73 X.21 Library

The Test Interface Module (TIM) located in the rear of the INTERVIEW determines the leads available for monitoring and control (Section 12). The variables and routines in this section apply to the X.21 interface module. RS-232, V.35, and RS-449 modules are treated in Section 63.

To use the C variables and routines explained in this section, you must select **Buffer Control Leads:** YES on the FEB Setup menu. See Section 9.1(B). If no other source for clock is provided, use internal clock (Line Setup menu). Finally, load in the X.21 package via the Layer Setup screen.

The variables and routines approximate X.21 Layer 1 spreadsheet-generated conditions and actions. Their use on the Protocol Spreadsheet is not limited to any particular layer, though normally they belong at Layer 1. Refer to Section 35 for more detailed explanations of the purposes of specific conditions and actions. Sometimes the name of the variable or routine is sufficient for identifying its related spreadsheet token. When this is not the case, the information is provided below.

73.1 Structures

Use the structure *xmit_list*, shown in Table 73-1, when transmitting line data via the *x21_transmit_call* routine. Refer to *x21_transmit_call* in Section 73.3(A) for an example of how to use this structure.

Table 73-1
X.21 Structures

Type	Variable	Value (hex/decimal)	Meaning
Structure Name:	xmit_list		Structure of a transmit list for *x21_transmit_call* routine. Declared as type *struct*. Declared automatically if a softkey-entered CALL_SETUP_SEND action is taken. Reference member variables of the structure as follows: *xmit_list.string_length*
unsigned char *	string		pointer to the location of the transmit string—the transmit string is declared separately
unsigned short	string_length	*0-ffff/0-65535*	length of the transmit string

73.2 Variables

With an X.21 TIM installed, you may monitor the T and R data leads, the C and I control leads, and UA. See Table 73-2.

The fast-event variable *fevar_eia_changed* detects a change in leads. It does not establish which lead(s) has changed, nor the validity of the lead's status. Two associated status variables, *current_eia_leads* and *previous_eia_leads*, indicate the condition of the leads. These are two-byte (*short*) variables. Each lead is represented by a different bit in the *short*. Table 73-2 provides the mask that can be used to isolate each lead.

Other bits in these variables monitor the validity of lead status. For the status of a lead to be considered valid in X.21, the lead must be stable for a minimum of 16 bit-times. Each lead's valid status is indicated by a separate bit in *current_eia_leads* and *previous_eia_leads*. Again, refer to Table 73-2.

Whenever a lead changes, the value in *current_eia_leads* is written to *previous_eia_leads*. Then *current_eia_leads* is updated.

(A) Masking To Detect a Change in a Given Lead

To test whether or not a given lead changed, I for example, while disregarding its status, enter the following condition on the Protocol Spreadsheet:

```
CONDITIONS:
{
fevar_eia_changed && (((current_eia_leads ^ previous_eia_leads) & 0x40) == 0x40)
}
```

Select a mask value from the list in Table 73-2 to indicate which lead you care about. Specify multiple leads with a mask derived via hexadecimal addition.

The mask for I is 0x40. In the example, the event *fevar_eia_changed* updated *current_eia_leads*. The new *current_eia_leads* was *bitwise-exclusive-OR*ed with *previous_eia_leads* to identify all the leads that changed. Then the result was *bitwise AND*ed with the I mask to determine if I was among the leads that changed. If this result was equal to the mask, the lead changed.

Following the evaluation of the condition, *previous_eia_leads* was updated to match *current_eia_leads*.

Table 73-2
X.21 Variables

Type	Variable	Value (hex/decimal)	Meaning
extern fast_event	fevar_eia_changed		True when the status changes for an EIA lead. Line Setup configured for emulate or monitor mode.
extern const volatile unsigned short	current_eia_leads	1	C-valid
		2	B (RS-232 mapping is SQ)
		4	I-valid (RI)
		8	R-valid (DSR)
		10/16	T-valid (DTR)
		40/64	I (CTS)
		80/128	C (RTS)
		100/256	R (RD)
		200/512	UA
		400/1024	T (TD)
			A value in this list indicates which lead(s) you care about. When *anded* (&) with *current_eia_leads*, the result equals zero if the lead is *on* (or the mask if the lead is *off*). For validity checks, the result of *anding* with *current_eia_leads* equals the mask for *valid* (or zero for *invalid*).
			<u>Examples</u>:
			STATE: c_on_and_valid *{ if ((current_eia_leads & 0x81) == 1) sound_alarm(); }*
			STATE: c_off_and_valid *{ if ((current_eia_leads & 0x81) == 0x81) sound_alarm(); }*
			Note: This variable will store EIA status if (1) internal or external clock is supplied, (2) EIA leads are enabled on FEB Setup, and (3) *fevar_eia_changed* has updated the leads. Line Setup configured for emulate or monitor mode.
extern const volatile unsigned short	previous_eia_leads		Same values as *current_eia_leads*. Updated when leads change, but only after logic has had a chance to compare current and previous leads. Line Setup configured for emulate or monitor mode.

(B) Masking For the Status or Validity of a Lead

You may also test the current status or validity of a lead, independent of any change. If a mask testing for status is *and*ed with *current_eia_leads*, zero will mean that the lead in on. If the result equals the mask, the lead is off. If a mask testing for validity is *and*ed with *current_eia_leads*, the lead status is valid when the result equals the mask. If the result is zero, the status is invalid.

To test for both status and validity, derive a mask via hexadecimal addition. *And* the mask with *current_eia_leads*, as in this *if* statement testing for I "on" and valid:

```
STATE: test_for_i_on_and_valid
   {
    if((current_eia_leads & 0x44) == 4)  sound_alarm();
   }
```

(C) Detect Change and Current Status

The two examples shown above could be combined to test for I changing from off to valid on:

```
CONDITIONS:
{
  (fevar_eia_changed && (((current_eia_leads ^ previous_eia_leads) & 0x40) == 0x40) &&
       ((current_eia_leads & 0x44) == 4))
}
```

This example approximates the translator's version of the spreadsheet-token condition LEADS I V-ON when it appears alone in a conditions block. When a LEADS condition is combined with another condition, in most cases the other condition will supply the event variable and only the lead status test will be used.

73.3 Routines

(A) Control and Transmit

Use the following routines in emulate mode only. If you try to call one of these routines in monitor mode, you may be returned to the main program menu. When you go to the Protocol Spreadsheet and search for errors, a message like the following may be displayed: *"Error 140: Unresolved reference ctl_eia."*

ctl_eia

Synopsis

```
extern void ctl_eia(on_mask, off_mask);
unsigned short on_mask;
unsigned short off_mask;
```

Description

The *ctl_eia* routine allows you to control the status of the two X.21 control-leads. Which lead you control depends on your emulation mode. When the Line Setup menu shows **Mode:** EMULATE DCE, you control I. An EMULATE DTE selection gives you control over C. The softkey equivalent of this routine is the LEADS action on the Protocol Spreadsheet.

Inputs

The first parameter indicates which lead you want to turn on. One bit in the parameter controls a given lead: I (01) and C (04). Wherever there is a *zero* in the first parameter, the corresponding lead will be turned on. A one in this parameter will *not* cause any lead to be turned off. A value of 0xff will mean *don't care* (no action).

The second parameter indicates which lead you want in the "off" condition. One bit in the parameter controls a given lead: I (01) and C (04). Wherever there is a *one* in the second parameter, the corresponding lead will be turned off. Zeroes in this parameter do *not* turn leads on. A value of 0 will mean *don't care* (no action).

> NOTE: If both bytes are attempting to control the same lead, the off parameter will override the on parameter.

Example

Suppose your emulate mode is EMULATE DCE. As a DCE, you control the I lead. (An attempt to control the status of C will fail, since the DTE controls this lead.) When C is raised, you want to turn I on; when C drops, turn I off.

```
LAYER: 1
    STATE: control_I
       CONDITIONS: LEADS C ON
       ACTIONS:
       {
        ctl_eia(0xfe, 0x00);
       }
       CONDITIONS: LEADS C OFF
       ACTIONS:
       {
        ctl_eia(0xff, 0x01);
       }
```

x21_idle_action

Synopsis

```
extern void x21_idle_action(character);
unsigned char character;
```

Description

Only for format SYNC, the *x21_idle_action* routine allows you to change the idle-line condition applied by the INTERVIEW. A LEADS R BELLS action, for example, requires the *x21_transmit_call* routine in addition to *x21_idle_action*.

Inputs

The only parameter is a character or numeric value representing the idle character.

Example

To signal an incoming call, you would use the *x21_transmit_call* routine to send the sync pattern. Then you would use the *x21_idle_action* routine to send an idle string of *bells*:

```
LAYER: 1
{
 unsigned char syncs [] = {0x16,0x16};
 struct xmit_list
  {
   unsigned char * string;
   unsigned short string_length;
  };
 struct xmit_list send_string [] = {&syncs[0], 2};
}
      STATE: signal_incoming_call
          CONDITIONS: KEYBOARD " "
          ACTIONS:
          {
           x21_transmit_call(1, &send_string[0], 0);
           x21_idle_action('␇');
          }
```

x21_transmit_call

Synopsis

```
extern void x21_transmit_call(count, struct_send_string_ptr, xmit_tag);
unsigned short count;
struct xmit_list
   {
    char * string_ptr;
    unsigned short string_length;
   };
struct xmit_list * struct_send_string_ptr;
unsigned short xmit_tag;
```

Description

The *x21_transmit_call* routine sends a specified data string in call-setup mode. The softkey equivalent of this routine is the CALL_SETUP_SEND action.

Inputs

The first parameter is the number of strings to be sent.

The second parameter is a pointer to a structure which in turn identifies the location and length of each string.

The third parameter is a transmit tag. In other contexts it identifies the type of BCC to be sent. In X.21, however, no BCC is sent from Layer 1. The value of this parameter should be zero.

Example

Assume you are emulating a DTE. To send a call request in call-setup mode, enter the following spreadsheet program:

```
LAYER: 1
{
 unsigned char syncs [] = {0x16,0x16};
 unsigned char number [] = "1234567";
 unsigned char end [] = "+";
 struct xmit_list
  {
   unsigned char * string;
   unsigned short string_length;
  };
 struct xmit_list send_string [] = {&syncs[0], 2, &number[0], sizeof(number) - 1, &end[0], 1};
}
```

```
STATE: send
   CONDITIONS: RECEIVE STRING "🅂++"
   ACTIONS:
   {
    x21_transmit_call(3, &send_string[0], 0);
   }
```

Notice in the preceding example that sync characters were sent in the same call to *x21_transmit_call* that sent the called number. The equivalent softkey-generated action is LEADS T DATA CALL_SETUP_SEND "␖␖1234567+".

x21_transmit_call_idle

Synopsis

```
extern void x21_transmit_call_idle(count, struct_send_string_ptr, xmit_tag, new_idle);
unsigned short count;
struct xmit_list
   {
    char * string_ptr;
    unsigned short string_length;
   };
struct xmit_list * struct_send_string_ptr;
unsigned short xmit_tag;
unsigned char new_idle;
```

Description

The *x21_transmit_call_idle* routine sends a data string in call-setup mode and includes a specified idle character. The softkey equivalent of this routine is the CALL_SETUP_SEND_IDLE action. This routine differs from *x21_transmit_call* in that a change in idle character is guaranteed to occur during the transmission.

Inputs

The first parameter is the number of strings to be sent.

The second parameter is a pointer to a structure which in turn identifies the location and length of each string.

The third parameter is a transmit tag. In other contexts it identifies the type of BCC to be sent. In X.21, however, no BCC is sent from Layer 1. The value of this parameter should be zero.

The fourth parameter is the idle character. Enter the idle character as a decimal or hexadecimal value, or as a character enclosed by single quotes ('').

Example

This example is the same as the one for *x21_transmit_call* except that here the idle character, +, is included in the call to *x21_transmit_call_idle*.

```
LAYER: 1
{
 unsigned char syncs [] = {0x16,0x16};
 unsigned char number [] = "1234567";
 struct xmit_list
  {
   unsigned char * string;
   unsigned short string_length;
  };
 struct xmit_list send_string [] = {&syncs[0], 2, &number[0], sizeof(number) - 1};
}
     STATE: send
        CONDITIONS: RECEIVE STRING "🅂++"
        ACTIONS:
        {
         x21_transmit_call_idle(2, &send_string[0], 0, 0x2b);
        }
```

Notice in the example that sync characters were sent in the same call to *x21_transmit_call_idle* that sent the called number. The equivalent softkey-generated action is CALL_SETUP_SEND_IDLE "␖␖1234567" NEW_IDLE "+".

set_tcr_b

Synopsis

```
extern void set_tcr_b (tcr_register_mask, tcr_register_value);
unsigned char tcr_register_mask;
unsigned char tcr_register_value;
```

Description

This routine clamps the transmit line to 0 (space) or 1 (mark), or unclamps it so that transmit routines may be executed. In X.21, steady zero will signal a clear request/indication or a clear confirm, while steady 1 will indicate one of the call-ready or call-setup states.

The X.21 softkey actions that are built on this routine are LEADS R *(T)* ONE, LEADS R *(T)* ZERO, and LEADS R *(T)* DATA. In other contexts, the routine simply initiates and terminates a *break*.

Inputs

The first parameter is the mask that is *and*ed with the current TCR register to turn the current values of bits 3 and 4 (counting 1–8 from the right) to zero. This mask is always 0xf3.

The second parameter contains the new values of bits 3 and 4 that will be written to the register. The three available parameters are 0x08 to clamp the line to zero, 0x0c to clamp the line to 1, and 0x04 to unclamp the line and permit data transmissions.

Example

Assume you are emulating a DTE. To indicate a clear confirmation, enter the following spreadsheet program:

```
LAYER: 1
     STATE:
        CONDITIONS: KEYBOARD " "
        ACTIONS:
        {
         set_tcr_b (0xf3, 0x08);
         ctl_eia(0xff, 0x04);
        }
```

The equivalent softkey-generated action is LEADS T ZERO C OFF.

