC | Define Level 2 frame numbering scheme as either MOD_8 or MOD_128. | ext_ctrl value |
| %[n]f | Specify Level 2 FCS value. | 0 (GOOD_FCS) |

| Specification | Use | Default Value |
|---|---|---|
| %[n]Fb | Define the length and location of the Level 3 Facilities data. | **Information Frame**<br>Packet type dependent.<br>**Supervisory Frame**<br>Length = 0<br>Data = None<br>**Unnumbered Frame**<br>Length = 0<br>Data = None |
| %[n]FDb, or<br>%[n]FDs | Define the length and location of the Level 2 and Level 3 frame data. | None when PDb, PDs, UDb, or UDs used. Otherwise, frame type dependent.<br>**Information Frame**<br>Packet type dependent.<br>**Supervisory Frame**<br>Length = 0<br>Data = None<br>**Unnumbered Frame**<br>Length = 0<br>Data = None |
| %[n]FTb | Define a standard Level 2 Frame Type. | When no FTI - I frame. |
| %[n]FTI | Either define a *non-standard* Level 2 Frame Type. or change frame format. | As defined by FT or FTI |
| %[n]LCI | Select a logical channel. | LCI_value |
| %[n]M | Signify requirement for another DATA packet. | 0 (OFF) |
| %[n]NR or<br>%[n]NS | Define Level 2 receive (NR) or send (NS) sequence number count. | 0 |

| Specification | Use | Default Value |
|---|---|---|
| %[n]PDb or<br>%[n]PDs | Define length and packet data | None when FDb, FDs, UDb, or UDs used. Otherwise, frame type dependent.<br>**Information Frame**<br>Packet type dependent.<br>**Supervisory Frame**<br>Length = 0<br>Data = None<br>**Unnumbered Frame**<br>Length = 0<br>Data = None |
| %[n]PF | Specify Level 2 COMMAND frame Poll bit or RESPONSE frame Final bit. | 0 (OFF) |
| %[n]PR or<br>%[n]PS | Define Level 3 receive (PR) or send (PS) sequence number count. | 0 |
| %[n]PT | Define a standard Level 3 Packet type. | DATA |
| %[n]Q | Specify Level 3 qualification requirement. | Q_bit value |
| %[n]Rb | Define the length and location of Level 3 Registration data. | Length = 0<br>Data = None |
| %[n]SN | Define Level 3 packet sequence numbering scheme as either MOD_8 or MOD_128. | SN_value |

| Specification | Use | Default Value |
|---|---|---|
| %[n]UDb or<br>%[n]UDs | Define the length and location of Level 3 user data. | None when FDb, FDs, PDb, or PDs used. Otherwise, frame type dependent.<br>**Information Frame**<br>Packet type dependent<br>**Supervisory Frame**<br>Length = 0<br>Data = None<br>**Unnumbered Frame**<br>Length = 0<br>Data = None |

## Return Values:

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| 161 | WARNING_161 | FTI and FT do not agree. |
| -121 | ERROR_121 | Emulator is not in correct stage to process this function. |
| -122 | ERROR_122 | Front end is not configured properly. |
| -123 | ERROR_123 | An error was found in the control string. |
| -124 | ERROR_124 | One of the control string parameters has an illegal value. |
| -125 | ERROR_125 | One of the control string parameters was entered more than once. |
| -126 | ERROR_126 | Change the send string length to be between 2 bytes and 4,106 bytes. |
| -131 | ERROR_131 | Front end is not running; stop_data( ) may have been called. |
| -161 | ERROR_161 | Mixed use of FD, PD, and UD. |
| -162 | ERROR_162 | Mixed use of CGAs and CGAb in the same control string. |
| -163 | ERROR_163 | Mixed use of CDAs and CDAb in the same control string. |
| -164 | ERROR_164 | String length for one or more of the following control string entries too long: CDAb, CDAs, CGAb, CGAs, or Fb. |
| -165 | ERROR_165 | Either CDAs or CGAs points to a string containing a non-digit ASCII character. |
| -166 | ERROR_166 | Non-standard value entered for %FT without specifying %FTI. |
| -167 | ERROR_167 | Use %A for this type of packet. |
| -168 | ERROR_168 | Use %Q for this type of packet. |

## See Also:

DataCommC
printf( )
scanf( )
sendf( )

## Examples

```
char   *User_Data, *Called_Addr, *Calling_Addr;
char  Facil[] = {0x42, 0x77};
Called_Addr = "5551212";
Calling_Addr = "555132";
User_Data = "This is UserData";
sendf_X25("%Add%NS%NR%PT%A%D%LCI%C%DC%CDAs%CGAs%*Fb%UDs",
          COMMAND, get_NS( ), get_NR( ), CLEAR, 0, 1, 3, 0, 0,
          Called_Addr,  Calling_Addr, 2, Facil,  User_Data);
```

The following example sends an I-frame without packet information.

```
sendf_X25("%5NR%7NS%FT%PDs", I," ");
```

# set_Abit( )

| OFF | LEVEL1 | LINK CONTROL | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
| ▨ | ▨ | ▨ | ▨ | ▨ | ▨ |

## Format

```
#include <X25.include>

int set_Abit(Abit)

unsigned char Abit;
```

## Description

This Level 3 function sets the default value that is placed in the TOA/NPI address format subfield of a transmitted packet. The default value is used in functions where the TOA/NPI value has not been passed as a parameter. The initial value of the default Abit is `0`.

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -101 | ERROR_101 | Check parameter 1 for the correct value. |

## See Also

`get_Abit( )`

## Example

This example sets the default Abit value to `0`.

```
set_Abit(0);
```

# set_busy_L2()

|  |  |  | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
|  |  |  | ▨ | ▨ | ▨ |

## Format

```
#include <X25.include>

int set_busy_L2( )
```

## Description

This Level 2 function forces the Level 2 emulator into the busy state and sets the LAPB busy flag to `1`. The busy flag has an initial value of `0`. Once it has been set using `set_busy_L2( )`, it can be cleared again using `clear_busy_L2( )`.

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -121 | ERROR_121 | Emulator not in correct stage to process this function. |

## See Also

`get_busy_L2( )` `clear_busy_L2( )`

## Example

This example sets the Level 2 busy flag to `1`.

```
set_busy_L2( );
```

## set_DBit( )

| OFF | LEVEL1 | LINK CONTROL | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
| ▨ | ▨ | ▨ | ▨ | ▨ | ▨ |

### Format

```
#include <X25.include>

int set_Dbit(Dbit)

int Dbit;
```

### Description

This Level 3 function sets the Delivery bit default value to either 1 or 0. The default value is placed into the Delivery bit field of a transmitted packet when the Delivery bit parameter has not been passed to a function. The default Dbit value is initially 0.

### Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -101 | ERROR_101 | Check parameter 1 for the correct value. |

### See Also

get_Dbit( )

### Example

This example sets the default Delivery bit value to 1.

```
set_Dbit(1);
```

## set_DBit( )

## set_facil_CALL( )

| OFF | LEVEL1 | LINK CONTROL | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
| ▨ | ▨ | ▨ | ▨ | ▨ | ▨ |

### Format

```
#include <X25.include>

int set_facil_CALL(length,string_ptr)

unsigned int length;
char *string_ptr;
```

### Description

This Level 3 function initializes the default string that is placed in the Facility field of CALL REQUEST or INCOMING CALL packets transmitted by the emulator when `call_establish( )` or `send_CALL( )` are used.

The value passed in for the `string_ptr` parameter is a pointer to the desired Facility field string, and the `length` parameter value is the length of the string which can range between 0 and 255.

The default Facility field string for CALL REQUEST and INCOMING CALL packets is initially a null string.

### Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -101 | ERROR_101 | Check parameter 1 for the correct value. |

### See Also

`get_facil_CALL( )`

## Example

This example sets the Facility field string equal to the three characters contained in `facil`.

```
char facil[] = {0x02, 0x42, 0x77};

set_facil_CALL(3,facil);
```

# set_facil_CALLC( )

| OFF | LEVEL1 | LINK CONTROL | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
| ▨ | ▨ | ▨ | ▨ | ▨ | ▨ |

## Format

```
#include <X25.include>

int set_facil_CALLC(length,string_ptr)

unsigned int length;
char *string_ptr;
```

## Description

This Level 3 function initializes the default string that is placed in the Facility field of CALL ACCEPTED or CALL CONNECTED packets which are transmitted automatically by the emulator or when `send_CALLC( )` is used.

The value passed in for the `string_ptr` parameter is a pointer to the desired Facility field string, and the `length` parameter value is the length of the string which can range between 0 and 255.

The default Facility string for CALL ACCEPTED or CALL CONNECTED packets is initially a null string.

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -101 | ERROR_101 | Check parameter 1 for the correct value. |

## See Also

`get_facil_CALLC( )`

## Example

This example sets the Facility field string equal to the three characters contained in `facil`.

```
char facil[] = {0x02, 0x42, 0x77};

set_facil_CALLC(3,facil);
```

# set_facil_CLEAR( )

| OFF | LEVEL1 | LINK CONTROL | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
| ▨ | ▨ |  |  | ▨ | ▨ |

## Format

```
#include <X25.include>

int set_facil_CLEAR(length,string_ptr)

unsigned int length;
char *string_ptr;
```

## Description

This Level 3 function initializes the default string that is placed in the Facility field of CLEAR REQUEST or CLEAR INDICATION packets which are transmitted automatically by the emulator or when `send_CLEAR( )` is used.

The value passed in for the `string_ptr` parameter is a pointer to the desired Facility field string, and the `length` parameter value is the length of the string, which can range between 0 and 255.

The default Facility field string for CLEAR REQUEST and CLEAR INDICATION packets is initially a null string.

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -101 | ERROR_101 | Check parameter 1 for the correct value. |

## See Also

`get_facil_CLEAR( )`

## Example

This example sets the Facility field string equal to the three characters contained in `facil`.

```
char facil[] = {0x02, 0x42, 0x77};

set_facil_CLEAR(3,facil);
```

## set_facil_CLEARC( )

| OFF | LEVEL1 | LINK_CONTROL | L2_FULL | CALL_CONTROL | L3_ALL |
|---|---|---|---|---|---|
| ▨ | ▨ | ▨ | ▨ | ▨ | ▨ |

### Format

```
#include <X25.include>

int set_facil_CLEARC(length,string_ptr)

unsigned int length;
char *string_ptr;
```

### Description

This Level 3 function initializes the default string that is placed in the Facility field of CLEAR CONFIRMATION packets which are transmitted automatically by the emulator or when `send_CLEARC( )` is used.

The value passed in for the `string_ptr` parameter is a pointer to the desired Facility field string, and the `length` parameter value is the length of the string which can range between 0 and 255.

The default Facility field string for CLEAR CONFIRMATION packets is a null string.

### Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -101 | ERROR_101 | Check parameter 1 for the correct value. |

### See Also

`get_facil_CLEARC( )`

### Example

This example sets the Facility field string equal to the three characters contained in `facil`.

```
char facil[] = {0x02, 0x42, 0x77};

set_facil_CLEARC(3,facil);
```

## set_LCI( )

| OFF | LEVEL1 | LINK CONTROL | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
| ▨ | ▨ | ▨ | ▨ | ▨ | ▨ |

### Format

```
#include <X25.include>

int set_LCI(lci)

unsigned int lci;
```

### Description

This Level 3 function sets the value of the default LCI field that is placed in the transmitted packets if an LCI value has not been passed to the send function. The default value is not used for RESTART REQUEST, RESTART CONFIRMATION, DIAGNOSTIC, REGISTRATION REQUEST or REGISTRATION CONFIRMATION packets since those packets are defined to have LCI values of 0.

The LCI default value may be set to any value in the range 1 to 4095. The initial default value is 1.

### Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -101 | ERROR_101 | Check parameter 1 for the correct value. |

### See Also

get_LCI( )

### Example

This example sets the value of the default LCI variable to 7.

```
set_LCI(7);
```

# set_Qbit( )

| OFF | LEVEL1 | LINK CONTROL | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
| ▨ | ▨ | ▨ | ▨ | ▨ | ▨ |

## Format

```
#include <X25.include>

int set_Qbit(Qbit)

unsigned char Qbit;
```

## Description

This Level 3 function sets the Qualifier bit default value to either `1` or `0`. The default value is placed in the Qualifier bit field of a transmitted packet when the Qbit parameter has not been passed to the send function. The initial default value is `0`.

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -101 | ERROR_101 | Check parameter 1 for the correct value. |

## See Also

`get_Qbit( )`

## Example

This example sets the value of the default Qbit variable to `1`.

```
set_Qbit(1);
```

## set_Qbit( )

# set_userdata_CALL( )

| OFF | LEVEL1 | LINK CONTROL | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
| ▨ | ▨ | ▨ | ▨ | ▨ | ▨ |

## Format

```
#include <X25.include>

int set_userdata_CALL(length,string_ptr)

unsigned int length;
char *string_ptr;
```

## Description

This Level 3 function sets the default string to be used for the Call User Data field of CALL REQUEST or INCOMING CALL packets transmitted by the emulator when `call_establish( )` or `send_CALL( )` is used.

The value passed in for the `string_ptr` parameter is a pointer to the desired Call User Data string, and the `length` parameter value is the length of the string which can range between `0` and `255`.

The default Call User Data field string for CALL REQUEST and INCOMING CALL packets is initially a null string.

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -101 | ERROR_101 | Check parameter 1 for the correct value. |

## See Also

`get_userdata_CALL( )`

## Example

These examples show how to set the Call User Data string equal to a Hex string and an ASCII string.

```
char User_Data[] = {0x00, 0x00, 0x00, 0x01};
set_userdata_CALL(4, User_Data);


char *User_Data;
User_Data = "This is User Data";
set_userdata_CALL(strlen(User_Data), User_Data);
```

# set_userdata_CALLC( )

| OFF | LEVEL1 | LINK CONTROL | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
| ▨ | ▨ | ▨ | ▨ | ▨ | ▨ |

## Format

```
#include <X25.include>

int set_userdata_CALLC(length,string_ptr)

unsigned int length;
char *string_ptr;
```

## Description

This Level 3 function sets the default string to be used for the Call User Data field of CALL ACCEPTED or CALL CONNECTED packets which are transmitted automatically by the emulator or when `send_CALLC( )` is used.

The value passed in for the `string_ptr` parameter is a pointer to the desired Call User Data string, and the `length` parameter value is the length of the string which can range between `0` and 255.

The default Called User Data field string for CALL ACCEPTED and CALL CONNECTED packets is initially a null string.

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -101 | ERROR_101 | Check parameter 1 for the correct value. |

## See Also

`get_userdata_CALLC( )`

## Example

These examples show how to set the Call User Data string equal to a Hex string and an ASCII string.

```
char User_Data[] = {0x00, 0x00, 0x00, 0x01};
set_userdata_CALLC(4, User_Data);


char *User_Data;
User_Data = "This is User Data";
set_userdata_CALLC(strlen(User_Data), User_Data);
```

# set_userdata_CLEAR( )

| OFF | LEVEL1 | LINK CONTROL | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
| ▨ | ▨ | ░ | ░ | ▨ | ▨ |

## Format

```
#include <X25.include>

int set_userdata_CLEAR(length,string_ptr)

unsigned int length;
char *string_ptr;
```

## Description

This Level 3 function sets the default string to be used for the Clear User Data field of CLEAR REQUEST or CLEAR INDICATION packets which are transmitted automatically by the emulator when `send_CLEAR( )` is used.

The value passed in for the `string_ptr` parameter is a pointer to the desired Clear User Data string, and the `length` parameter value is the length of the string which can range between `0` and `255`.

The default Clear User Data field string for CLEAR REQUEST and CLEAR INDICATION packets is initially a null string.

## Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -101 | ERROR_101 | Check parameter 1 for the correct value. |

## See Also

`get_userdata_CLEAR( )`

## Example

These examples set the Call User Data string equal to two different character strings.

```
char User_Data[] = {0x00, 0x00, 0x00, 0x01};
set_userdata_CLEAR(4, User_Data);


char *User_Data;
User_Data = "This is User Data";
set_userdata_CLEAR(strlen(User_Data), User_Data);
```

## set_windowsize_L3()

| OFF | LEVEL1 | LINK CONTROL | L2 FULL | CALL CONTROL | L3 ALL |
|---|---|---|---|---|---|
| ▨ | ▨ | ▨ | ▨ | ▨ | ▨ |

### Format

```
#include <X25.include>

int set_windowsize_L3(trmt_window)

unsigned char trmt_window;
```

### Description

This Level 3 function sets the transmit window size for all LCIs. The initial window size is 2.

### Return Values

| Value | Constant | Definition |
|---|---|---|
| 0 | SUCCESSFUL | Specified task is completed. |
| -101 | ERROR_101 | Check parameter 1 for the correct value. |

### See Also

`get_windowsize_L3( )`

### Example

This example sets the transmit window to 10.

```
unsigned char transmit_window = 10;
set_SN_L3(MOD_128);
set_windowsize_L3(transmit_window);
```

4

# Include Files

The X.25 Test Environment uses the DataCommC include files (see chapter 4, "Include Files," *HP 18320A DataCommC Library Reference*) and the `X25.Include` file. The `X25.Include` file is stored in the `c:/C/Include` directory. It can be edited or printed using the DataCommC Development Environment.

The following rules apply to the `X25.Include` file.

- Enter the include file name as `X25.Include`.
- In source code the entry `#include X25.h` is the equivalent of `#include X25.Include`.
- In program files the entry is `#include <X25.Include>`.