(B) Phase

The following routines are valid in either emulate or monitor mode.

enter_call_phase

Synopsis

```
extern void enter_call_phase();
```

Description

During the call-establishment phase, this routine overrides existing selections on the Line Setup menu with ASCII code, 7-bit odd parity, and SYNC format.

Example

When a lead changes, look for these conditions: T and R on (space), C and I off, and all leads valid. If conditions are true, enter call phase.

```
{
 extern fast_event fevar_ela_changed;
 extern const volatile unsigned short current_eia_leads;
}
LAYER: 1
     STATE: look_for_change_to_call_phase
        CONDITIONS:
        {
         fevar_ela_changed && ((current_eia_leads & 0x5dd) == 0xdd)
        }
        ACTIONS:
        {
         enter_call_phase();
        }
```

enter_data_phase

Synopsis

extern void enter_data_phase();

Description

During the data-transfer phase, this routine implements existing selections on the Line Setup menu.

Example

When a lead changes, look for these conditions: T and R off (mark), C and I on, and all leads valid. If conditions are true, enter data phase.

```
{
 extern fast_event fevar_eia_changed;
 extern const volatile unsigned short current_eia_leads;
}
LAYER: 1
     STATE: look_for_change_to_data_phase
        CONDITIONS:
        {
         fevar_eia_changed && ((current_eia_leads & 0x5dd) == 0x51d)
        }
        ACTIONS:
        {
         enter_data_phase();
        }
```

74 X.25 Layer 2 Library

When the X.25 Layer 2 package is loaded in via the Layer Setup screen, the following external routines and variables become available for use by the programmer. Their use on the Protocol Spreadsheet is not limited to any particular layer, though normally they belong at Layer 2.

The variables and routines approximate X.25 Layer 2 spreadsheet-generated conditions and actions. Refer to Section 36 for more detailed explanations of the purposes of specific conditions and actions. Sometimes the name of the variable or routine is sufficient for identifying its related spreadsheet token. When this is not the case, the information is provided below.

74.1 Structures

The structure *send_frame_structure* defines the format of transmitted X.25 frames. See Table 74-1. Use this structure to send frames via the *send_frame* routine in emulate mode. See Section 74.3(B). Each variable in the structure relates to some softkey selection or user entry in the SEND action.

74.2 Variables

(A) Monitoring Events

1. *Emulate or monitor mode.* X.25 Layer 2 events include frames detected, good or bad BCC's, and aborts. All event variables in Table 74-2 containing a *dte_* or *dce_* prefix are valid in either emulate or monitor mode. These event variables are *dte_frame, dce_frame, dte_good_bcc, dce_good_bcc, dte_bad_bcc, dce_bad_bcc, dte_abort, dce_abort.* The variable *dce_good_bcc*, for example, equates to DCE GDBCC.

   You can use both *dte* and *dce* variables relating to the same event in one conditions block. Suppose you want to count all bad BCC's from either side of the line. Enter the following CONDITIONS/ACTIONS block:

```
CONDITIONS:
{
 dte_bad_bcc || dce_bad_bcc
}
ACTIONS: COUNTER bad_bcc INC
```

Table 74-1
X.25 Layer 2 Structures

Type	Variable	Value (hex/decimal)	Meaning
Structure Name: **send_frame_structure**			Structure of a frame in X.25. Declared as type struct. Declared automatically if a softkey-entered SEND action is taken. Program frames assigned to structure as follows: struct send_frame_structure *name*. Reference a structure variable as follows: *name*.bcc_type. If values in the frame structure are not initialized by the user, they default to 0. You may initialize the values when the structure is declared: struct send_frame_structure *name* = {1, 1, 1, 0, 1, 1, 3, 0x71, 3, 0};
unsigned char	addr_type	0 1 2	command response other
unsigned char	frame_type		*(The codes for frame_type are the same as for the X.25-variable rcvd_frame_type.)*
unsigned char	nr_type	0 1 2 3	auto value received ns plus 1 last nr sent
unsigned char	ns_type	0 1 2 3	auto skip last nr received value
unsigned char	p_f_type	0 1 2	0 1 loopback
unsigned char	bcc_type	0 1 2 3	default (bad bcc) good bcc bad bcc abort
unsigned char	addr_value	1 3	to DCE to DTE
unsigned char	cntrl_byte	*(actual value of the control byte)*	
unsigned char	nr_value	*0-7 (MOD 8)*	If nr_type = 1
unsigned char	ns_value	*0-7 (MOD 8)*	If ns_type = 3

Table 74-2
X.25 Layer 2 Variables

Type	Variable	Value (hex/decimal)	Meaning
extern event	dte_frame		True when a DTE frame is detected. Line Setup configured for emulate or monitor mode.
extern event	dce_frame		True when a DCE frame is detected. Line Setup configured for emulate or monitor mode.
extern event	dte_good_bcc		True when a good BCC is calculated for a DTE frame. Line Setup configured for emulate or monitor mode.
extern event	dce_good_bcc		True when a good BCC is calculated for a DCE frame. Line Setup configured for emulate or monitor mode.
extern event	dte_bad_bcc		True when a bad BCC is calculated for a DTE frame. Line Setup configured for emulate or monitor mode.
extern event	dce_bad_bcc		True when a bad BCC is calculated for a DCE frame. Line Setup configured for emulate or monitor mode.
extern event	dte_abort		True when an abort is detected for a DTE frame. Line Setup configured for emulate or monitor mode.
extern event	dce_abort		True when an abort is detected for a DCE frame. Line Setup configured for emulate or monitor mode.
extern event	rcvd_frame		True when a frame is received. Line Setup configured for emulate mode only.
extern event	invalid_frame		True when an invalid frame is detected. Line Setup configured for emulate mode only.
extern event	l2_T1		True when the T1 timeout-timer has expired. Line Setup configured for emulate mode only.
extern event	bcc_error		True when a BCC error is detected. Line Setup configured for emulate mode only.
extern event	nr_error		True when an N(R) error is detected in a received INFO or supervisory frame. Line Setup configured for emulate mode only.
extern event	ns_error		True when an N(S) error is detected in a received INFO frame. Line Setup configured for emulate mode only.

Table 74-2 (continued)

Type	Variable	Value (hex/decimal)	Meaning
extern event	frame_sent		True when frame is passed down to Layer 1. Line Setup configured for emulate mode only.
extern volatile const unsigned char	m_frame_addr	1	to DCE
		3	to DTE
			Line Setup configured for emulate or monitor mode.
extern volatile const unsigned char	m_frame_type	*(same as rcvd_frame_type—Line Setup configured for emulate or monitor mode)*	
extern volatile const unsigned char	m_frame_cntrl_byte_1	*(actual value of control byte—Line Setup configured for emulate or monitor mode)*	
extern volatile const unsigned char	m_frame_pf	0	pf=0
		10/16	pf=1
			Line Setup configured for emulate or monitor mode.
extern volatile const unsigned char	m_frame_bcc_type	1	good
		2	bad
		3	abort
			Line Setup configured for emulate or monitor mode.
extern volatile const unsigned char	rcvd_frame_addr	1	to DCE
		3	to DTE
			Line Setup configured for emulate mode only.
extern volatile const unsigned char	rcvd_frame_type	0	info
		1	rr
		5	rnr
		9	rej
		d/13	srej
		2f/47	sabm
		6f/111	sabme
		43/67	disc
		f/15	dm
		f/15	sarm
		63/99	ua
		87/135	frmr
		ff/255	other
		ff/255	unknown
			Line Setup configured for emulate mode only.
extern volatile const unsigned char	rcvd_frame_cntrl_byte_1	*(actual value of control byte—Line Setup configured for emulate mode only)*	
extern volatile const unsigned char	rcvd_frame_pf	0	pf=0
		10/16	pf=1
			Line Setup configured for emulate mode only.
extern volatile const unsigned ohar	rcvd_frame_bcc_type	1	good
		2	bad
		3	abort
			Line Setup configured for emulate mode only.
extern volatile const unsigned char	rcvd_frame_nr	*0-7 (MOD 8)*	Line Setup configured for emulate mode only.

Table 74-2 (continued)

Type	Variable	Value (hex/decimal)	Meaning
extern volatile const unsigned char	rcvd_frame_ns	*0-7 (MOD 8)*	Line Setup configured for emulate mode only.
extern volatile unsigned short	rcvd_frame_buff_seg		Inter-layer message buffer number (actually, an IAPX-286 segment number) in a received frame. This segment number can be converted to a pointer by shifting it left 16 bits. Line Setup configured for emulate mode only.
extern volatile unsigned short	rcvd_frame_sdu_offset		Offset to where the service data unit begins in an inter-layer message buffer in a received frame. Add to buffer segment number (converted to pointer) to point to first byte in frame. Line Setup configured for emulate mode only.
extern volatile unsigned short	rcvd_frame_sdu_size		Size of service data unit in a received frame. Line Setup configured for emulate mode only.
extern volatile unsigned short	l2_current_window_edge		When equal to upper edge, window is full; when equal to lower edge, window is empty; when not equal to upper edge, window is not full; and when not equal to lower edge, window is not empty. Line Setup configured for emulate mode only.
extern volatile unsigned short	l2_lower_window_edge		see l2_current_window_edge
extern volatile unsigned short	l2_upper_window_edge		see l2_current_window_edge
extern volatile unsigned short	l2_resend_edge		When resend edge is not equal to lower window edge, there is more to resend; when resend edge is equal to lower window edge, there is no more to resend. Line Setup configured for emulate mode only.
extern unsigned char	l2_enhance	0	normal
		1	reverse
		4	low
		5	reverse low
		8	blink
		9	reverse blink
		12/18	blink low
			Line Setup configured for emulate or monitor mode.
extern unsigned char	l2_suppress	0	off
		1	on
			Line Setup configured for emulate or monitor mode.

Using spreadsheet tokens, the same test needs two CONDITIONS/ACTIONS blocks:

```
CONDITIONS: DTE BDBCC
ACTIONS: COUNTER bad_bcc INC
CONDITIONS: DCE BDBCC
ACTIONS: COUNTER bad_bcc INC
```

When the user selects DTE or DCE on the first rack of softkeys for Layer 2 conditions, a second rack appears from which he must select a particular frame type. A DTE INFO condition, for example, when translated, includes two C variables, one event variable and one status variable:

```
{
 dte_frame && (m_frame_type == 0)
}
```

As a C programmer, you do not need to specify a frame type. To include all frames in a condition, use the event variable only:

```
CONDITIONS:
{
 dte_frame
}
```

2. *Emulate mode only*. Some events may be detected in emulate mode only. The event variables are *rcvd_frame*, *invalid_frame*, *l2_T1*, *bcc_error*, *nr_error*, *ns_error*, and *frame_sent*.

   If you try to use one of these variables in monitor mode, you may be returned to the main program menu. When you go to the Protocol Spreadsheet and search for errors, a message like the following may be displayed: "*Error 140: Unresolved reference rcvd_frame.*"

   When the user selects RCV on the first rack of softkeys for Layer 2 conditions, a second rack appears from which he must select a particular frame type. When the translator converts a RCV INFO condition into C, it will include two C variables, one event variable and one status variable:

   ```
   {
    rcvd_frame && (rcvd_frame_type == 0)
   }
   ```

   The C programmer does not have to specify a frame type. To include all received frames in a condition, use the event variable only:

   ```
   CONDITIONS:
   {
    rcvd_frame
   }
   ```

   Error detecting may be accomplished via *bcc_error*, *nr_error*, *ns_error*, and *invalid_frame*. These variables equate to the softkey tokens bearing similar names.

One of the emulate-mode variables monitors an emulate action. The event variable *frame_sent* will come true as soon as the frame has been passed to the layer below. Note that if Layer 1 is an X.21 protocol in call-setup phase, a frame that is "sent" at Layer 2 will stop at Layer 1 and will not be transmitted out onto the line.

(B) Status Variables

Status variables are those in Table 74-2 that do not include *event* in the Type column. Their associated event variables guarantee that they are updated and tested.

The softkey-generated condition for received Info frames is RCV INFO. The C version of the same condition should look like this:

```
CONDITIONS:
{
 rcvd_frame && (rcvd_frame_type == 0)
}
```

1. *Frame characteristics.* All status variables in Table 74-2 containing an *m_* prefix are valid in either emulate or monitor mode: *m_frame_addr, m_frame_type, m_frame_cntrl_byte_1, m_frame_pf,* and *m_frame_bcc_type.* Use these variables to monitor a particular address, frame type, control byte, P/F value, or BCC.

   All status variables in Table 74-2 containing a *rcvd_* prefix are valid in emulate mode only: *rcvd_frame_addr, rcvd_frame_type, rcvd_frame_cntrl_byte_1, rcvd_frame_bcc_type, rcvd_frame_pf, rcvd_frame_nr,* and *rcvd_frame_ns.* Use these variables to monitor a particular address, frame type, control byte, BCC, or P/F, N(R), or N(S) value.

   If you try to use an emulate-mode variable in monitor mode, you may be returned to the main program menu. When you go to the Protocol Spreadsheet and search for errors, a message like the following may be displayed: "*Error 140: Unresolved reference rcvd_frame_type.*"

2. *Frame buffers.* As BOP frames are received, they are automatically placed in IL message buffers to be passed up the layers. Three emulate-mode variables provide the user with access to the information in the frame that is located beyond the control byte. These variables are *rcvd_frame_buff_seg, rcvd_frame_sdu_offset,* and *rcvd_frame_sdu_size.* See Section 66.1 for a more detailed discussion of the buffer components to which these variables refer.

Make a pointer to an IL buffer by casting *rcvd_frame_buff_seg* as a *long*, shifting it left sixteen bits, adding *rcvd_frame_sdu_offset*, and casting the result to a pointer. Increment the pointer twice (thereby adding two to the offset).

```
{
 unsigned char * ptr;
 ptr = (void *)(((long)rcvd_frame_buff_seg << 16) + rcvd_frame_sdu_offset);
 ptr+=2;
}
```

It is now pointing at the first byte in the X.25 Layer 3 header. You may continue to move through the frame for its entire length, indicated in *rcvd_frame_sdu_size*.

3. *Transmit window*. Four related variables test the status of the Layer 2 window. The particular values of these variables at any given time is not significant. What is significant is how they compare to each other. The softkey status condition on the left makes the variable comparison on the right:

WINDOW FULL	*l2_current_window_edge == l2_upper_window_edge*
WINDOW EMPTY	*l2_current_window_edge == l2_lower_window_edge*
WINDOW NOT_FULL	*l2_current_window_edge != l2_upper_window_edge*
WINDOW NOT_EMPTY	*l2_current_window_edge != l2_lower_window_edge*
MORE_TO_RESEND	*l2_resend_edge != l2_lower_window_edge*
NO_MORE_TO_RESEND	*l2_resend_edge == l2_lower_window_edge*

(C) Controlling Protocol Trace Display

To enhance or suppress particular frames on the Layer 2 Protocol Trace screen in emulate or monitor mode, assign a coded value to *l2_enhance* or *l2_suppress*. The values are listed in Table 74-2. To assign a value to either of these variables, place the statement in an ACTIONS block. For example, display RNR frames in reverse-video and suppress display of invalid frames:

```
CONDITIONS: RCV RNR
ACTIONS:
{
 l2_enhance = 1;
}
CONDITIONS: RCV INVALID
ACTIONS:
{
 l2_suppress = 1;
}
```

Check the value of these display-control variables in a CONDITIONS block

```
CONDITIONS: RCV INFO
{
 l2_enhance == 1
}
ACTIONS:
{
 l2_enhance = 0;
}
```

or an ACTIONS block:

```
CONDITIONS: RCV INFO
ACTIONS:
{
 if(l2_enhance == 1)
   l2_enhance = 0;
}
```

74.3 Routines

Use the following routines in emulate mode only. If you try to call one of these routines in monitor mode, you may be returned to the main program menu. When you go to the Protocol Spreadsheet and search for errors, a message like the following may be displayed: *"Error 140: Unresolved reference l2_give_data."*

(A) Receive

l2_give_data

Synopsis

```
extern void l2_give_data();
```

Description

The *l2_give_data* routine takes an interlayer message buffer associated with a received INFO frame, changes the SDU offset to point to higher-level data, and sends a DL_DATA IND primitive up to Layer 3 along with a reference to this buffer. The softkey equivalent of this routine is the GV_DATA action on the Protocol Spreadsheet.

Example

Layer 3 wants access to the line in order to receive and send data. Assuming X.25 personality packages are loaded at Layers 2 and 3, enter the following program:

```
LAYER: 2
    STATE: datalink
       CONDITIONS: DL_CONNECT REQ
       ACTIONS: DL_CONNECT CONF
       CONDITIONS: DL_DATA REQ
       ACTIONS: SEND INFO "((DL_DATA))"
       CONDITIONS: RCV INFO
       ACTIONS:
       {
        l2_give_data();
       }
```

(B) Transmit

resend_frame

Synopsis

```
extern void resend_frame(pf, first_or_next);
unsigned char pf;
unsigned char first_or_next;
```

Description

The *resend_frame* routine will set the P/F bit to a specified value and resend either the first or next frame in the window. The softkey equivalent of this routine is the (PROTOCL) RESEND action on the Protocol Spreadsheet.

Inputs

The first parameter is the value of the P/F bit in the frame. It may be set to either 0 or 1.

The second parameter indicates whether the first frame in the window will be sent, or whether the next frame in the window will be sent. The first resend action will send the first frame in the window regardless of whether first or, next has been selected. Legal entries are 0 (first) or 1 (next).

Example

Suppose you want to resend the entire transmit window if you receive a REJ frame.

```
LAYER: 2
    STATE: xfer
       /* Whatever conditions and actions send data precede the following condition. */

       CONDITIONS: RCV REJ RESP
       NEXT_STATE: recover
```

```
     STATE: recover
        CONDITIONS: ENTER_STATE
        ACTIONS:
        {
         resend_frame(1, 0);
        }
        CONDITIONS: FRAME_SENT
             MORE_TO_RESEND
        ACTIONS:
        {
         resend_frame(1,1);
        }
        CONDITIONS: FRAME_SENT
             NO_MORE_TO_RESEND
        NEXT_STATE: xfer
```

reset_nr

Synopsis

extern void reset_nr();

Description

This routine resets the N(R) field in information and supervisory frames to zero. The softkey equivalent of this routine is the (PROTOCL) RSET_NR action on the Protocol Spreadsheet.

Example

When a link is established, reset N(R).

```
LAYER: 2
     STATE: reset
        CONDITIONS: ENTER_STATE
        ACTIONS: SEND SABM
        CONDITIONS: RCV UA
        ACTIONS:
        {
         reset_nr();
        }
```

reset_ns

Synopsis

extern void reset_ns();

Description

The N(S) field in information frames is reset to zero and the transmit window is cleared. The softkey equivalent of this routine is the (PROTOCL) RSET_NS action on the Protocol Spreadsheet.

Example

When a link is established, reset N(S).

```
LAYER: 2
    STATE: reset
        CONDITIONS: ENTER_STATE
        ACTIONS: SEND SABM
        CONDITIONS: RCV UA
        ACTIONS:
        {
         reset_ns();
        }
```

send_frame

Synopsis

```
extern void send_frame(il_buffer_number, relay_baton, data_start_offset, transmit_frame_ptr);
unsigned short il_buffer_number;
unsigned short relay_baton;
unsigned short data_start_offset;
struct send_frame_structure
{
 unsigned char addr_type;
 unsigned char frame_type;
 unsigned char nr_type;
 unsigned char ns_type;
 unsigned char p_f_type;
 unsigned char bcc_type;
 unsigned char addr_value;
 unsigned char cntrl_byte;
 unsigned char nr_value;
 unsigned char ns_value;
};
struct send_frame_structure * transmit_frame_ptr;
```

Description

The *send_frame* routine adds a frame-level header to an interlayer message buffer and passes the buffer to Layer 1. The softkey equivalent of this routine is the SEND action on the Protocol Spreadsheet.

Inputs

The first parameter is the interlayer message buffer number. See Section 66.3(A), Layer-Independent OSI routines.

The second parameter is the maintain bit used to hold the buffer while the send operation is being performed. See Section 66.3(A).

The third parameter is the offset from the beginning of the buffer to the start of the service data unit. See Section 66.3(A).

The fourth parameter is a pointer to the frame structure to be sent. For a description of *send_frame_structure*, see Table 74-1.

Example

Send an Info frame containing a canned fox message and a good BCC onto the line.

```
{
 static unsigned short il_buffer_number;
 static unsigned short relay_baton;
 static unsigned short data_start_offset;
 struct send_frame_structure
  {
   unsigned char addr_type;
   unsigned char frame_type;
   unsigned char nr_type;
   unsigned char ns_type;
   unsigned char p_f_type;
   unsigned char bcc_type;
   unsigned char addr_value;
   unsigned char cntrl_byte;
   unsigned char nr_value;
   unsigned char ns_value;
  };
 struct send_frame_structure transmit_frame;
 static char transmit_string [] = "((FOX))";
}
LAYER: 2
     STATE; send_a_frame
        CONDITIONS: KEYBOARD " "
        ACTIONS:
        {
         _get_il_msg_buff(&il_buffer_number, &relay_baton);
         _start_il_buff_list(il_buffer_number,&data_start_offset);
         transmit_frame.bcc_type  = 1;
         _insert_il_buff_list_cnt(il_buffer_number, data_start_offset, &transmit_string[0],
              (sizeof(transmit_string) - 1));
         send_frame(il_buffer_number, relay_baton, data_start_offset, &transmit_frame);
        }
```

75 X.25 Layer 3 Library

When the X.25 Layer 3 package is loaded in via the Layer Setup screen, the following external routines and variables become available for use by the programmer. Their use on the Protocol Spreadsheet is not limited to any particular layer, though normally they belong at Layer 3.

The variables and routines approximate X.25 Layer 3 spreadsheet-generated conditions and actions. Refer to Section 37 for more detailed explanations of the purposes of specific conditions and actions. Sometimes the name of the variable or routine is sufficient for identifying its related spreadsheet token. When this is not the case, the information is provided below.

75.1 Structures

The *send_packet_structure* defines the format of transmitted X.25 packets. See Table 75-1. Use this structure to send packets via the *send_packet* routine in emulate mode. See Section 75.3(B). Each variable in the structure relates to some softkey selection or user entry in the SEND action.

75.2 Variables

(A) Monitoring Events

1. *Emulate or monitor mode.* Two X.25 Layer 3 event variables are valid in either emulate or monitor mode. These event variables are *dte_packet* and *dce_packet.*

   When the user selects DTE or DCE on the first rack of softkeys for Layer 3 conditions, a second rack appears from which he must select a particular packet type. A DTE DATA condition, for example, when translated, includes two C variables, one event variable and one status variable:

```
{
 dte_packet && (m_packet_type == 0)
}
```

Table 75-1
X.25 Layer 3 Structures

Type	Variable	Value (hex/decimal)	Meaning
Structure Name:	**send_packet_structure**		Structure of a packet in X.25. Declared as type struct. Declared automatically if a softkey-entered SEND action is taken. Program packets assigned to structure as follows: struct send_packet_structure *name*. Reference a structure variable as follows: *name*.q_bit. If values in the frame structure are not initialized by the user, they default to 0. You may initialize the values when the structure is declared: struct send_packet_structure *name* = {2, 0x13, 0x13, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 2, &*facilities_string*[0], 0, 0};
unsigned char	path_num	*0-8* fe/254	path number use path number of last received packet
unsigned char	packet_type	*(The codes for packet_type are the same as for the X.25-variable m_packet_type.)*	
unsigned char	packet_type_byte	*(actual value of the packet type byte)*	

unsigned char	m_bit	0
		1
			m = 0
			m = 1

unsigned char	d_bit	0
		40/64
			d = 0
			d = 1

unsigned char	q_bit	0
		80/128
			q = 0
			q = 1

unsigned char	pr_type	0 1 2 3	auto value received ps plus 1 last pr sent
unsigned char	ps_type	0 1 2 3	auto skip received pr value
unsigned char	pr_value	*0-7 (MOD 8)*	if pr_type = 1
unsigned char	ps_value	*0-7 (MOD 8)*	if ps_type = 3
unsigned char	cause	*(value of cause byte—see Figure 36-15)*	
unsigned char	diag_flag	0 1	diagnostic field not present diagnostic field is present
unsigned char	diag	*(value of diagnostic byte—consult CCITT Recommendation X.25, pp. 237-8)*	
unsigned char	spare	0	reserved space
unsigned char	facilities_len	*0-ff/0-255*	length of the facilities field
char *	facilities		pointer to the location of the facilities field—the facilities field is declared separately
unsigned short	data_len		reserved for future use
char *	data		reserved for future use

Table 75-2
X.25 Layer 3 Variables

Type	Variable	Value (hex/decimal)	Meaning
extern event	dte_packet		True when a DTE packet is detected. Line Setup configured for emulate or monitor mode.
extern event	dce_packet		True when a DCE packet is detected. Line Setup configured for emulate or monitor mode.
extern event	rcvd_packet		True when a packet is received from Layer 2. Line Setup configured for emulate mode only.
extern event	invalid_packet		True when an invalid packet is detected. Line Setup configured for emulate mode only.
extern event	pr_error		True when an P(R) error is detected in a data or supervisory packet. Line Setup configured for emulate mode only.
extern event	ps_error		True when an P(S) error is detected in a data packet. Line Setup configured for emulate mode only.
extern event	packet_sent		True when a packet has been passed down to Layer 2. Line Setup configured for emulate mode only.
extern volatile unsigned short	m_packet_lcn	*0-fff/0-4095*	Logical channel number. Line Setup configured for emulate or monitor mode.
extern volatile unsigned char	m_packet_lcn_grp	*0-f/0-15*	Logical channel group number. Line Setup configured for emulate or monitor mode.

extern volatile const unsigned char	m_packet_q	0
		80/128
			q=0
			q=1
			Line Setup configured for emulate or monitor mode.

extern volatile const unsigned char	m_packet_d	0
		40/64
			d=0
			d=1
			Line Setup configured for emulate or monitor mode.

extern volatile const unsigned char	m_packet_m	0
		10/16
			m=0
			m=1
			Line Setup configured for emulate or monitor mode.

extern volatile const unsigned char	m_packet_pr	*0-7 (MOD 8)*	Line Setup configured for emulate or monitor mode.
extern volatile const unsigned char	m_packet_ps	*0-7 (MOD 8)*	Line Setup configured for emulate or monitor mode.