The following is a listing of the `X25.Include` file that has been reformatted for easier reading and the comments have been expanded to better describe functions and operations. *The functions and commands in this listing and the functions and commands in the* `X25.Include` *file are identical at the time of publication.* As with all software products, enhancements and revisions may be incorporated prior to shipment. Therefore, it is possible that there may be *minor* differences. The comments and organization are not identical.

```
/*****************************************************************************
*                                                                            *
*  MODULE NAME:  User Program Include File                                   *
*  Created:      5/10/89                                                     *
*  Description:                                                              *
*    This is the include to be used by the user program.  It only includes   *
*    those declarations that the user needs to access.                       *
*                                                                            *
*  To access this file, enter #include <X25_user.include>                    *
*                                                                            *
*  This file is automatically included in any library or emulator having    *
*  #include <X25.include>.                                                   *
*                                                                            *
*****************************************************************************/
```

```
/*  Routines that Return Other than Integer values.  */

extern long get_T1();
extern long get_N1();
```

```
/*  Routines that Return Integer Values.  */

/*****************************************************************************
*                                                                            *
*  The possible return values are divided into three categories:            *
*                                                                            *
*    SUCCESSFUL --- The function has performed its specified task            *
*                     successfully.                                          *
*                                                                            *
*    ERRORS     --- The function is halted because an unexpected situation   *
*                     (error) is encountered.                                *
*                                                                            *
*    WARNINGS   --- A situation which does not affect the normal operation   *
*                     of the function but is worth mentioning to the user    *
*                     as a warning.                                          *
*                                                                            *
*  Listed below are the names and the numeric values of the possible return *
*  values:                                                                   *
*****************************************************************************/
```

```
#ifndef DTE_FRAME
#include <message.include>
#endif

/* #define SUCCESSFUL   0    /* Operation successful (defined in retval.inc)  */
```

```
/*************************************/
/*  X25 ERROR Return Value Defines:  */
/*************************************/

/*  ERROR_101 to ERROR_120 indicate errors in fixed parameters.  */

#define ERROR_101    -101  /* Check parameter 1 for correct value.        */
#define ERROR_102    -102  /* Check parameter 2 for correct value.        */
#define ERROR_103    -103  /* Check parameter 3 for correct value.        */
#define ERROR_104    -104  /* Check parameter 4 for correct value.        */
#define ERROR_105    -105  /* Check parameter 5 for correct value.        */
#define ERROR_106    -106  /* Check parameter 6 for correct value.        */
#define ERROR_107    -107  /* Check parameter 7 for correct value.        */
#define ERROR_108    -108  /* Check parameter 8 for correct value.        */
#define ERROR_109    -109  /* Check parameter 9 for correct value.        */
#define ERROR_110    -110  /* Check parameter 10 for correct value.       */
```

```
/*  ERROR_121 to ERROR_140 are general errors.  */

#define ERROR_121    -121  /*  Emulator is not in the correct stage to    */
                           /*    process this command.                    */
#define ERROR_122    -122  /*  Front end is not configured properly.      */
#define ERROR_123    -123  /*  An error was found in the control string.  */
#define ERROR_124    -124  /*  One of the control string parameters has   */
                           /*    an illegal value.                        */
#define ERROR_125    -125  /*  One of the control string parameters was   */
                           /*    entered more than once.                  */
#define ERROR_126    -126  /*  Change the send string length to be between */
                           /*    2 bytes and 4,106 bytes.                 */
#define ERROR_127    -127  /*  LCI is not configured to be a PVC or the   */
                           /*    LCI is configured to be an SVC and there */
                           /*    is not an active call on that channel.   */
#define ERROR_128    -128  /*  The emulator can only be staged down; use  */
                           /*    emulate_x25(OFF) and emulate_x25(stage)  */
                           /*    to stage up.                             */
#define ERROR_129    -129  /*  Incorrect pod attached.                    */
#define ERROR_130    -130
#define ERROR_131    -131  /*  Front end is not running; stop_data() may  */
                           /*    have been called.                        */
#define ERROR_132    -132  /*  Inactive LCI; LCI is a PVC with no activity */
                           /*    or there is not an active call on the LCI.*/
#define ERROR_133    -133  /*  Invalid LCI; LCI is not in the PVC or SVC  */
                           /*    range specified by config_LCI_ranges()   */
/*  ERROR_141 to ERROR_160 are used to indicate negative acknowledgement    */
/*  to a send or emulator command message.                                  */

#define ERROR_141     -141  /*  LAPB window is full.                        */
#define ERROR_142     -142  /*  LAPB is in the wrong state.                 */
#define ERROR_143     -143  /*  LCI window is full (send_DATA() only).      */
#define ERROR_144     -144  /*  LCI is in the wrong state.                  */
#define ERROR_145     -145  /*  LCI is invalid.                             */
#define ERROR_146     -146  /*  All LCIs have been used.                    */
#define ERROR_147     -147  /*  Invalid value for stage in CHG_STAGE        */
                            /*    command.                                  */
#define ERROR_148     -148
#define ERROR_149     -149
#define ERROR_150     -150
```

```
/*  ERROR_161 to ERROR_170 are routine dependent errors.  */

#define ERROR_161     -161
#define ERROR_162     -162
#define ERROR_163     -163
#define ERROR_164     -164
#define ERROR_165     -165
#define ERROR_166     -166
#define ERROR_167     -167
#define ERROR_168     -168
#define ERROR_169     -169
#define ERROR_170     -170
```

```
/****************************************/
/*  X25 WARNING Return Value Defines:  */
/****************************************/

/*  WARNING_101 to WARNING_120 are warnings concerning fixed parameters.  */

#define WARNING_101   101    /*  Cannot process parameter 1; it is assigned   */
                             /*  the default value.                           */
#define WARNING_102   102    /*  Cannot process parameter 2; it is assigned   */
                             /*  the default value.                           */
#define WARNING_103   103    /*  Cannot process parameter 3; it is assigned   */
                             /*  the default value.                           */
#define WARNING_104   104    /*  Cannot process parameter 4; it is assigned   */
                             /*  the default value.                           */
#define WARNING_105   105    /*  Cannot process parameter 5; it is assigned   */
                             /*  the default value.                           */
#define WARNING_106   106    /*  Cannot process parameter 6; it is assigned   */
                             /*  the default value.                           */
#define WARNING_107   107    /*  Cannot process parameter 7; it is assigned   */
                             /*  the default value.                           */
#define WARNING_108   108    /*  Cannot process parameter 8; it is assigned   */
                             /*  the default value.                           */
#define WARNING_109   109    /*  Cannot process parameter 9; it is assigned   */
                             /*  the default value.                           */
#define WARNING_110   110    /*  Cannot process parameter 10; it is assigned  */
                             /*  the default value.                           */
```

```
/*  WARNING_121 to WARNING_140 are general warnings.  */

#define WARNING_121   121    /*  Emulator already in the specified            */