Table 75-2 (continued)

Type	Variable	Value (hex/decimal)	Meaning
extern volatile const unsigned char	m_packet_cause	*(same as rcvd_pkt_cause—Line Setup configured for emulate or monitor mode)*	
extern volatile const unsigned char	m_packet_diag_code	*(same as rcvd_pkt_diagn—Line Setup configured for emulate or monitor mode)*	
extern volatile const unsigned char	m_packet_type_byte	*(actual value of packet type byte—Line Setup configured for emulate or monitor mode)*	
extern volatile const unsigned char	m_packet_type	b/11	call
		f/15	call acc
		13/19	clear
		17/23	clear conf
		0	data
		23/35	int
		27/39	int conf
		1	rr
		5	rnr
		9	rej
		1b/27	reset
		1f/31	reset conf
		fb/251	restart
		ff/255	restart conf
		f1/241	diag
		f3/243	reg
		f7/247	reg conf
		11/17	other pkt
		11/17	unknown pkt
			Line Setup configured for emulate or monitor mode.
extern volatile unsigned short	rcvd_pkt_lcn	*0-ffff/0-4095*	Logical channel number in a received packet. Line Setup configured for emulate mode only.
extern volatile const unsigned char	rcvd_pkt_q	0	q=0
		80/128	q=1
			Line Setup configured for emulate mode only.
extern volatile const unsigned char	rcvd_pkt_d	0	d=0
		40/64	d=1
			Line Setup configured for emulate mode only.
extern volatile const unsigned char	rcvd_pkt_m	0	m=0
		10/16	m=1
			Line Setup configured for emulate mode only.
extern volatile const unsigned char	rcvd_pkt_pr	*0-7 (MOD 8)*	Line Setup configured for emulate mode only.
extern volatile const unsigned char	rcvd_pkt_ps	*0-7 (MOD 8)*	Line Setup configured for emulate mode only.
extern volatile const unsigned char	rcvd_pkt_cause	*(see Figure 36-15—Line Setup configured for emulate mode only)*	
extern volatile const unsigned char	rcvd_pkt_diagn	*(consult CCITT Recommendation X.25, pp.237-8—Line Setup configured for emulate mode only)*	

Table 75-2 (continued)

Type	Variable	Value (hex/decimal)	Meaning
extern volatile const unsigned char	rcvd_pkt_type_byte	*(actual value of packet type byte—Line Setup configured for emulate mode only)*	
extern volatile const unsigned char	rcvd_packet_type	b/11 f/15 13/19 17/23 0 23/35 27/39 1 5 9 1b/27 1f/31 fb/251 ff/255 f1/241 f3/243 f7/247 11/17 11/17	call call acc clear clear conf data int int conf rr rnr rej reset reset conf restart restart conf diag reg reg conf other pkt unknown pkt Line Setup configured for emulate mode only.
extern volatile unsigned short	m_packet_buff_seg		Inter-layer message buffer number (actually, an iAPX-286 segment number). This segment number can be converted to a pointer by shifting it left 16 bits. Line Setup configured for emulate or monitor mode.
extern volatile unsigned short	m_packet_info_seg		Same as *m_packet_buff_seg*.
extern volatile unsigned short	m_packet_sdu_offset		Offset to where the service data unit begins in an inter-layer message buffer. Add to *m_pkt_buff_seg* (converted to pointer) to point to first packet-header byte. Line Setup configured for emulate or monitor mode.
extern volatile unsigned short	m_packet_info_offset		Offset to where the packet information begins, excluding the header. Add to *m_pkt_buff_seg* (converted to pointer) to point to packet data. Line Setup configured for emulate or monitor mode.
extern volatile unsigned short	m_packet_length		Length of the packet, including header. Line Setup configured for emulate or monitor mode.
extern volatile unsigned short	m_packet_info_length		Length of the packet information, excluding the header. Line Setup configured for emulate or monitor mode.

Table 75-2 (continued)

Type	Variable	Value (hex/decimal)	Meaning
extern volatile unsigned short	rcvd_pkt_buff_seg		Inter-layer message buffer number (actually, an IAPX-286 segment number) in a received packet. This segment number can be converted to a pointer by shifting it left 16 bits. Line Setup configured for emulate mode only.
extern volatile unsigned short	rcvd_pkt_info_seg		Same as *rcvd_pkt_buff_seg*.
extern unsigned short	rcvd_pkt_sdu_offset		Offset to where the service data unit begins in an inter-layer message buffer in a packet received. Add to *rcvd_pkt_buff_seg* (converted to pointer) to point to first packet-header byte. Line Setup configured for emulate mode only.
extern volatile unsigned short	rcvd_pkt_info_offset		Offset to where the packet information begins, excluding the header. Add to *rcvd_pkt_buff_seg* (converted to pointer) to point to packet data. Line Setup configured for emulate mode only.
extern unsigned short	rcvd_pkt_length		Length of a received packet, including header information. Line Setup configured for emulate mode only.
extern volatile unsigned short	rcvd_pkt_info_length		Length of the information in a received packet, excluding the header. Line Setup configured for emulate mode only.
extern volatile unsigned char *	m_packet_ptr		Pointer to the packet, beginning at the first byte in the header. Line Setup configured for emulate or monitor mode.
extern volatile unsigned char *	m_packet_info_ptr		Pointer to the information in a packet. Initially points to the byte immediately following the packet-type byte. Line Setup configured for emulate or monitor mode.
extern volatile unsigned char *	rcvd_packet_ptr		Pointer to the packet, beginning at the first byte in the header. Line Setup configured for emulate mode only.
extern volatile unsigned char *	rcvd_pkt_info_ptr		Pointer to the packet information, initially located at the byte immediately following the packet header. Line Setup configured for emulate mode only.

Table 75-2 (continued)

Type	Variable	Value (hex/decimal)	Meaning
extern volatile const unsigned char	rcvd_device_path		Path number connecting received packet to particular LCN and particular set of call parameters on the X.25 Packet Level Setup screen. Line Setup configured for emulate mode only.
extern unsigned char	l3_enhance	0 1 4 5 8 9 12/18	normal reverse low reverse low blink reverse blink blink low Line Setup configured for emulate or monitor mode.
extern unsigned char	l3_suppress	0 1	off on Line Setup configured for emulate or monitor mode.

A C programmer does not have to specify a packet type. To include all packets in a condition, use the event variable only:

```
CONDITIONS:
{
 dte_packet
}
```

2. *Emulate mode only*. Some events may be detected in emulate mode only. These are *rcvd_packet*, *invalid_packet*, *pr_error*, *ps_error*, and *packet_sent*.

If you try to use one of these variables in monitor mode, you may be returned to the main program menu. When you go to the Protocol Spreadsheet and search for errors, a message like the following may be displayed: "*Error 140: Unresolved reference rcvd_packet.*"

When the user selects RCV on the first rack of softkeys for Layer 3 conditions, a second rack appears from which he must select a particular packet type. When the translator converts a RCV DATA condition into C, it will include two C variables, one event variable and one status variable:

```
{
 rcvd_packet && (rcvd_packet_type == 0)
}
```

As a C programmer, you do not have to specify a packet type. To include all received packets in a condition, use the event variable only:

```
CONDITIONS:
{
 rcvd_packet
}
```

Error detecting may be accomplished via *pr_error*, *ps_error*, and *invalid_packet*. These variables equate to the softkey tokens bearing similar names.

One of the emulate-mode variables monitors an emulate action. "SEND"ing a packet means queuing a packet to be passed down to Layer 2. If the Layer 2 link is not established, for example, the packet will be held at Layer 3 pending link establishment. The event variable *packet_sent* will not come true until the packet actually has been passed to the layer below. Use this condition to start accurate response-time measurements at the packet level rather than at the line level.

(B) Status Variables

Status variables are those in Table 75-2 that do not include *event* in the Type column. Their associated event variables guarantee that they are updated and tested.

The softkey-generated condition for received Data packets is RCV DATA. The C version of the same condition should look like this:

```
CONDITIONS:
{
rcvd_packet && (rcvd_packet_type == 0)
}
```

1. *Packet characteristics*. All status variables in Table 75-2 containing an *m_* prefix are valid in either emulate or monitor mode: *m_packet_lcn*, *m_packet_lcn_grp*, *m_packet_q*, *m_packet_d*, *m_packet_m*, *m_packet_pr*, *m_packet_ps*, *m_packet_cause*, *m_packet_diag_code*, *m_packet_type*, and *m_packet_type_byte*.

   All status variables in Table 75-2 containing a *rcvd_* prefix are valid in emulate mode only: *rcvd_pkt_lcn*, *rcvd_pkt_q*, *rcvd_pkt_d*, *rcvd_pkt_m*, *rcvd_pkt_pr*, *rcvd_pkt_ps*, *rcvd_pkt_cause*, *rcvd_pkt_diagn*, *rcvd_pkt_type*, and *rcvd_pkt_type_byte*.

   If you try to use an emulate-mode variable in monitor mode, you may be returned to the main program menu. When you go to the Protocol Spreadsheet and search for errors, a message like the following may be displayed: "*Error 140: Unresolved reference rcvd_packet_type.*"

2. *Packet buffers*. Packets are passed up to Layer 3 from Layer 2 in IL message buffers. Several variables provide the user with access to the information in the packet that is located beyond the packet-type byte. These variables are *rcvd_pkt_buff_seg*, *m_packet_buff_seg*, *rcvd_pkt_sdu_offset*, *m_packet_sdu_offset*, *rcvd_pkt_length*, and *m_packet_length*. See Section 66.1 for a more detailed discussion of the buffer components to which these variables refer.

3. *Pointers*. Two variables, *rcvd_pkt_info_ptr* and *m_packet_info_ptr*, point to the first byte beyond the packet header. You may move these pointers to

access data throughout the length of the packet. The length is indicated by *rcvd_pkt_info_length* (or *m_packet_info_length*).

4. *Path.* An IL buffer that is sent down the layers or received up the layers is provided with a "path" number that ties it, at X.25 Layer 3, to a particular LCN as well as to a particular set of Call Request parameters on the X.25 Packet Level Setup screen.

   When a call request is sent or received by the INTERVIEW, the call parameters are correlated to the Packet Level Setup screen. If the INTERVIEW sends a call request that specifies a path number, or if the INTERVIEW receives a call request that matches one of the path entries on the setup screen, the LCN of the call request is tied to the path number (path #3, for example), and any subsequent packets with the same LCN will satisfy *rcvd_device_path == 3* conditions.

(C) Controlling Protocol Trace Display

To enhance or suppress particular packets on the Layer 3 Protocol Trace screen in emulate or monitor mode, assign a coded value to *l3_enhance* or *l3_suppress*. The values are listed in Table 75-2. To assign a value to either of these variables, place the statement in an ACTIONS block. For example, display RNR packets in reverse-video and suppress display of invalid packets:

```
CONDITIONS: RCV RNR
ACTIONS:
{
 l3_enhance = 1;
}
CONDITIONS: RCV INVALID
ACTIONS:
{
 l3_suppress = 1;
}
```

Check the value of these display-control variables in a CONDITIONS block

```
CONDITIONS: RCV DATA
{
 l3_enhance == 1
}
ACTIONS:
{
 l3_enhance = 0;
}
```

or an ACTIONS block:

```
CONDITIONS: RCV DATA
ACTIONS:
{
 if(l3_enhance == 1)
   l3_enhance = 0;
}
```

75.3 Routines

Use the following routines in emulate mode only. If you try to call one of these routines in monitor mode, you may be returned to the main program menu. When you go to the Protocol Spreadsheet and search for errors, a message like the following may be displayed: *"Error 140: Unresolved reference l3_give_data."*

(A) Receive

l3_give_data

Synopsis

```
extern void l3_give_data();
```

Description

The *l3_give_data* routine takes an interlayer message buffer associated with a received data packet, changes the SDU offset to point to higher-level data, and sends an N_DATA IND primitive up to Layer 4 along with a reference to this buffer. The softkey equivalent of this routine is the GV_DATA action on the Protocol Spreadsheet.

Example

Layer 4 wants access to the line in order to receive and send data. Assuming X.25 personality packages are loaded at Layers 2 and 3, enter the following program:

```
LAYER: 2
    STATE: datalink
       CONDITIONS: DL_CONNECT REQ
       ACTIONS: DL_CONNECT CONF
       CONDITIONS: DL_DATA REQ
       ACTIONS: SEND INFO "((DL_DATA))"
       CONDITIONS: RCV INFO
       ACTIONS: GIVE_DATA
LAYER: 3
    STATE: pass_data_up
       CONDITIONS: N_CONNECT REQ
       ACTIONS: SEND CALL
       CONDITIONS: RCV CALL_CONF
       ACTIONS: N_CONNECT IND
       CONDITIONS: N_DATA REQ
       ACTIONS: SEND DATA "((N_DATA))"
       CONDITIONS: RCV DATA
       ACTIONS:
       {
       l3_give_data();
       }
LAYER: 4
    STATE: establish_link
       CONDITIONS: ENTER_STATE
       ACTIONS: N_CONNECT REQ
```

(B) Transmit

l3_clear_path

Synopsis

```
extern void l3_clear_path(path_number);
unsigned char path_number;
```

Description

The *l3_clear_path* routine resets P(R)- and P(S)-related variables, clears the transmit window, and resets the LCN and address fields to void (unless permanently assigned on the Layer 3 X.25 Packet Level Setup screen) on a designated path.

Inputs

The only parameter is the path number which is to be cleared. The value may be 0 - 8, or 0xfe if you want the path number to be that of the last received packet.

Example

When a Clear packet is received, clear the path.

```
LAYER: 3
    STATE: clearing
        CONDITIONS: RCV CLEAR
        ACTIONS: SEND CLEAR_CONF
        {
         l3_clear_path(0xfe);
        }
```

l3_more_to_resend

Synopsis

```
extern unsigned char l3_more_to_resend(path_number);
unsigned char path_number;
```

Description

The *l3_more_to_resend* routine determines whether or not there are any more packets in the transmit window to resend. It is used in combination with a transitional condition such as *packet_sent* as a condition on the Protocol Spreadsheet. The softkey equivalent is PACKET_SENT MORE_TO_RESEND or PACKET_SENT NO_MORE_TO_RESEND.

Inputs

The only parameter is the path number associated with the transmit window. The value may be 0 - 8, or 0xfe if you want the path number to be that of the last received packet.

Returns

If there is more to resend, the returned value is non-zero. If there is no more to resend, or if the given path is invalid, the returned value is 0.

Example

In this example, the entire transmit window will be resent.

```
{
 extern event packet_sent;
}
LAYER: 3
     STATE: xfer
        /* Whatever conditions and actions send data precede the following condition. */

        CONDITIONS: RCV REJ
        ACTIONS: RESEND_FIRST
        NEXT_STATE: recover
     STATE: recover
        CONDITIONS: ENTER_STATE
        {
         packet_sent &&(l3_more_to_resend(0xfe) != 0)
        }
        ACTIONS: RESEND_NEXT
        CONDITIONS:
        {
         packet_sent &&(l3_more_to_resend(0xfe) == 0)
        }
        NEXT_STATE: xfer
```

l3_window_full

Synopsis

```
extern unsigned char l3_window_full(path_number);
unsigned char path_number;
```

Description

This routine determines whether the Layer 3 window for a specified path is full or not full. When the window is full, no additional packets will be buffered until some acknowledgment is received. It is used in combination with a transitional condition such as *receive_packet* as a condition on the Protocol Spreadsheet. The softkey equivalent is RCV RR (PROTOCL) WINDOW NOT_FULL or RCV RR (PROTOCL) WINDOW FULL.

Inputs

The only parameter is the path number whose window is to be checked. The value may be 0 – 8, or 0xfe if you want the path number to be that of the last received packet.

Returns

If the window is full, or if the given path is invalid, the returned value is non-zero. If the window is not full, the returned value is 0.

Example

Transmit data packets until the transmit window is full.

```
{
 extern event packet_sent;
}
LAYER: 3
     STATE: check_window
        CONDITIONS:
        {
         packet_sent && (l3_window_full(0xfe) != 0)
        }
        ACTIONS: SEND DATA "((FOX))"
```

l3_window_empty

Synopsis

```
extern unsigned char l3_window_empty(path_number);
unsigned char path_number;
```

Description

This routine determines whether the Layer 3 window for a specified path is empty or not empty. It is used in combination with a transitional condition such as *receive_packet* as a condition on the Protocol Spreadsheet. The softkey equivalent is RCV RR (PROTOCL) WINDOW NOT_EMPTY or RCV RR (PROTOCL) WINDOW EMPTY.

Inputs

The only parameter is the path number whose window is to be checked. The value may be 0 – 8, or 0xfe if you want the path number to be that of the last received packet.

Returns

If the window is empty, or if the given path is invalid, the returned value is non-zero. If the window is not empty, the returned value is 0.

Example

If a timeout expires and the transmit window is not empty, resend the first packet in the window.

```
{
 extern event timeout_ack_expired;
 extern event rcvd_packet;
}
LAYER: 3
     STATE: check_window
        CONDITIONS: PACKET_SENT
        ACTIONS: TIMEOUT ack RESTART
        CONDITIONS:
        {
         rcvd_packet
        }
        ACTIONS: TIMEOUT ack STOP
        CONDITIONS:
        {
         timeout_ack_expired && (l3_window_empty(0xfe) != 0)
        }
        ACTIONS: RESEND FIRST
```

resend_packet

Synopsis

```
extern void resend_packet(path_number, first_or_next);
unsigned char path_number;
unsigned char first_or_next;
```

Description

The *resend_packet* routine will resend either the first or next packet in the window along a specified path. The softkey equivalent of this routine is the RESEND action on the Protocol Spreadsheet.

Inputs

The first parameter is the value of the path on which to resend the packet. It may be 0 - 8, or 0xfe for the path of the last received packet.

The second parameter indicates whether the first packet in the window will be sent, or whether the next packet in the window will be sent. The first resend action will send the first packet in the window regardless of whether first or next has been selected. Legal entries are 0 (first) or 1 (next).

Example

Suppose you want to resend the entire transmit window if you receive a REJ packet. In this example, it's being sent along the path of the last received packet.

```
LAYER: 3
    STATE: xfer
      /* Whatever conditions and actions send data precede the following condition. */

      CONDITIONS: RCV REJ
      NEXT_STATE: recover
    STATE: recover
      CONDITIONS: ENTER_STATE
      ACTIONS:
      {
       resend_packet(0xfe, 0);
      }
      CONDITIONS: PACKET_SENT
          MORE_TO_RESEND
      ACTIONS:
      {
       resend_packet(0xfe,1);
      }
      CONDITIONS: PACKET_SENT
          NO_MORE_TO_RESEND
      NEXT_STATE: xfer
```

reset_pr_ps

Synopsis

```
extern void reset_pr_ps(path_number);
unsigned char path_number;
```

Description

The P(R) and P(S) fields in data and supervisory packets are reset to zero. The transmit window is also cleared. The softkey equivalent of this routine is the (PROTOCL) RSTPRPS action on the Protocol Spreadsheet.

Inputs

The only parameter is the path number on which P(R) and P(S) are to be reset. The value may be 0 – 8, or 0xfe if you want the path number to be that of the last received packet.

Example

In this example, P(R) and P(S) are reset on path 2 whenever a Reset packet is received.

```
LAYER: 3
    STATE: reset
      CONDITIONS: RCV RESET
      ACTIONS:
      {
       reset_pr_ps(2);
      }
```

send_packet

Synopsis

```
extern void send_packet(il_buffer_number, relay_baton, data_start_offset,
     transmit_packet_ptr);
unsigned short il_buffer_number;
unsigned short relay_baton;
unsigned short data_start_offset;

struct send_packet_structure
{
 unsigned char path_num;
 unsigned char packet_type;
 unsigned char packet_type_byte;
 unsigned char m_bit;
 unsigned char d_bit;
 unsigned char q_bit;
 unsigned char pr_type;
 unsigned char ps_type;
 unsigned char pr_value;
 unsigned char ps_value;
 unsigned char cause;
 unsigned char diag_flag
 unsigned char diag;
 unsigned char cntrl_byte;
 unsigned char facilities_len;
 char * facilities;
 unsigned short data_len;
 char * data;
};
struct send_packet_structure * transmit_packet_ptr;
```

Description

The *send_packet* routine adds a packet-level header to an interlayer message buffer and passes the buffer to Layer 2. The softkey equivalent of this routine is the SEND action on the Protocol Spreadsheet.

Inputs

The first parameter is the interlayer message buffer number. See Section 66.3(A), Layer-Independent OSI routines.

The second parameter is the maintain bit used to hold the buffer while the send operation is being performed. See See Section 66.3(A).

The third parameter is the offset from the beginning of the buffer to the start of the service data unit. See See Section 66.3(A).

The fourth parameter is a pointer to the packet structure to be sent. For a description of *send_packet_structure* see Table 75-1.

Example

To successfully send a packet out to the line, you must include the Layer 2 section of the program below. In this example, you are sending a Call Request packet with a facilities field present.

```
{
 static unsigned short il_buffer_number;
 static unsigned short relay_baton;
 static unsigned short data_start_offset;

 struct send_packet_structure
  {
   unsigned char path_num;
   unsigned char packet_type;
   unsigned char packet_type_byte;
   unsigned char m_bit;
   unsigned char d_bit;
   unsigned char q_bit;
   unsigned char pr_type;
   unsigned char ps_type;
   unsigned char pr_value;
   unsigned char ps_value;
   unsigned char cause;
   unsigned char diag_flag
   unsigned char diag;
   unsigned char cntrl_byte;
   unsigned char facilities_len;
   char * facilities;
   unsigned short data_len;
   char * data;
  };
 static char transmit_string [] = "((FOX))";
 static char facilities_string [] = "01010401414503430707";
 struct send_packet_structure transmit_packet = {0, 0x13, 0x13, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
      (sizeof(facilities_string)-1), &facilities_string[0], 0, 0};
}
LAYER: 2
     STATE: datalink
        CONDITIONS: DL_CONNECT REQ
        ACTIONS: DL_CONNECT CONF
        CONDITIONS: DL_DATA REQ
        ACTIONS: SEND INFO "((DL_DATA))"
        CONDITIONS: RCV INFO
        ACTIONS: GIVE_DATA
LAYER: 3
     STATE: send_a_packet
        CONDITIONS: KEYBOARD " "
        ACTIONS:
        {
         _get_il_msg_buff(&il_buffer_number, &relay_baton);
         _start_il_buff_list(il_buffer_number,&data_start_offset);
         _insert_il_buff_list_cnt(il_buffer_number, data_start_offset, &transmit_string[0],
              (sizeof(transmit_string) - 1));
         send_packet(il_buffer_number, relay_baton, data_start_offset, &transmit_packet);
        }
```

NOTE: A null is appended to the end of an array initialized as a string inside quotation marks; it is not appended to the end of an array entered inside curly braces. So, if *facilities_string* was initialized as a list of values, like this—

```
static char facilities_string [] = {1, 1, 4, 1, 0x41, 0x45, 0x03, 0x43, 7, 7};
```

—then *transmit_packet* would look like this—

```
struct send_packet_structure transmit_packet = {0, 0x13, 0x13, 0, 0, 0, 0, 0, 0, 0, 1,
    1, 0, 0, sizeof(facilities_string), &facilities_string[0], 0, 0};
```

76 SDLC Library

When the SDLC package is loaded in via the Layer Setup screen, the following external routines and variables become available for use by the programmer. Their use on the Protocol Spreadsheet is not limited to any particular layer, though normally they belong at Layer 2.

The variables and routines approximate SDLC Layer 2 spreadsheet-generated conditions and actions. Refer to Section 38 for more detailed explanations of the purposes of specific conditions and actions. Sometimes the name of the variable or routine is sufficient for identifying its related spreadsheet token. When this is not the case, the information is provided below.

76.1 Structures

The structure *send_frame_structure* defines the format of transmitted SDLC frames. See Table 76-1. Use this structure to send frames via the *send_frame* routine in emulate mode. See Section 76.3(B). Each variable in the structure relates to some softkey selection or user entry in the SEND action.

76.2 Variables

(A) Monitoring Events

1. *Emulate or monitor mode*. SDLC events include frames detected, good or bad BCC's, and aborts. All event variables in Table 76-2 containing a *dte_* or *dce_* prefix are valid in either emulate or monitor mode. These event variables are *dte_frame, dce_frame, dte_good_bcc, dce_good_bcc, dte_bad_bcc, dce_bad_bcc, dte_abort, dce_abort.* The variable *dce_good_bcc*, for example, equates to DCE GDBCC.

   You can use both *dte* and *dce* variables relating to the same event in one conditions block. Suppose you want to count all bad BCC's from either side of the line. Enter the following CONDITIONS/ACTIONS block:

   ```
   CONDITIONS:
   {
     dte_bad_bcc || dce_bad_bcc
   }
   ACTIONS: COUNTER bad_bcc INC
   ```

Table 76-1
SDLC Structures

Type	Variable	Value (hex/decimal)	Meaning
Structure Name:	**send_frame_structure**		Structure of a frame in SDLC. Declared as type struct. Declared automatically if a softkey-entered SEND action is taken. Program frames assigned to structure as follows: struct send_frame_structure *name*. Reference a structure variable as follows: *name*.bcc_type. If values in the frame structure are not initialized by the user, they default to 0. You may initialize the values when the structure is declared: struct send_frame_structure *name* = {2, 1, 1, 0, 1, 1, 3, 0x71, 3, 0};
unsigned char	addr_type	2 3	address is specified in *addr_value* variable below loopback
unsigned char	frame_type	*(The codes for frame_type are the same as for the SDLC-variable rcvd_frame_type.)*	
unsigned char	nr_type	0 1 2 3	auto value received ns plus 1 last nr sent
unsigned char	ns_type	0 1 2 3	auto skip last nr received value
unsigned char	p_f_type	0 1 2	0 1 loopback
unsigned char	bcc_type	0 1 2 3	default (bad bcc) good bcc bad bcc abort
unsigned char	addr_value	*00-ff/0-255*	
unsigned char	cntrl_byte	*(actual value of the control byte)*	
unsigned char	nr_value	*0-7 (MOD 8)*	if nr_type = 1
unsigned char	ns_value	*0-7 (MOD 8)*	if ns_type = 3

Table 76-2
SDLC Variables

Type	Variable	Value (hex/decimal)	Meaning
extern event	dte_frame		True when a DTE frame is detected. Line Setup configured for emulate or monitor mode.
extern event	dce_frame		True when a DCE frame is detected. Line Setup configured for emulate or monitor mode.
extern event	dte_good_bcc		True when a good BCC is calculated for a DTE frame. Line Setup configured for emulate or monitor mode.
extern event	dce_good_bcc		True when a good BCC is calculated for a DCE frame. Line Setup configured for emulate or monitor mode.
extern event	dte_bad_bcc		True when a bad BCC is calculated for a DTE frame. Line Setup configured for emulate or monitor mode.
extern event	dce_bad_bcc		True when a bad BCC is calculated for a DCE frame. Line Setup configured for emulate or monitor mode.
extern event	dte_abort		True when an abort is detected for a DTE frame. Line Setup configured for emulate or monitor mode.
extern event	dce_abort		True when an abort is detected for a DCE frame. Line Setup configured for emulate or monitor mode.
extern event	rcvd_frame		True when a frame is received. Line Setup configured for emulate mode only.
extern event	invalid_frame		True when an invalid frame is detected. Line Setup configured for emulate mode only.
extern event	l2_T1		True when the T1 timeout-timer has expired. Line Setup configured for emulate mode only.
extern event	bcc_error		True when a BCC error is detected. Line Setup configured for emulate mode only.
extern event	nr_error		True when an N(R) error is detected in a received INFO or supervisory frame. Line Setup configured for emulate mode only.
extern event	ns_error		True when an N(S) error is detected in a received INFO frame. Line Setup configured for emulate mode only.

Table 76-2 (continued)

Type	Variable	Value (hex/decimal)	Meaning
extern event	frame_sent		True when frame is passed down to Layer 1. Line Setup configured for emulate mode only.
extern volatile const unsigned char	m_frame_addr	*00-ff/0-255*	Line Setup configured for emulate or monitor mode.
extern volatile const unsigned char	m_frame_type	*(same as rcvd_frame_type—Line Setup configured for emulate or monitor mode)*	
extern volatile const unsigned char	m_frame_cntrl_byte_1	*(actual value of control byte—Line Setup configured for emulate or monitor mode)*	

extern volatile const unsigned char	m_frame_pf	0
		10/16
			pf=0
			pf=1
			Line Setup configured for emulate or monitor mode.

extern volatile const unsigned char	m_frame_bcc_type	1 2 3	good bad abort Line Setup configured for emulate or monitor mode.
extern volatile const unsigned char	rcvd_frame_addr	*00-ff/0-255*	Line Setup configured for emulate mode only.
extern volatile const unsigned char	rcvd_frame_type	0 1 5 9 d/13 3 7 7 f/15 23/35 43/67 43/67 63/99 83/131 87/135 af/175 c7/199 cf/207 e3/227 ef/239 b/11 ff/240 ff/240	info rr rnr rej srej ui rim sim dm up disc rd ua snrm frmr xid cfgr snrme test bcn lpda other unknown Line Setup configured for emulate mode only.
extern volatile const unsigned char	rcvd_frame_cntrl_byte_1	*(actual value of control byte—Line Setup configured for emulate mode only)*	

extern volatile const unsigned char	rcvd_frame_pf	0
		10/16
			pf=0
			pf=1
			Line Setup configured for emulate mode only.