                             /*  state/stage; no action is taken.             */
```

```
/*  WARNING_161 to WARNING_170 are routine dependent warnings.  */

#define WARNING_161   161
#define WARNING_162   162
#define WARNING_163   163
#define WARNING_164   164
#define WARNING_165   165
#define WARNING_166   166
#define WARNING_167   167
#define WARNING_168   168
#define WARNING_169   169
#define WARNING_170   170
```

```
/**************************************/
/*  Defines for Emulation Stages.  */
/**************************************/

#define   LEVEL1        0x0011
#define   LINK_CONTROL  0x0022
#define   L2_FULL       0x0032
#define   CALL_CONTROL  0x0043
#define   L3_ALL        0x0053
```

```
/*********************************************************/
/*  Defines for Emulation Program Process Priorities.  */
/*********************************************************/

#define   L1RCV_P        131    /* L1 Receiver's Priority     */
#define   L1XMT_P        130    /* L1 Transmitter's Priority  */
#define   L2EM_P         120    /* L2 Emulator's Priority     */
#define   L3EM_P         110    /* L3 Emulator's Priority     */
```

```
/******************************************/
/*  X.25 Message Types and Subtypes:  */
/******************************************/

/*  Type - SEND_DOWN.  */

#define SEND_DOWN   10

/*  SEND_DOWN Subtypes:  */
/*  None  */
```

```
/*  Type - SEND_ACK:  */

#define SEND_ACK    11

/*  SEND_ACK Subtypes:  */
/* None */
/*  Type - SEND_NACK:  */

#define SEND_NACK   12

/*  SEND_NACK Subtypes - reason_code:  */

#define LAPB_WINDOW_FULL     -141
#define LAPB_IN_WRONG_STATE  -142
#define LCI_WINDOW_FULL      -143
#define LCI_IN_WRONG_STATE   -144
#define LCI_INVALID          -145
```

```
/*  Type - EM_CMD:  */

#define EM_CMD      13

/*  EM_CMD Subtypes - command:  */

#define CALL_EST                1
#define CALL_CLEAR              2
#define CHANNEL_RESET           3
#define RESTART_L3              4
#define LINKUP                  5
#define LINKDOWN                6
#define GET_LEADS               7
#define CHG_STAGE               8
#define CONFIG                  9
#define START_DATA             10
#define EXIT                   11
```

```
/*  Type - EM_ACK  */

#define EM_ACK      14

/*  Type EM_ACK Subtypes:  */

/*  None  */
```

```
/*  Type - EM_NACK  */

#define EM_NACK      15

/*  EM_NACK Subtypes - reason_ code  */

/*  #define LAPB_IN_WRONG_STATE  -142  Already defined.  */
/*  #define LCI_IN_WRONG_STATE   -144  Already defined.  */
/*  #define LCI_INVALID          -145  Already defined.  */

#define ALL_LCIS_USED            -146
#define INVALID_STAGE            -147
```

```
/*  Type - USER_DATA  */

#define USER_DATA   204

/*  USER_DATA Subtype - data type.  */

#define DTE_FRAME        1
#define DCE_FRAME        2
#define SDTE_FRAME       3
#define SDCE_FRAME       4
```

```
/*  Type - USER_STATUS  */

#define USER_STATUS 205

/*  USER_STATUS Subtype - status type.  */

/*  #define CALL_EST                1   Already defined  */
/*  #define CALL_CLEAR              2   Already defined  */
/*  #define CHANNEL_RESET           3   Already defined  */
/*  #define RESTART_L3              4   Already defined  */

#define LAPB_STATE_CHANGE           8
#define L3_ERROR                    9
#define L3_DIAG                    10
```

```
typedef struct
   {
    char           *send_string_ptr; /*  Pointer to the string to be sent  */
    INT16          num_bytes;        /*  Number of bytes in the string     */
    INT32          senders_QID;      /*  QID of the sending process        */
    X25_SEND_INFO  *send_info_ptr;   /*  Pointer to send down information  */
   } X25_SEND;
```

```
/*******************************/
/*  EM_CMD Message Structure  */
/*******************************/

typedef struct
   {
    INT16  em_cmd_LCI;
            char   *em_cmd_pkt_ptr;    /*  Pointer to packet to send to     */
                                       /*  initiate the requested protocol  */
                                       /*  event                            */
    INT16  em_cmd_pkt_length;          /*  Length of packet to be sent      */
   } EM_CMD_L3;
```

```
typedef struct
   {
   INT16  em_cmd_new_value;  /*  Used only for CHG_STAGE:                 */
                             /*  new_value = new_stage                    */
   } EM_CMD_GENERAL;
```

```
typedef struct
   {
   INT8  em_cmd_parm_ID;     /*  Used only for CONFIG:  parm_ID = GLBS_PTR */
   INT32 em_cmd_parm_value;
   } EM_CMD_CONFIG;
```

```
typedef struct
   {
   INT32   em_cmders_QID;  /*  QID of the sender of the command union      */
      {
      EM_CMD_L3      em_cmd_l3;
      EM_CMD_GENERAL em_cmd_gen;
      EM_CMD_CONFIG  em_cmd_config;
      } em_cmd_info;
   } X25_CMD;
```

```
/*******************************/
/*  EM_ACK Message Structure  */
/*******************************/

typedef struct
   {
    INT16 em_ack_ret_val;  /*  Used only for GET_LEADS to return the lead  */
                           /*  status                                      */
   } X25_CMD_ACK;
```

```
/************************************/
/*  USER_STATUS Message Structure  */
/************************************/

typedef struct
   {
   void  *ustatus_event_ptr;  /*  Pointer to data passed to user:          */
                              /*   NIL if not used -                       */
                              /*     LAPB_STATE_CHANGE                     */
                              /*           OR                              */
                              /*  Points to the packet event -             */
                              /*    CALL,CLEAR,RESET, RESTART              */
                              /*    CALL_EST, CALL_CLEAR,                  */
                              /*    CHANNEL_RESET and RESTART_L3           */
                              /*          OR                               */
                              /*  Points to the packet event that caused   */
                                    the L3_ERROR or L3_DIAG                */
   INT16 ustatus_LCI;         /*  Used for CALL_EST, CALL_CLEAR,           */
                              /*    CHANNEL_RESET, RESTART_L3, L3_ERROR,   */
                              /*    and L3_DIAG.                           */
   INT8  ustatus_reason_code; /*  Used for CALL_EST to indicate NORMAL or  */
                              /*    COLLISION                              */
                              /*  Used for L3_ERROR to indicate error #    */
                              /*  Used for L3_DIAG  to indicate            */
                              /*    diagnostic #                           */
                              /*  Used for LAPB_STATE_CHANGE to indicate   */
                              /*    new LAPB state                         */
   } X25_STATUS;
```

```
/**********************************/
/*  USER_DATA Message Structure  */
/**********************************/

typedef struct
   {
    void  *udata_event_ptr;
    INT8 *udata_data_ptr;
    INT16 udata_length;
   } X25_DATA;
```

```
/***************************/
/*  X25 Message Structure  */
/***************************/

typedef struct
   {
    INT32        link;                       /*  RESERVED  */
    INT32        home_exch;                  /*  RESERVED  */
    INT8         type;
    INT8         subtype;
    union {
      X25_SEND      send_down;
      X25_CMD       em_cmd;
      X25_CMD_ACK   em_ack;
      X25_STATUS    status;
      X25_DATA      data;
      } body;
   } X25_MESSAGE;
```

```
/****************************************************************************/
/*                        HP message structure                              */
/*                                                                          */
/*  This should be the type of ALL messages used when programming using     */
/*  the X.25 Test Environment package.                                      */
/*                                                                          */
/****************************************************************************/

typedef union {
   MESSAGE   message;
   X25_MESSAGE x25_message;
  } HP_MESSAGE;
```

```
/***************************/
/*  X25 Message Constants  */
/***************************/

/*  EM_CMD_CONFIG message  */

/* parm_ID */
#define GLBS_PTR    1
```

```
/*  SEND_DOWN message  */

/* pf,nr,ns,ps,pr */
#define UNSPECIFIED  0xff
```

```
/*  USER_STATUS message  */

/*  reason_code  */
#define NORMAL_CALL  0
#define COLLISION    1
```

```
/***********************/
/*  General Constants  */
/***********************/

#define MOD_8           1
#define MOD_128         2

#define NO_PVC          0
#define NO_SVC          0

#define NOT_CONFIGURED  0
```

```
/**************************/
/*  LAPB State Constants  */
/**************************/

#define DISC_PHASE     1   /*  Disconnect Phase                          */
#define LINK_DISC      2   /*  Link Disconnect                           */
#define LINK_SETUP     3   /*  Link Setup                                */
#define FRMR_STATE     4   /*  Frame Reject                              */
#define NORMAL_DT      5   /*  Normal Data Transfer                      */
#define LOCAL_BUSY     6   /*  Local Station Busy                        */