Table 76-2 (continued)

Type	Variable	Value (hex/decimal)	Meaning
extern volatile const unsigned char	rcvd_frame_bcc_type	1 2 3	good bad abort Line Setup configured for emulate mode only.
extern volatile const unsigned char	rcvd_frame_nr	*0-7 (MOD 8)*	Line Setup configured for emulate mode only.
extern volatile const unsigned char	rcvd_frame_ns	*0-7 (MOD 8)*	Line Setup configured for emulate mode only.
extern volatile unsigned short	rcvd_frame_buff_seg		Inter-layer message buffer number (actually, an IAPX-286 segment number) in a received frame. This segment number can be converted to a pointer by shifting it left 16 bits. Line Setup configured for emulate mode only.
extern volatile unsigned short	rcvd_frame_sdu_offset		Offset to where the service data unit begins in an inter-layer message buffer in a received frame. Add to buffer segment number (converted to pointer) to point to first byte in frame. Line Setup configured for emulate mode only.
extern volatile unsigned short	rcvd_frame_sdu_size		Size of service data unit in a received frame. Line Setup configured for emulate mode only.
extern volatile unsigned short	l2_current_window_edge		When equal to upper edge, window is full; when equal to lower edge, window is empty; when not equal to upper edge, window is not full; and when not equal to lower edge, window is not empty. Line Setup configured for emulate mode only. Valid for point-to-point operation only.
extern volatile unsigned short	l2_lower_window_edge		see l2_current_window_edge
extern volatile unsigned short	l2_upper_window_edge		see l2_current_window_edge
extern volatile unsigned short	l2_resend_edge		When resend edge is not equal to lower window edge, there is more to resend; when resend edge is equal to lower window edge, there is no more to resend. Line Setup configured for emulate mode only.

Table 76-2 (continued)

Type	Variable	Value (hex/decimal)	Meaning
extern unsigned char	l2_enhance	0	normal
		1	reverse
		4	low
		5	reverse low
		8	blink
		9	reverse blink
		12/18	blink low
			Line Setup configured for emulate or monitor mode.
extern unsigned char	l2_suppress	0	off
		1	on
			Line Setup configured for emulate or monitor mode.

Using spreadsheet tokens, the same test needs two CONDITIONS/ACTIONS blocks:

```
CONDITIONS: DTE BDBCC
ACTIONS: COUNTER bad_bcc INC
CONDITIONS: DCE BDBCC
ACTIONS: COUNTER bad_bcc INC
```

When the user selects DTE or DCE on the first rack of softkeys for Layer 2 conditions, a second rack appears from which he must select a particular frame type. A DTE INFO condition, for example, when translated, includes two C variables, one event variable and one status variable:

```
{
 dte_frame && (m_frame_type == 0)
}
```

As a C programmer, you do not have to specify a frame type. To include all frames in a condition, use the event variable only:

```
CONDITIONS:
{
 dte_frame
}
```

2. *Emulate mode only.* Some events may be detected in emulate mode only. The event variables are *rcvd_frame*, *invalid_frame*, *l2_T1*, *bcc_error*, *nr_error*, *ns_error*, and *frame_sent*.

   If you try to use one of these variables in monitor mode, you may be returned to the main program menu. When you go to the Protocol Spreadsheet and search for errors, a message like the following may be displayed: *"Error 140: Unresolved reference rcvd_frame."*

   When the user selects RCV on the first rack of softkeys for Layer 2 conditions, a second rack appears from which he must select a particular frame type. When the translator converts a RCV INFO condition into C, it will include two C variables, one event variable and one status variable:

```
{
rcvd_frame && (rcvd_frame_type == 0)
}
```

In a C condition, a frame type does not have to be specified. To include all received frames in a condition, use the event variable only:

```
CONDITIONS:
{
rcvd_frame
}
```

Error detecting may be accomplished via *bcc_error*, *nr_error*, *ns_error*, and *invalid_frame*. These variables equate to the softkey tokens bearing similar names.

One of the emulate–mode variables monitors an emulate action. The event variable *frame_sent* will come true as soon as the frame has been passed to the layer below.

(B) Status Variables

Status variables are those in Table 76-2 that do not include *event* in the Type column. Their associated event variables guarantee that they are updated and tested.

The softkey-generated condition for received Info frames is RCV INFO. The C version of the same condition should look like this:

```
CONDITIONS:
{
rcvd_frame && (rcvd_frame_type == 0)
}
```

1. *Frame characteristics.* All status variables in Table 76-2 containing an *m_* prefix are valid in either emulate or monitor mode: *m_frame_addr*, *m_frame_type*, *m_frame_cntrl_byte_1*, *m_frame_pf*, and *m_frame_bcc_type*. Use these variables to monitor a particular address, frame type, control byte, P/F value, or BCC.

   All status variables in Table 76-2 containing a *rcvd_* prefix are valid in emulate mode only: *rcvd_frame_addr*, *rcvd_frame_type*, *rcvd_frame_cntrl_byte_1*, *rcvd_frame_bcc_type*, *rcvd_frame_pf*, *rcvd_frame_nr*, and *rcvd_frame_ns*. Use these variables to monitor a particular address, frame type, control byte, BCC, or P/F, N(R), or N(S) value.

If you try to use an emulate-mode variable in monitor mode, you may be returned to the main program menu. When you go to the Protocol Spreadsheet and search for errors, a message like the following may be displayed: "*Error 140: Unresolved reference rcvd_frame_type.*"

2. *Frame buffers.* As BOP frames are received, they are automatically placed in IL message buffers to be passed up the layers. Three emulate-mode variables provide the user with access to the information in the frame that is located beyond the control byte. These variables are *rcvd_frame_buff_seg, rcvd_frame_sdu_offset,* and *rcvd_frame_sdu_size.* See Section 66.1 for a more detailed discussion of the buffer components to which these variables refer.

   Make a pointer to an IL buffer by casting *rcvd_frame_buff_seg* as a *long,* shifting it left sixteen bits, adding *rcvd_frame_sdu_offset*, and casting the result to a pointer. Increment the pointer twice (thereby adding two to the offset).

```
{
 unsigned char * ptr;
 ptr = (void *)(((long)rcvd_frame_buff_seg << 16) + rcvd_frame_sdu_offset);
 ptr+=2;
}
```

   It is now pointing at the first byte in the information field. You may continue to move through the frame for its entire length, indicated in *rcvd_frame_sdu_size.*

3. *Transmit window.* Four related variables test the status of the Layer 2 window in point-to-point operation. The particular values of these variables at any given time is not significant. What is significant is how they compare to each other. The softkey status condition on the left makes the variable comparison on the right:

WINDOW FULL	*l2_current_window_edge == l2_upper_window_edge*
WINDOW EMPTY	*l2_current_window_edge == l2_lower_window_edge*
WINDOW NOT_FULL	*l2_current_window_edge != l2_upper_window_edge*
WINDOW NOT_EMPTY	*l2_current_window_edge != l2_lower_window_edge*
MORE_TO_RESEND	*l2_resend_edge != l2_lower_window_edge*
NO_MORE_TO_RESEND	*l2_resend_edge == l2_lower_window_edge*

(C) Controlling Protocol Trace Display

To enhance or suppress particular frames on the Layer 2 Protocol Trace screen in emulate or monitor mode, assign a coded value to *l2_enhance* or *l2_suppress*. The values are listed in Table 76-2. To assign a value to either of these variables, place the statement in an ACTIONS block. For example, display RNR frames in reverse-video and suppress display of invalid frames:

```
CONDITIONS: RCV RNR
ACTIONS:
{
 l2_enhance = 1;
}
CONDITIONS: RCV INVALID
ACTIONS:
{
 l2_suppress = 1;
}
```

Check the value of these display-control variables in a CONDITIONS block

```
CONDITIONS: RCV INFO
{
 l2_enhance == 1
}
ACTIONS:
{
 l2_enhance = 0;
}
```

or an ACTIONS block:

```
CONDITIONS: RCV INFO
ACTIONS:
{
 if(l2_enhance == 1)
   l2_enhance = 0;
}
```

76.3 Routines

Use the following routines in emulate mode only. If you try to call one of these routines in monitor mode, you will be returned to the main program menu. When you go to the Protocol Spreadsheet and search for errors, a message like the following will be displayed: "*Error 140: Unresolved reference l2_give_data.*"

(A) Receive

l2_give_data

Synopsis

extern void l2_give_data();

Description

The *l2_give_data* routine takes takes an interlayer message buffer associated with a received INFO frame, changes the SDU offset to point to higher-level data, and sends a DL_DATA IND primitive up to Layer 3 along with a reference to this buffer. The softkey equivalent of this routine is the GIVE_DATA action on the Protocol Spreadsheet.

Example

Layer 3 wants access to the line in order to receive and send data. Assuming the SDLC personality package is loaded at Layer 2, enter the following program:

```
LAYER: 2
    STATE: datalink
       CONDITIONS: DL_CONNECT REQ
       ACTIONS: DL_CONNECT CONF
       CONDITIONS: DL_DATA REQ
       ACTIONS: SEND INFO "((DL_DATA))"
       CONDITIONS: RCV INFO
       ACTIONS:
       {
        l2_give_data();
       }
```

(B) Transmit

resend_frame

Synopsis

extern void resend_frame(pf, first_or_next);
unsigned char pf;
unsigned char first_or_next;

Description

The *resend_frame* routine will resend either the first or next frame in the window with the P/F bit set to a specified value.

> NOTE: To ensure that this routine executes properly, select **Emulation Addressing:** POINT-TO-POINT on the SDLC Frame Level Setup screen. For either point-to-point or multi-drop operation, see *resend_frame_multi.*

Inputs

The first parameter is the value of the P/F bit in the frame. It may be set to either zero or one.

The second parameter indicates whether the first frame in the window will be sent, or whether the next frame in the window will be sent. The first resend action will send the first frame in the window regardless of whether first or next has been selected. Legal entries are 0 (first) or 1 (next).

Example

Suppose you want to resend the entire transmit window if you receive a REJ frame.

```
LAYER: 2
    STATE: xfer
       /* Whatever conditions and actions send data precede the following condition. */

       CONDITIONS: RCV REJ RESP
       NEXT_STATE: recover
    STATE: recover
       CONDITIONS: ENTER_STATE
       ACTIONS:
       {
        resend_frame(1, 0);
       }
       CONDITIONS: FRAME_SENT
            MORE_TO_RESEND
       ACTIONS:
       {
        resend_frame(1,1);
       }
       CONDITIONS: FRAME_SENT
            NO_MORE_TO_RESEND
       NEXT_STATE: xfer
```

resend_frame_multi

Synopsis

```
extern void resend_frame_multi(pf, first_or_next, addr_value);
unsigned char pf;
unsigned char first_or_next;
unsigned char addr_value;
```

Description

The *resend_frame_multi* routine will resend either the first or next frame in the window to a specified controller address. The softkey equivalent of this routine is the (PROTOCL) RESEND action on the Protocol Spreadsheet.

> NOTE: For multi-drop operation, select **Emulation Addressing:** MULTI-DROP on the SDLC Frame Level Setup screen.

Inputs

The first parameter is the value of the P/F bit in the frame. It may be set to either zero or one.

The second parameter indicates whether the first frame in the window will be sent, or whether the next frame in the window will be sent. The first resend action will send the first frame in the window regardless of whether first or next has been selected. Legal entries are 0 (first) or 1 (next).

The third parameter is the value of the controller address. The specified address must be listed in the ADDR table on the SDLC Frame Level Setup screen (see Section 38.1). Values in the decimal range 0 through 255 are valid. Value may be represented as decimal, hexadecimal, or octal. For loopback, use the variable *rcvd_frame_addr*.

Example

Suppose you want to resend the entire transmit window for the controller address in a received REJ frame.

```
{
 extern volatile const unsigned char rcvd_frame_addr;
}
LAYER: 2
    STATE: xfer
       /* Whatever conditions and actions send data precede the following condition. */

       CONDITIONS: RCV REJ RESP
       NEXT_STATE: recover
    STATE: recover
       CONDITIONS: ENTER_STATE
       ACTIONS:
       {
        resend_frame_multi(1, 0, rcvd_frame_addr);
       }
       CONDITIONS: FRAME_SENT
           MORE_TO_RESEND
       ACTIONS:
       {
        resend_frame(1,1, rcvd_frame_addr );
       }
       CONDITIONS: FRAME_SENT
           NO_MORE_TO_RESEND
       NEXT_STATE: xfer
```

reset_nr

Synopsis

extern void reset_nr();

Description

This routine resets the N(R) field in information and supervisory frames to zero.

NOTE: To ensure that this routine executes properly, select **Emulation Addressing:** POINT-TO-POINT on the SDLC Frame Level Setup screen. For either point-to-point or multi-drop operation, see *reset_nr_multi*.

Example

When a link is established, reset N(R).

```
LAYER: 2
     STATE: reset
        CONDITIONS: ENTER_STATE
        ACTIONS: SEND SABM
        CONDITIONS: RCV UA
        ACTIONS:
        {
         reset_nr();
        }
```

reset_nr _multi

Synopsis

```
extern void reset_nr_multi(addr_value);
unsigned char addr_value;
```

Description

This routine resets to zero the N(R) field in information and supervisory frames for a specified controller address. The softkey equivalent of this routine is the (PROTOCL) RSET_NR action on the Protocol Spreadsheet.

NOTE: For multi-drop operation, select **Emulation Addressing:** MULTI-DROP on the SDLC Frame Level Setup screen.

Inputs

The only parameter is the value of the controller address. The specified address must be listed in the **ADDR** table on the SDLC Frame Level Setup screen (see Section 38.1). Values in the decimal range 0 through 255 are valid. Value may be represented as decimal, hexadecimal, or octal. For loopback, use the variable *rcvd_frame_addr*.

Example

When a link is established, reset N(R) for address in received UA frame.

```
{
 extern volatile const unsigned char rcvd_frame_addr;
}
LAYER: 2
     STATE: reset
        CONDITIONS: ENTER_STATE
        ACTIONS: SEND SABM
        CONDITIONS: RCV UA
        ACTIONS:
        {
         reset_nr_multi(rcvd_frame_addr);
        }
```

reset_ns

Synopsis

extern void reset_ns();

Description

The N(S) field in information frames is reset to zero and the transmit window is cleared.

> NOTE: To ensure that this routine executes properly, select **Emulation Addressing:** POINT-TO-POINT on the SDLC Frame Level Setup screen. For either point-to-point or multi-drop operation, see *reset_ns_multi*.

Example

When a link is established, reset N(S).

```
LAYER: 2
     STATE: reset
        CONDITIONS: ENTER_STATE
        ACTIONS: SEND SABM
        CONDITIONS: RCV UA
        ACTIONS:
        {
         reset_ns();
        }
```

reset_ns _multi

Synopsis

```
extern void reset_ns_multi(addr_value);
unsigned char addr_value;
```

Description

This routine resets to zero the N(S) field in information frames for a specified controller address. It also clears the transmit window. The softkey equivalent of this routine is the (PROTOCL) RSET_NS action on the Protocol Spreadsheet.

> NOTE: For multi-drop operation, select **Emulation Addressing:** MULTI-DROP on the SDLC Frame Level Setup screen.

Inputs

The only parameter is the value of the controller address. The specified address must be listed in the ADDR table on the SDLC Frame Level Setup screen (see Section 38.1). Values in the decimal range 0 through 255 are valid. Value may be represented as decimal, hexadecimal, or octal. For loopback, use the variable *rcvd_frame_addr*.

Example

When a link is established, reset N(S) for address in received UA frame.

```
{
 extern volatile const unsigned char rcvd_frame_addr;
}
LAYER: 2
     STATE: reset
        CONDITIONS: ENTER_STATE
        ACTIONS: SEND SABM
        CONDITIONS: RCV UA
        ACTIONS:
        {
         reset_ns_multi(rcvd_frame_addr);
        }
```

send_frame

Synopsis

```
extern void send_frame(il_buffer_number, relay_baton, data_start_offset, transmit_frame_ptr);
unsigned short il_buffer_number;
unsigned short relay_baton;
unsigned short data_start_offset;
struct send_frame_structure
{
 unsigned char addr_type;
 unsigned char frame_type;
 unsigned char nr_type;
 unsigned char ns_type;
 unsigned char p_f_type;
 unsigned char bcc_type;
 unsigned char addr_value;
 unsigned char cntrl_byte;
 unsigned char nr_value;
 unsigned char ns_value;
};
struct send_frame_structure * transmit_frame_ptr;
```

Description

The *send_frame* routine adds a frame-level header to an interlayer message buffer and passes the buffer to Layer 1. The softkey equivalent of this routine is the SEND action on the Protocol Spreadsheet.

Inputs

The first parameter is the interlayer message buffer number. See Section 66.3(A), Layer-Independent OSI routines.

The second parameter is the maintain bit used to hold the buffer while the send operation is being performed. See Section 66.3(A).

The third parameter is the offset from the beginning of the buffer to the start of the service data unit. See Section 66.3(A).

The fourth parameter is a pointer to the frame structure to be sent. For a description of *send_frame_structure* see Table 76-1.

Example

Send an Info frame containing a canned fox message and a good BCC onto the line.

```
{
 static unsigned short il_buffer_number;
 static unsigned short relay_baton;
 static unsigned short data_start_offset;
 struct send_frame_structure
  {
   unsigned char addr_type;
   unsigned char frame_type;
   unsigned char nr_type;
   unsigned char ns_type;
   unsigned char p_f_type;
   unsigned char bcc_type;
   unsigned char addr_value;
   unsigned char cntrl_byte;
   unsigned char nr_value;
   unsigned char ns_value;
  };
 struct send_frame_structure transmit_frame;
 static char transmit_string [] = "((FOX))";
}
```

```
LAYER: 2
     STATE: send_a_frame
        CONDITIONS: KEYBOARD " "
        ACTIONS:
        {
         _get_il_msg_buff(&il_buffer_number, &relay_baton);
         _start_il_buff_list(il_buffer_number,&data_start_offset);
         transmit_frame.bcc_type = 1;
         _insert_il_buff_list_cnt(il_buffer_number, data_start_offset, &transmit_string[0],
              (sizeof(transmit_string) - 1));
         send_frame(il_buffer_number, relay_baton, data_start_offset, &transmit_frame);
        }
```

77 SNA Library

When the SNA package is loaded in via the Layer Setup screen, the following external variables become available for use by the programmer. Their use on the Protocol Spreadsheet is not limited to any particular layer, though normally they belong at Layer 2.

SDLC variables and routines, while they are included in the SNA layer–personality package, are not documented here. They are documented fully in Section 76.

Those variables that are specific to the SNA package are documented here. They pertain to fields in SNA transmission headers, request/response headers, and request/response units. These variables have no spreadsheet–token equivalents.

77.1 Structures

Use the SDLC *send_frame_structure* shown in Table 76-1.

77.2 Variables

The variables discussed below apply when the Line Setup menu shows either emulate or monitor mode. Emulate mode, however, is not supported by emulate–only conditions and actions on the Protocol Spreadsheet.

(A) Monitoring Events

Use the SDLC event variables discussed in Section 76.2(A).

(B) Status Variables

All SNA variables in Table 77-1 are status variables. Also refer to the SDLC status variables listed in Table 76-2.

There are no softkey tokens on the spreadsheet that are equivalent to the SNA variables listed in Table 77-1. To search for Info frames with a FID2 transmission header, for example, use C variables. The condition should look like this:

```
CONDITIONS:
{
 dte_frame && (m_frame_type == 0) && (m_packet_fid_type == 2)
}
```

Table 77-1
SNA Variables†

Type	Variable	Value (hex/decimal)	Meaning
extern volatile unsigned short	m_packet_length		Length of the packet,including the transmission and request/response headers. Line Setup configured for emulate or monitor mode.
	Transmission Header:		
extern volatile const unsigned char	m_packet_fid_type		Format Identification Type:
		0	FID0; TH 10 bytes
		1	FID1; TH 10 bytes
		2	FID2; TH 6 bytes
		3	FID3; TH 2 bytes
		4	FID4; TH 26 bytes
		f/15	FIDF; TH 26 bytes
			Line Setup configured for emulate or monitor mode.
extern volatile const unsigned short	m_packet_daf	*0-ffff/0-65535*	Destination address field—2 bytes in FID0 and FID1; 1 byte in FID2. Line Setup configured for emulate or monitor mode.
extern volatile const unsigned short	m_packet_def	*0-ffff/0-65535*	Destination element field—2 bytes; FID4 only. Line Setup configured for emulate or monitor mode.
extern volatile const unsigned long	m_packet_dsaf	*0-ffffffff/* *0-4294967295*	Destination subarea address field—4 bytes; FID4 only. Line Setup configured for emulate or monitor mode.
extern volatile const unsigned char	m_packet_lsid		Local Session Identification: FID3 only
		(actual value	SSCP-PU session
		of byte)	SSCP-LU session
			Reserved
			LU-LU session
			Line Setup configured for emulate or monitor mode.
extern volatile const unsigned short	m_packet_oaf	*00-ff/0-255*	Origin address field—2 bytes in FID0 and FID1; 1 byte in FID2. Line Setup configured for emulate or monitor mode.
extern volatile const unsigned short	m_packet_oef	*00-ff/0-255*	Origin element field—2 bytes; FID4 only. Line Setup configured for emulate or monitor mode.
extern volatile const unsigned long	m_packet_osaf	*0-ffffffff/* *0-4294967295*	Origin subarea address field—4 bytes; FID4 only. Line Setup configured for emulate or monitor mode.

† Refer to Table 76-2 for SDLC variables.

Table 77-1 (continued)

Type	Variable	Value (hex/decimal)	Meaning
	Transmission Header (continued):		
extern volatile unsigned char *	th_ptr		Pointer for the transmission header; begins at the byte containing FID type. Line Setup configured for emulate or monitor mode.
	Request/Response Header:		
extern volatile const unsigned char	m_packet_ru_category		Request/Response Unit:
		0	Function Management Data (FMD)
		20/32	Network Control (NC)
		40/64	Data Flow Control (DFC)
		60/96	Session Control (SC)
			Line Setup configured for emulate or monitor mode.
extern volatile const unsigned char	m_packet_fi		Format Indicator:
		0	User data without header in RU
		8	In LU-LU frame, indicates header follows the RH. In SC, NC, or DFC RU, indicates a formatted RU beginning with a request code.
			Line Setup configured for emulate or monitor mode.
extern volatile const unsigned char	m_packet_rri		Request/Response Indicator:
		0	request
		80/128	response
			Line Setup configured for emulate or monitor mode.
extern volatile unsigned char	m_packet_rti		Response Type Indicator:
		0	positive response
		10/16	negative response
			Line Setup configured for emulate or monitor mode.
extern volatile unsigned char	m_packet_sdi		Sense Data Indicator:
		0	sense data not included
		4	sense data included
			Line Setup configured for emulate or monitor mode.
extern volatile unsigned char *	rh_ptr		Pointer for the request/response header; begins at the byte containing the request/response indicator. Line Setup configured for emulate or monitor mode.
	Request/Response Unit:		
extern volatile unsigned char *	ru_ptr		Pointer for the request/response unit; begins at the first byte in the unit. Line Setup configured for emulate or monitor mode.

1. *Info frame characteristics.* Most status variables in Table 77-1 contain an *m_* prefix, indicating that they are valid in emulate or monitor mode. Some variables are associated with the transmission header: *m_packet_fid_type, m_packet_daf, m_packet_def, m_packet_dsaf, m_packet_lsid, m_packet_oaf, m_packet_oef,* and *m_packet_osaf.* Other variables are associated with the request/response header: *m_packet_ru_category, m_packet_fi, m_packet_rri, m_packet_rti,* and *m_packet_sdi.*

2. *Pointers.* There are three pointers to SNA fields. *th_ptr* points to first byte of the transmission header, *rh_ptr* points to the first byte of request/response header, and *ru_ptr* points to the start of the request/response unit.

(C) Controlling Protocol Trace Display

To enhance or suppress particular packets on the Layer 2 Protocol Trace screen in emulate or monitor mode, assign a coded value to *l2_enhance* or *l2_suppress.* The values are listed in Table 76-2. To assign a value to either of these variables, place the statement in an ACTIONS block. For example, display only Info frames with FID2 transmission headers. Of these, display frames with sense data in reverse-video.

```
CONDITIONS:
{
 dte_frame && (m_frame_type == 0) && (m_packet_fid_type != 2)
}
ACTIONS:
{
 l2_suppress = 1;
}
CONDITIONS:
{
 dte_frame && (m_frame_type == 0) && (m_packet_fid_type == 2) && (m_packet_sdi == 4)
}
ACTIONS:
{
 l2_enhance = 1;
}
```

Check the value of these display-control variables in a CONDITIONS block

```
CONDITIONS:
{
 dte_frame && (m_frame_type == 0) && (m_packet_fid_type != 2) && (l2_suppress == 0)
}
ACTIONS:
{
 l2_suppress = 1;
}
```

or an ACTIONS block:

```
CONDITIONS:
{
 dte_frame && (m_frame_type == 0) && (m_packet_fid_type != 2)
}
ACTIONS:
{
 if(l2_suppress == 0)
   l2_suppress = 1;
}
```

77.3 Routines

There are no routines associated exclusively with SNA. Use the SDLC routines discussed in Section 76.3. To send a frame including SNA protocol, for example, include a *transmit_string* of SNA data in the *send_frame* routine.

78 DDCMP Library

When the DDCMP package is loaded in via the Layer Setup screen, the following external variables become available for use by the programmer. Their use on the Protocol Spreadsheet is not limited to any particular layer, though normally they belong at Layer 1.

78.1 Structures

There are no *extern* structures associated exclusively with DDCMP.

78.2 Variables

The only variables exclusive to DDCMP relate to block checking. When the DDCMP package is loaded in, the results of both header and data block checks are displayed on the data screen. If you want your program to detect good or bad BCC's, you may use the BCC selections on the trigger menus and at Layer 1 of the Protocol Spreadsheet to interrogate the header block check only.

If you want to detect a good or bad data block check, you must use one of the C event variables listed in Table 78-1.

Here is a program that counts bad DTE BCC's for both header and data:

```
{
  extern fast_event fevar_bd_bcc2_td;
}
LAYER: 1
  STATE: count_all_bad_dte_bccs
     CONDITIONS: DTE BAD_BCC
     ACTIONS: COUNTER t_bdbcc INC
     CONDITIONS:
     {
       fevar_bd_bcc2_td
     }
     ACTIONS: COUNTER t_bdbcc INC
```

78.3 Routines

There are no routines associated exclusively with DDCMP.

Table 78-1
DDCMP Variables

Type	Variable	Value (hex/decimal)	Meaning
extern fast_event	fevar_gd_bcc_rd		True when a good *header* BCC is received on RD. Line Setup configured for emulate or monitor mode.
extern fast_event	fevar_gd_bcc_td		True when a good *header* BCC is received on TD. Line Setup configured for emulate or monitor mode.
extern fast_event	fevar_bd_bcc_rd		True when a bad *header* BCC is received on RD. Line Setup configured for emulate or monitor mode.
extern fast_event	fevar_bd_bcc_td		True when a bad *header* BCC is received on TD. Line Setup configured for emulate or monitor mode.
extern fast_event	fevar_gd_bcc2_td		True when a good *data* BCC is received on TD. Line Setup configured for emulate or monitor mode.
extern fast_event	fevar_gd_bcc2_rd		True when a good *data* BCC is received on RD. Line Setup configured for emulate or monitor mode.
extern fast_event	fevar_bd_bcc2_td		True when a bad *data* BCC is received on TD. Line Setup configured for emulate or monitor mode.
extern fast_event	fevar_bd_bcc2_rd		True when a bad *data* BCC is received on RD. Line Setup configured for emulate or monitor mode.