#define REMOTE_BUSY    7   /*  Remote Station Busy                       */
#define BOTH_BUSY      8   /*  Both Stations Busy                        */
#define WAIT_ACK       9   /*  Waiting for Acknowledgment;               */
#define WAIT_ACK_LB   10   /*  Waiting for Acknowledgment Local Station  */
                           /*  Busy                                      */
#define WAIT_ACK_RB   11   /*  Waiting for Acknowledgment Remote Station */
                           /*  Busy                                      */
#define WAIT_ACK_BB   12   /*  Waiting for Acknowledgment Both Stations  */
                           /*  Busy                                      */
#define REJ_SENT      13   /*  Reject Sent                               */
#define REJ_SENT_LB   14   /*  Reject Sent; Local Station Busy           */
#define REJ_SENT_RB   15   /*  Reject Sent; Remote Station Busy          */
#define REJ_SENT_BB   16   /*  Reject Sent; Both Stations Busy           */
```

```
***************************/
/*  LAPB Frame Constants  */
/**************************/

#define SUBSCRIBER    0
#define NETWORK       1
```

```
/*  Poll/final bit constants  */

#define POLL_1          1
#define POLL_0          0
#define FINAL_1         1
#define FINAL_0         0
```

```
/*  Frame Type Identifier:  FTI  */

#define I_FTI           1     /*  Information frames  */
#define S_FTI           2     /*  Supervisory frames  */
#define U_FTI           3     /*  Unnumbered frames   */
```

```
/*  Frame Type :  FT  */

#define  I                    0x0000
#define  SABM                 0x002F
#define  SABME                0x006F
#define  DISC                 0x0043
#define  DISCM                0x000F
#define  UA                   0x0063
#define  FRMR                 0x0087
#define  RR_L2                0x0001
#define  RNR_L2               0x0005
#define  REJ_L2               0x0009
```

```
/*  COMMAND or RESPONSE Constants  */

#define COMMAND         256
#define RESPONSE        257
```

```
/*************************/
/*  X.25 Packet Defines  */
/*************************/

/*  Packet Type:  PT  */

#define CALL            0x000B   /*  Incoming call/call request  */
#define CALLC           0x000F   /*  Call accepted/connected     */
#define CLEAR           0x0013   /*  Clear request/indication    */
#define CLEARC          0x0017   /*  Clear confirmation          */
#define DATA            0x0000   /*  Data packet                 */
#define INT             0x0023   /*  Interrupt                   */
#define INTC            0x0027   /*  Interrupt confirm           */
#define RR_L3           0x0001   /*  Level 3 RR                  */
#define RNR_L3          0x0005   /*  Level 3 RNR                 */
#define REJ_L3          0x0009   /*  DTE level 3 reject          */
#define RESET           0x001B   /*  Reset request/indication    */
#define RESETC          0x001F   /*  Reset confirm               */
#define RESTART         0x00FB   /*  Restart request/indication  */
#define RESTARTC        0x00FF   /*  Restart confirmation        */
#define DIAG            0x00F1   /*  Diagnostic                  */
#define REG             0x00F3   /*  Registration request        */
#define REGC            0x00F7   /*  Registration confirm        */
#define UNDEFINE        0x00F5   /*  Undefined Packet            */
```

```
/***************************/
/*  X25 Decode Constants   */
/***************************/

/*  Field Length  */

#define X25_MIN_LEN        3
#define MAX_USERLEN      128
#define MAX_FACILLEN     109
#define X25_MAX_ERR       11
#define INVALID_PACKET      -1  /*  Packet length < MIN_LEN(3) - unable to */
                                /*    decode                               */
#define PACKET_LONG      0x0800 /*  Packet length is too long for the      */
                                /*     packet type                         */
#define PACKET_SHORT     0x0400 /*  Packet < min length of the packet type */
#define Qbit_ERR         0x0200 /*  Invalid Qbit for the packet type       */
#define Dbit_ERR         0x0100 /*  Invalid Dbit for the packet type       */
#define SN_ERR           0x0080 /*  Invalid SN -- 0 or 3                   */
#define LCI_ERR          0x0040 /*  Invalid LCI for the packet type        */
#define ADDR_LEN_ERR     0x0020 /*  addr len > remainder of the packet     */
#define NONBCD_ERR       0x0010 /*  Non BCD number in addr field           */
#define REGIST_LEN_ERR   0x0008 /*  regist len > remainder of pt or 109    */
#define FACIL_LEN_ERR    0x0004 /*  facil length > remainder of packet or  */
                                /*    > 109                                */
#define USER_LEN_ERR     0x0002 /*  user length > remainder of packet or   */
                                /*    > 128                                */
#define PTI_ERR          0x0001 /*  Undefined PTI                          */
```

```
/*  Facility Constants  */

#define RESERVED          0xFF
#define FACIL_ERROR       0xFFFFFFFF
#define FACIL_OK          0x00
```

```
/*  Standard Facility Codes  */

#define THROUGH           0x02
#define CUG_B_FORMAT      0x03
#define CUG_OUT_B_FORMAT  0x09
#define REVERSE_FAST      0x01
#define CHARGE_REQUEST    0x04
#define CALLED_ADDR_MOD   0x08
#define MARKER            0x00
#define PACK_SIZE         0x42
#define WIND_SIZE         0x43
#define CUG_E_FORMAT      0x47
#define CUG_OUT_E_FORMAT  0x48
#define BILAT_CUG         0x41
#define RPOA_B            0x44
#define TRANSIT           0x49
#define NUI_SELECT        0xC6
#define MONETARY_CHRG     0xC5
#define SEGMENT_COUNT     0xC2
#define CALL_DURATION     0xC1
#define RPOA_E            0xC4
#define CALL_DEFLECT      0xD1
#define CALL_REDIR_NOT    0xC3
#define EXTENSION         0xFF
```

```
/*  CCITT Specified Facility Codes  */

#define CALLED_ADDR_EXT   0xCB
#define CALLING_ADDR_EXT  0xC9
#define MINIMUM_THROUGH   0x0A
#define END_TO_END_TRANS  0xCA
#define EXPEDITE_FACIL    0x0B
```

```
/*  Registration Field Codes  */

#define NEGOT_FACILS_P1     0x05
#define NEGOT_FACILS_ANY    0x45
#define AVAIL_FACILS        0x46
#define NON_NEGOT_FACILS    0x06
#define DEFAULT_THROUGHPUT  0x02
#define DEFAULT_PACKET      0x42
#define DEFAULT_WINDOW      0x43
#define LCI_RANGES          0xC8
```

```
/*  facility error defines  */

#define FACIL_LENGTH_ERR  0x01
#define FACILS_TOO_LONG   0x02
#define BCD_ERROR         0x04
#define NUM_RANGE_ERROR   0x08
#define UNKNOWN_FACILITY  0x10
#define DUPLICATE_FACIL   0x20
#define LCI_RANGE_ERROR   0x40
```

```
/*facility category defines  */

#define STANDARD_FACILS    0x00
#define CCITT_SPEC_FACILS  0x01
#define REGISTRATIONS      0x02
#define UNKNOWN_CATEGORY   0x04

typedef struct {
    unsigned char  called_code;
    unsigned char  calling_code;
    unsigned int  called_value;
    unsigned int  calling_value;
   } PACKET_SIZE;
```

```
typedef struct {
    unsigned char called_value;
    unsigned char calling_value;
   } WINDOW_SIZE;
```

```
typedef struct {
    unsigned char  code;
    unsigned char  called_code;
    unsigned char  calling_code;
    unsigned int  called_value;
    unsigned int  calling_value;
   } THROUGHPUT;
```

```
typedef struct {
    unsigned int  bcd;
    unsigned int  hex;
   } CUG_RPOA;
```

```
typedef struct {
    unsigned char  code;
    unsigned char  rev_code;
    unsigned char  fs_code;
   } REV_FS;
```

```
typedef struct {
    unsigned char  *id;
  } NUI_SEL;
```

```
typedef struct {
    unsigned char  code;
    unsigned char  chrg_code;
  } CHARGE_REQ;
```

```
typedef struct {
    unsigned char  *charge;
  } MONETARY;
```

```
typedef struct
   {
    unsigned char to_result;
    unsigned char from_result;
    unsigned long to_bcd;
    unsigned long to_hex;
    unsigned long from_bcd;
    unsigned long from_hex;
   } PERIOD;
```

```
typedef struct
   {
     unsigned char  num_periods;
     PERIOD         period[14];
   } SEG_COUNT;
```

```
typedef struct {
    SEG_COUNT *count;
   } SEGMENTS;
```

```
typedef struct
   {
    unsigned char  results;
    unsigned char  days;
    unsigned char  hours;
    unsigned char  mins;
    unsigned char  secs;
   } TIME_COUNT;
```

```
typedef struct
   {
     unsigned char  num_periods;
     TIME_COUNT time_count[26];
   } TIME;
```

```
typedef struct {
    TIME *time;
  } DURATION;
```

```
typedef struct {
    unsigned char results;
    unsigned int bcd;
    unsigned int hex;
  } ID;
```

```
typedef struct {
    unsigned char  num_periods;
    ID             id[55];
  } RPOA;
```

```
typedef struct {
    RPOA *ids;
  } RPOA_EXT;
```

```
typedef struct {
    unsigned char  reason_code;
    unsigned char  dte_defl_bits;
    unsigned char  reason;
    unsigned char  addr_length;
    unsigned char  *addr;
  } CALL_DEF;
```

```
typedef struct {
    unsigned char  code;
    unsigned char  dte_defl;
    unsigned char  reason;
  } ADDR_MOD;
```

```
typedef struct {
    unsigned int  delay;
  } TRANS_DELAY;
```

```
typedef struct {
    unsigned char  type;
  } MARK;
```

```
typedef struct {
    unsigned char  *ptr;
  } EXTENS;
```

```
typedef struct {
    unsigned char   code;
    unsigned char   encoding;
    unsigned char   addr_length;
    unsigned char   *addr;
  } ADDR_EXT;
```

```
typedef struct {
    unsigned int  cumul_delay;
    unsigned int  req_delay;
    unsigned int  max_delay;
   } TRANS_ETE;
```

```
typedef struct {
    unsigned char   code;
    unsigned char   exp_code;
   } EXPED;
```

```
typedef struct {
    unsigned char   byte1;
    unsigned char   byte2;
   } REGIS;
```

```
typedef struct {
    unsigned int    lic;
    unsigned int    hic;
    unsigned int    ltc;
    unsigned int    htc;
    unsigned int    loc;
    unsigned int    hoc;
    unsigned int    num_lcis;
   } LCIS;
```

```
typedef struct {
    LCIS *lcis;
   } LCI_RANG;
```

```
typedef struct {
    unsigned char   *ptr;
   } UNKNOWN;
```

```
typedef struct {
    unsigned char  facil_results;
    unsigned char  facil_category;
    unsigned char  facil_code;
    int            facil_lgth;
    union  {
        PACKET_SIZE   packet_size;
        WINDOW_SIZE   window_size;
        THROUGHPUT    throughput;
        CUG_RPOA      cug_rpoa;
        REV_FS        rev_fs;
        NUI_SEL       nui_sel;
        CHARGE_REQ    charge_req;
        MONETARY      monetary;
        SEGMENTS      segments;
        DURATION      duration;
        RPOA_EXT      rpoa_ext;
        CALL_DEF      call_def;
        ADDR_MOD      addr_mod;
        TRANS_DELAY   trans_delay;
        MARK          marker;
        EXTENS        extension;
        ADDR_EXT      addr_ext;
        TRANS_ETE     trans_ete;
        EXPED         expedite;
        REGIS         regis;
        LCI_RANG      lci_ranges;
        UNKNOWN       unknown;
      } value;
   } FACIL;
```

```
typedef struct {
    unsigned char results;
    unsigned char num_facils;
    unsigned char facils_length;
    FACIL facility[55];  /*  Max total facil length (109)/min   */
                         /*  Individual facil length (2)        */
   } FACILITIES;
```

```
/**********************************/
/*  X25 Decode Output Structure  */
/**********************************/

typedef struct{
    int             CDAlen ;      /*  Called Address Length   */
    int             CGAlen ;      /*  Calling Address Length   */
    unsigned char   Calling[128];
    unsigned char   Called [128];
    unsigned char   Cause;
    unsigned char   Diag;
    unsigned char   *pFacil;
    int             FacilLen;
   } X25ADDR_PACKET;
```

```
typedef struct {
    unsigned char    Cause;
    unsigned char    Diag;
   } X25SUPERV;
```

```
typedef struct {
    int             DiagLen;
    unsigned char   Diag;
    unsigned char   DiagExpl[4];
   } X25DIAG;
```

```
typedef struct{
    unsigned char   PR;
    unsigned char   Mbit;
    unsigned char   PS;
   } X25DATA;
```

```
typedef  union {
    X25ADDR_PACKET      Addr_FacilPacket;
    X25SUPERV           SupervPacket;
    X25DIAG             DiagPacket;
    X25DATA             DataPacket;
   } X25PACKETS;
```

```
typedef struct{
    int              result;       /*  For errors                          */
    int              is_dce ;      /*  DCE_X25/DTE_X25,                    */
                                   /*  1/0,TRUE/FALSE                      */
    int              packet_len;   /*  Number of bytes in packet.          */
    unsigned char    *packet_ptr;  /*  Pointing at the start of the packet */
    int              l4_raw_len;   /*  Number of bytes in level4 info      */
    unsigned char    *l4_raw_ptr;  /*  Pointing at the start of level4 info */

    /*  X.25 packet fields:  */

    unsigned char    Qbit;
    unsigned char    Dbit;
    unsigned char    SN;
    unsigned char    LCGN
    unsigned char    LCN;
    int              LCI;
    unsigned char    PTI;
    unsigned char    PR
    FACILITIES       *facil_struct;  /*  Pointer to facility structure     */
    X25PACKETS       Packet_Type;
  } DEC_X25;
```

```
/**********************************/
/*  X25 Level 3 State Constants  */
/**********************************/

/*  LCI 0 Packet Level State  */

#define PACKET_LEVEL_READY      1    /*  R1  */
#define DTE_RESTART_REQ         2    /*  R2  */
#define DCE_RESTART_IND         3    /*  R3  */
```

```
/*  LCIs 1-4,095 LCI State  */

#define READY                   4    /*  P1  */
#define DTE_CALL_REQ            5    /*  P2  */
#define DCE_CALL_IND            6    /*  P3  */
#define CALL_COLLISION          7    /*  P5  */
#define DTE_CLEAR_REQ           8    /*  P6  */
#define DCE_CLEAR_IND           9    /*  P7  */
#define DTE_RESET_REQ          10    /*  D2  */
#define DCE_RESET_IND          11    /*  D3  */
#define DT_NORMAL              12    /*  D1 (P4)  */
#define DT_REMOTE_LCI_BUSY     13    /*  New state used for emulation -    */
                                     /*    not in X.25 CCITT spec          */
```

```
/***************************************************/
/*  X25 Level 3 Error and Diagnostic Code Values   */
/***************************************************/

/*  Values for reason_codes for USER_STATUS messages when type is L3_ERROR */

#define DC_BAD_PS                        1
#define DC_BAD_PR                        2
#define DC_PTI_INVALID_STATE_R1         17
#define DC_PTI_INVALID_STATE_R2         18
#define DC_PTI_INVALID_STATE_R3         19
#define DC_PTI_INVALID_STATE_P1         20
#define DC_PTI_INVALID_STATE_P2         21
#define DC_PTI_INVALID_STATE_P3         22
#define DC_PTI_INVALID_STATE_P4         23
#define DC_PTI_INVALID_STATE_P5         24
#define DC_PTI_INVALID_STATE_P6         25
#define DC_PTI_INVALID_STATE_P7         26
#define DC_PTI_INVALID_STATE_D1         27
#define DC_PTI_INVALID_STATE_D2         28
#define DC_PTI_INVALID_STATE_D3         29
#define DC_UNDEF_PACKET                 33
#define DC_INVALID_PTI_FOR_PVC          35
#define DC_PACKET_SHORT                 38
#define DC_PACKET_LONG                  39
#define DC_LCIOPKT_W_LCINONO            41  /*  Packet which is only allowed  */
                                            /*    on LCI 0 has non-zero LCI   */
#define DC_FACIL_CODE_NOT_ALLOWED       65
#define DC_FACIL_PARM_NOT_ALLOWED       66
#define DC_FACIL_LEN_ERR                69
#define DC_DUPLICATE_FACILITY           73
```

```
/*  The following are non-standard diagnostic codes.  The comments define  */
/*  the specific errors.                                                   */

#define DC_ADDR_LEN_ERR        101   /*  Length of the address field is too   */
                                     /*   long or short                       */
#define DC_USER_LEN_ERR        102   /*  Length of the user data field is too */
                                     /*   long or short                       */
#define DC_NONBCD_ERR          103   /*  One or more of the address digits is */
                                     /*    not a BCD digit                    */
```

```
/*  Values for reason_codes for USER_STATUS messages when type is L3_DIAG  */

#define DC_UNASSIGNED_LCI        36
/* #define DC_PACKET_SHORT       38  Already defined */
#define DC_INVALID_GFI           40
```

```
/**************************/
/*  LAPB Decode Defines  */
/**************************/

#define UNDEF          0xFF

/*  Control Field Format  */

#define INFO_LAPB       0
#define SUPERV          1
#define UNNUM           3
```

```
/*  Control field reference values  */

#define I_REF     0
#define RR_REF    1
#define RNR_REF   2
#define REJ_REF   3
#define SABM_REF  4
#define SABME_REF 5
#define DISC_REF  6
#define DM_REF    7
#define UA_REF    8
#define FRMR_REF  9
#define UNDEF_REF 10
```

```
/*  Decode result constants  */

#define OK              0       /*  No error during decoding                */
#define INVALID_FRAME   0x01    /*  Invalid frame                           */
#define ADDR_ERR        0x02    /*  Wrong value in address field            */
#define FRMR_TOO_SHORT  0x04    /*  FRMR Frame is too short to completely   */
                                /*    decode                                */
#define UNEX_INFO       0x08    /*  Supervisory or Unnumbered frame is too  */
                                /*    long.                                 */
```

```
/*********************************/
/* LAPB Decode Output Structure  */
/*********************************/

/*  FRMR rejected frame information  */

typedef struct {
    INT16   ctrl;        /*  Control field of rejected frame               */
    INT8    Vr;          /*  Value of V(r)                                 */
    INT8    Vs;          /*  Value of V(s)                                 */
    INT8    command;     /*  If TRUE, rejected frame is a command frame    */
    INT8    W_bit;       /*  Value of bit W                                */
    INT8    X_bit;       /*  Value of bit X                                */
    INT8    Y_bit;       /*  Value of bit Y                                */
    INT8    Z_bit;       /*  Value of bit Z                                */
  } FRMR_INFO;
```

```
/*  Decode of frame  */
typedef struct {
    INT8       result;      /*  If INVALID: error in length, only            */
                            /*    L2raw_lgth and L2raw_ptr are defined       */
    INT32      L2raw_lgth;  /*  0 if there is no level2 frame information    */
    INT8       *L2raw_ptr;  /*  NIL if no level 2 frame information          */
                            /*--------------- ADDRESS FIELD ---------------*/
    INT8       addr;        /*  Value of address field                       */
    INT8       command;     /*  If not is a response                         */
                            /*--------------- CONTROL FIELD ---------------*/
    INT16      ctrl_field;  /*  Entire control field                         */
    INT8       pf_bit;      /*  Value of P/F bit;                            */
    INT8       ctrl_format; /*  Unnumbered:  Nr and Ns not used              */
                            /*  Supervisory: Nr used, Ns not used            */
                            /*  Other:  Nr and Ns used                       */
    INT8       Ns;          /*  Value of N(s)                                */
    INT8       Nr;          /*  Value of N(r)                                */
    INT8       ctrl_id;     /*  Identifier of control field                  */
    INT8       ctrl_ref;    /*  Values from 0 to 10 which have been          */
                            /*    randomly assigned to the various frame     */
                            /*    types.  Refer to the LAPB Decode Defines   */
                            /*    section above.                             */
                            /*----------------- INFO FIELD ----------------*/
    FRMR_INFO frmr;         /*  FRMR rejected frame control field            */
                            /*    information.  Only used if                 */
                            /*    ctrl_id == FRMR                            */
    INT16      L3raw_lgth;  /*  Length of L3 packet, 0 if not used.          */
    INT8       *L3raw_ptr;  /*  Pointer to L3 packet, 0 if not used.         */
  } LAPB_DECODE;
```

```
/************************/
/*  X25 Message Macros  */
/************************/

/*  In the following, p is of type HP_MESSAGE.  */

/*  Accessing the EM_CMD message structure  */

#define EM_CMD_MSG(p)             (p).x25_message.body.em_cmd
#define EM_CMDERS_QID(p)          EM_CMD_MSG(p).em_cmders_QID
#define EM_CMD_L3(p)              EM_CMD_MSG(p).em_cmd_info.em_cmd_l3
#define EM_CMD_GEN(p)             EM_CMD_MSG(p).em_cmd_info.em_cmd_gen
#define EM_CMD_CONFIG(p)          EM_CMD_MSG(p).em_cmd_info.em_cmd_config
#define EM_CMD_LCI(p)             EM_CMD_L3(p).em_cmd_LCI
#define EM_CMD_PKT_PTR(p)         EM_CMD_L3(p).em_cmd_pkt_ptr
#define EM_CMD_PKT_LENGTH(p)      EM_CMD_L3(p).em_cmd_pkt_length
#define EM_CMD_NEW_VALUE(p)       EM_CMD_GEN(p).em_cmd_new_value
#define EM_CMD_PARM_ID(p)         EM_CMD_CONFIG(p).em_cmd_parm_ID
#define EM_CMD_PARM_VALUE(p)      EM_CMD_CONFIG(p).em_cmd_parm_value
```

```
/*  Accessing the EM_ACK message structure  */

#define EM_ACK_MSG(p)             (p).x25_message.body.em_ack
#define EM_ACK_RET_VAL(p)         EM_ACK_MSG(p).em_ack_ret_val
```

```
/*  Accessing the USER_STATUS message structure  */

#define STATUS_MSG(p)             (p).x25_message.body.status
#define USTATUS_EVENT_PTR(p)      STATUS_MSG(p).ustatus_event_ptr
#define USTATUS_LCI(p)            STATUS_MSG(p).ustatus_LCI
#define USTATUS_REASON_CODE(p)    STATUS_MSG(p).ustatus_reason_code
```

```
/*  Accessing the SEND_DOWN message structure  */

#define SEND_DOWN_MSG(p)          (p).x25_message.body.send_down
#define SEND_STRING_PTR(p)        SEND_DOWN_MSG(p).send_string_ptr
#define NUM_BYTES(p)              SEND_DOWN_MSG(p).num_bytes
#define SENDERS_QID(p)            SEND_DOWN_MSG(p).senders_QID
#define SEND_INFO_PTR(p)          SEND_DOWN_MSG(p).send_info_ptr
#define FCS_TYPE_L1(p)            SEND_INFO_PTR(p)->l1_info.fcs_type
#define FCS_TYPE_L2(p)            SEND_INFO_PTR(p)->lapb_info.fcs_type
#define PF(p)                     SEND_INFO_PTR(p)->lapb_info.pf
#define NS(p)                     SEND_INFO_PTR(p)->lapb_info.ns
#define NR(p)                     SEND_INFO_PTR(p)->lapb_info.nr
#define LCI(p)                    SEND_INFO_PTR(p)->x25l3_info.LCI
#define QBIT(p)                   SEND_INFO_PTR(p)->x25l3_info.Qbit
#define DBIT(p)                   SEND_INFO_PTR(p)->x25l3_info.Dbit
#define SN(p)                     SEND_INFO_PTR(p)->x25l3_info.SN
#define MBIT(p)                   SEND_INFO_PTR(p)->x25l3_info.Mbit
#define PR(p)                     SEND_INFO_PTR(p)->x25l3_info.pr
#define PS(p)                     SEND_INFO_PTR(p)->x25l3_info.ps
```

```
/*  Accessing the USER-DATA message structure  */

#define DATA_MSG(p)            (p).x25_message.body.data
#define UDATA_EVENT_PTR(p)     DATA_MSG(p).udata_event_ptr
#define UDATA_DATA_PTR(p)      DATA_MSG(p).udata_data_ptr
#define UDATA_LENGTH(p)        DATA_MSG(p).udata_length
```

```
/*  Accessing the X25 Message Type and Subtype  */

#define X25_TYPE(p)            (p).x25_message.type
#define X25_SUBTYPE(p)         (p).x25_message.subtype
```

```
/*******************************************/
/*  X25 Library Routine name Definitions:  */
/*******************************************/

#define  config_device           config_device_X25
#define  config_clock            config_clock_X25
#define  config_datacode_parity  config_datacode_parity_X25
#define  get_device              get_device_X25
#define  config_extctrl          config_extctrl_X25
#define  config_frame_addrs      config_frame_addrs_X25
#define  config_logical_device   config_logical_device_X25
#define  config_k                config_k_X25
#define  config_N1               config_N1_X25
#define  config_N2               config_N2_X25
#define  config_T1               config_T1_X25
#define  config_RR_idle_L2       config_RR_idle_L2_X25
#define  get_extctrl             get_extctrl_X25
#define  get_frame_addrs         get_frame_addrs_X25
#define  get_logical_device      get_logical_device_X25
#define  get_k                   get_k_X25
#define  get_N1                  get_N1_X25
#define  get_N2                  get_N2_X25
#define  get_T1                  get_T1_X25
#define  get_state_L2            get_state_L2_X25
#define  get_frame               get_frame_X25
#define  get_NR                  get_NR_X25
#define  get_NS                  get_NS_X25
#define  get_busy_L2             get_busy_L2_X25
#define  get_RR_idle_L2          get_RR_idle_L2_X25
#define  set_busy_L2             set_busy_L2_X25
#define  clear_busy_L2           clear_busy_L2_X25
#define  send_DISC               send_DISC_X25
#define  send_DM                 send_DM_X25
#define  send_FRMR               send_FRMR_X25
#define  send_REJ_L2             send_REJ_L2_X25
#define  send_RNR_L2             send_RNR_L2_X25
#define  send_RR_L2              send_RR_L2_X25
#define  send_SABM               send_SABM_X25
#define  send_SABME              send_SABME_X25
#define  send_UA                 send_UA_X25
#define  send_I                  send_I_X25
#define  decode_frame            decode_frame_X25
#define  call_clear              call_clear_X25
#define  call_establish          call_establish_X25
#define  reset_channel           reset_channel_X25
#define  restart_L3              restart_L3_X25
#define  config_LCI_ranges       config_LCI_ranges_X25
#define  config_SN_L3            config_SN_L3_X25
```

```
#define  get_Dbit               get_Dbit_X25
#define  get_facil_CALL         get_facil_CALL_X25
#define  get_facil_CALLC        get_facil_CALLC_X25
#define  get_facil_CLEAR        get_facil_CLEAR_X25
#define  get_facil_CLEARC       get_facil_CLEARC_X25
#define  get_Abit               get_Abit_X25
#define  get_LCI                get_LCI_X25
#define  get_LCI_ranges         get_LCI_ranges_X25
#define  get_packet             get_packet_X25
#define  get_PR                 get_PR_X25
#define  get_PS                 get_PS_X25
#define  get_Qbit               get_Qbit_X25
#define  get_state_L3           get_state_L3_X25
#define  get_SN_L3              get_SN_L3_X25
#define  get_userdata_CALL      get_userdata_CALL_X25
#define  get_userdata_CALLC     get_userdata_CALLC_X25
#define  get_userdata_CLEAR     get_userdata_CLEAR_X25
#define  get_windowsize_L3      get_windowsize_L3_X25
#define  set_Dbit               set_Dbit_X25
#define  set_facil_CALL         set_facil_CALL_X25
#define  set_facil_CALLC        set_facil_CALLC_X25
#define  set_facil_CLEAR        set_facil_CLEAR_X25
#define  set_facil_CLEARC       set_facil_CLEARC_X25
#define  set_Abit               set_Abit_X25
#define  set_LCI                set_LCI_X25
#define  set_Qbit               set_Qbit_X25
#define  set_userdata_CALL      set_userdata_CALL_X25
#define  set_userdata_CALLC     set_userdata_CALLC_X25
#define  set_userdata_CLEAR     set_userdata_CLEAR_X25
#define  set_windowsize_L3      set_windowsize_L3_X25
#define  send_CALL              send_CALL_X25
#define  send_CALLC             send_CALLC_X25
#define  send_CLEAR             send_CLEAR_X25
#define  send_CLEARC            send_CLEARC_X25
#define  send_DATA              send_DATA_X25
#define  send_DIAG              send_DIAG_X25
#define  send_INT               send_INT_X25
#define  send_INTC              send_INTC_X25
#define  send_REG               send_REG_X25
#define  send_REGC              send_REGC_X25
#define  send_RESET             send_RESET_X25
#define  send_RESETC            send_RESETC_X25
#define  send_RESTART           send_RESTART_X25
#define  send_RESTARTC          send_RESTARTC_X25
#define  send_REJ_L3            send_REJ_L3_X25
#define  send_RNR_L3            send_RNR_L3_X25
#define  send_RR_L3             send_RR_L3_X25
#define  decode_packet          decode_packet_X25
```