79 ISDN D Channel Library

To use the C structures, variables, and routines explained in this section, your INTERVIEW must be equipped with Option 15. Install the ISDN Test Interface Module (TIM) in the rear of the INTERVIEW, as explained in Section 12. Also install the ISDN mux board according to the directions in Appendix J4. Load in the ISDN_D Layer 1 package via the Layer Setup screen. The ISDN_D package contains the variables and most of the routines documented below. Finally, select one of the B channels in the **Channel** field on the ISDN Interface Setup screen. See Section 51.5.

The configuration of the INTERVIEW described above supports dual-channel monitoring. Dual-channel monitoring means tracking one of the B channels and the D channel. All menu selections (with the possible exception of **Speaker** on the ISDN Interface Setup menu), triggers, and spreadsheet conditions and actions apply to the B channel selected. Use the C structures, variables, and routines in this section to monitor the D channel.

> NOTE: When the ISDN Interface Setup screen shows **Channel:** D, your unit is configured for single-channel monitoring. Menu selections, triggers, and the Protocol Spreadsheet apply to the D channel. Do not load in the ISDN_D Layer 1 package.

You may develop your own program to monitor the D channel, or simply load and run the program contained in the ISDN trace application package (available as OPT-951-35).

79.1 Structures

Use the structure *xmit_list*, shown in Table 79-1, when transmitting on the D channel via the *send_d_frame* routine. Refer to *send_d_frame* in Section 79.3 for an example of how to use this structure.

Table 79-1
ISDN Structures

Type	Variable	Value (hex/decimal)	Meaning
Structure Name:	xmit_list		Structure of a transmit list for *send_d_frame* routine. Declared as type *struct*. Reference member variables of the structure as follows: *xmit_list.string_length*.
unsigned char *	string		pointer to the location of the transmit string—the transmit string is declared separately
unsigned short	string_length	*0-ffff/0-65535*	length of the transmit string

79.2 Variables

There are three event variables associated with the ISDN_D personality package. They are *d_dte_frame*, *d_dce_frame*, and *d_rcv_frame*. See Table 79-2.

(A) Monitoring Events

1. *In monitor mode.* When a frame is detected on the D channel, one monitor event, *d_dce_frame* or *d_dte_frame*, is signaled. Use both event variables to construct an ISDN trace.

2. *In emulate mode.* In emulate mode, the receive event *d_rcv_frame* and one of the monitor events are signaled when a frame is received on the D channel. The INTERVIEW's transmissions on the D channel may not be monitored when the unit is in dual-channel mode. The implication of this difference is that ISDN trace programs written in monitor mode may not be placed intact in an emulation program.

Table 79-2
ISDN Variables

Type	Variable	Value (hex/decimal)	Meaning
extern event	d_dte_frame		True when a DTE frame is detected on the D channel. Line Setup configured for emulate or monitor mode.
extern event	d_dce_frame		True when a DCE frame is detected on the D channel. Line Setup configured for emulate or monitor mode.
extern event	d_rcv_frame		True when a frame is received on the D channel. Line Setup configured for emulate mode only.

79.3 Routines

There are two routines associated with the ISDN_D package: *send_d_frame* and *send_d_frame_il*. Another ISDN routine, *set_isdn_speaker_chan*, controls the speaker for either of the B channels. This routine is supplied by the ISDN Test Interface Module.

(A) Transmit

Use the following routines in emulate mode only. If you try to call one of these routines in monitor mode, you may be returned to the main program menu. When you go to the Protocol Spreadsheet and search for errors, a message like the following may be displayed: *"Error 140: Unresolved reference send_d_frame_il."*

send_d_frame

Synopsis

```
extern void send_d_frame(count, struct_send_string_ptr, xmit_tag);
unsigned short count;
struct xmit_list
   {
    unsigned char * string_ptr;
    unsigned short string_length;
   };
struct xmit_list * struct_send_string_ptr;
unsigned short xmit_tag;
```

Description

The *send_d_frame* routine sends a specified string on the D channel with a user-determined BCC.

Inputs

The first parameter is the number of strings to be sent.

The second parameter is a pointer to a structure which in turn identifies the location and length of each string.

The third parameter is a transmit tag which includes a BCC in bits 0–2: good (001), bad (010), or abort (011). Bits 3–7 are reserved for future use. Integers may be used to indicate the value of the transmit tag: good (1), bad (2), and abort (3).

Example

Assume you want to send on channel D a fox message inside of an X.25 data packet with a good block check. You might have 2 strings, one with the Layers 2 and 3 header information, and one with the fox message. You would send these strings as follows:

```
{
 unsigned char headers [] = {0x01, 0x00, 0x10, 0x04, 0x00};
 unsigned char message [] = "((FOX))";
 struct xmit_list
  {
   unsigned char * string;
   unsigned short string_length;
  };
 struct xmit_list send_string [] = {&headers[0], 5, &message[0], sizeof(message) - 1};
}
```

```
LAYER: 1
     STATE: send_message
        CONDITIONS: KEYBOARD " "
        ACTIONS:
        {
         send_d_frame(2, &send_string[0], 1);
        }
```

send_d_frame_il

Synopsis

```
extern void send_d_frame_il(il_buffer_number, relay_baton, data_start_offset, transmit_tag);
unsigned short il_buffer_number;
unsigned short relay_baton;
unsigned short data_start_offset;
unsigned short transmit_tag;
```

Description

This routine sends a designated interlayer message buffer out on the D channel.

Inputs

The first parameter is the interlayer message buffer number.

The second parameter is the maintain bit used to hold the buffer while the send operation is performed at Layer 1.

The third parameter is the offset from the beginning of the buffer to the service data unit (SDU).

The fourth parameter is a transmit tag which includes a BCC in bits 0–2: good (001), bad (010), or abort (011). Bits 3–7 are reserved for future use. Integers may be used to indicate the value of the transmit tag: good (1), bad (2), and abort (3).

Example

Send the same text as in the example for *send_d_frame*. Refer to Section 66.3(A) for a description of the *_get_il_msg_buff*, *_start_il_buff_list*, and *_insert_il_buff_list_cnt* routines.

```
{
 unsigned short il_buffer_number;
 unsigned short relay_baton;
 unsigned short data_start_offset;
 unsigned char message [] = "⁰₁\x000¹₀⁰₄\x000((FOX))";
}
```

```
LAYER: 1
    STATE: send_message
      CONDITIONS: KEYBOARD " "
      ACTIONS:
      {
       _get_il_msg_buff(&il_buffer_number, &relay_baton);
       _start_il_buff_list(ll_buffer_number, &data_start_offset);
       _insert_il_buff_list_cnt(il_buffer_number, data_start_offset, &message[0],
           (sizeof(message) - 1));
       send_d_frame_ll(il_buffer_number, relay_baton, data_start_offset, 1);
      }
```

(B) Speaker Control

set_isdn_speaker_chan

Synopsis

```
extern void set_isdn_speaker_chan(selection);
unsigned short selection;
```

Description

The *set_isdn_speaker_chan* routine allows the programmer to control the speaker located on the ISDN mux board, Option 15. The programmer may enable the speaker for one of the B channels. This selection is independent of the channel selected for monitor or emulation on the ISDN Interface Setup screen.

Inputs

The only parameter is the channel selection. A value of one means turn the speaker on for channel B1. Enable the speaker for channel B2 with two. Turn the speaker off by setting the value to zero.

Example

Suppose you want to know whether data or voice is being transmitted over channel B1. Use the *set_isdn_speaker_chan* routine to enable the speaker for B1. Even if you are otherwise using the INTERVIEW to monitor B2, you will hear the B1 transmissions.

```
LAYER: 1
    STATE: enable_b1
      CONDITIONS: KEYBOARD "sS"
      ACTIONS:
      {
       set_isdn_speaker_chan(1);
      }
```

80 LAPD Library

When the LAPD package is loaded in via the Layer Setup screen, the following external routines and variables become available for use by the programmer. Their use on the Protocol Spreadsheet is not limited to any particular layer, though normally they belong at Layer 2.

The variables and routines approximate LAPD Layer 2 spreadsheet-generated conditions and actions. Refer to Section 42 for more detailed explanations of the purposes of specific conditions and actions. Sometimes the name of the variable or routine is sufficient for identifying its related spreadsheet token. When this is not the case, the information is provided below.

80.1 Structures

The structure *send_frame_structure* defines the format of transmitted LAPD frames. See Table 80-1. Use this structure to send frames via the *send_frame* routine in emulate mode. See Section 80.3(B). Each variable in the structure relates to some softkey selection or user entry in the SEND action.

80.2 Variables

(A) Monitoring Events

1. *Emulate or monitor mode.* LAPD events include frames detected, good or bad BCC's, and aborts. All event variables in Table 80-2 containing a *dte_* or *dce_* prefix are valid in either emulate or monitor mode. These event variables are *dte_frame, dce_frame, dte_good_bcc, dce_good_bcc, dte_bad_bcc, dce_bad_bcc, dte_abort, dce_abort.* The variable *dce_good_bcc*, for example, equates to DCE GDBCC.

   You can use both *dte* and *dce* variables relating to the same event in one conditions block. Suppose you want to count all bad BCC's from either side of the line. Enter the following CONDITIONS/ACTIONS block:

   ```
   CONDITIONS:
   {
     dte_bad_bcc || dce_bad_bcc
   }
   ACTIONS: COUNTER bad_bcc INC
   ```

Table 80-1
LAPD Structures

Type	Variable	Value (hex/decimal)	Meaning
Structure Name:	**send_frame_structure**		Structure of a frame in LAPD. Declared as type struct. Declared automatically if a softkey-entered SEND action is taken. Program frames assigned to structure as follows: struct send_frame_structure *name*. Reference a structure variable as follows: *name*.bcc_type. If values in the frame structure are not initialized by the user, they default to 0. You may initialize the values when the structure is declared: struct send_frame_structure *name* = {1, 1, 2, 0, 0, 0, 1, 1, 1, 0, 0, 0};
unsigned char	sapi_type	1	no other value valid—indicates a value is given
unsigned char	tei_type	1	no other value valid—indicates a value is given
unsigned char	cr_type	0 1 2	0 1 loopback
unsigned char	frame_type	*(The codes for frame_type are the same as for the LAPD-variable rcvd_frame_type.)*	
unsigned char	nr_type	0 1 2 3	auto value received ns plus 1 last nr sent
unsigned char	ns_type	0 1 2 3	auto skip last nr received value
unsigned char	p_f_type	0 1 2	0 1 loopback
unsigned char	bcc_type	0 1 2 3	default (bad bcc) good bcc bad bcc abort
unsigned char	sapi_value	*00-3f/0-63*	
unsigned char	tei_value	*00-7f/0-127*	
unsigned char	cntrl_byte	*(actual value of the control byte)*	
unsigned char	nr_value	*0-7 (MOD 8)*	if nr_type = 1
unsigned char	ns_value	*0-7 (MOD 8)*	if ns_type = 3

Table 80-2
LAPD Variables

Type	Variable	Value (hex/decimal)	Meaning
extern event	dte_frame		True when a DTE frame is detected. Line Setup configured for emulate or monitor mode.
extern event	dce_frame		True when a DCE frame is detected. Line Setup configured for emulate or monitor mode.
extern event	dte_good_bcc		True when a good BCC is calculated for a DTE frame. Line Setup configured for emulate or monitor mode.
extern event	dce_good_bcc		True when a good BCC is calculated for a DCE frame. Line Setup configured for emulate or monitor mode.
extern event	dte_bad_bcc		True when a bad BCC is calculated for a DTE frame. Line Setup configured for emulate or monitor mode.
extern event	dce_bad_bcc		True when a bad BCC is calculated for a DCE frame. Line Setup configured for emulate or monitor mode.
extern event	dte_abort		True when an abort is detected for a DTE frame. Line Setup configured for emulate or monitor mode.
extern event	dce_abort		True when an abort is detected for a DCE frame. Line Setup configured for emulate or monitor mode.
extern event	rcvd_frame		True when a frame is received. Line Setup configured for emulate mode only.
extern event	invalid_frame		True when an invalid frame is detected. Line Setup configured for emulate mode only.
extern event	l2_T1		True when the T1 timeout-timer has expired. Line Setup configured for emulate mode only.
extern event	bcc_error		True when a BCC error is detected. Line Setup configured for emulate mode only.
extern event	nr_error		True when an N(R) error is detected in a received INFO or supervisory frame. Line Setup configured for emulate mode only.
extern event	ns_error		True when an N(S) error is detected in a received INFO frame. Line Setup configured for emulate mode only.

Table 80-2 (continued)

Type	Variable	Value (hex/decimal)	Meaning
extern event	frame_sent		True when frame is passed down to Layer 1. Line Setup configured for emulate mode only.
extern volatile const unsigned char	m_frame_addr_sapi	*00-3f/0-63*	Line Setup configured for emulate or monitor mode.
extern volatile const unsigned char	m_frame_addr_tei	*00-7f/0-127*	Line Setup configured for emulate or monitor mode.
extern volatile const unsigned char	m_frame_addr_cr	0 1	0 1 Line Setup configured for emulate or monitor mode.
extern volatile const unsigned char	m_frame_type	*(same as rcvd_frame_type—Line Setup configured for emulate or monitor mode)*	
extern volatile const unsigned char	m_frame_cntrl_byte_1	*(actual value of control byte—Line Setup configured for emulate or monitor mode)*	

extern volatile const unsigned char	m_frame_pf	0
		10/16
			pf=0
			pf=1
			Line Setup configured for emulate or monitor mode.

extern volatile const unsigned char	m_frame_bcc_type	1 2 3	good bad abort Line Setup configured for emulate or monitor mode.
extern volatile const unsigned char	m_frame_nr	*0-7 (MOD 8)*	Line Setup configured for emulate or monitor mode.
extern volatile const unsigned char	m_frame_ns	*0-7 (MOD 8)*	Line Setup configured for emulate or monitor mode.
extern volatile const unsigned char	rcvd_frame_addr_sapi	*00-3f/0-63*	Line Setup configured for emulate mode only.
extern volatile const unsigned char	rcvd_frame_addr_tei	*00-7f/0-127*	Line Setup configured for emulate mode only.
extern volatile const unsigned char	rcvd_frame_addr_cr	0 1 2	0 1 loopback Line Setup configured for emulate mode only.
extern volatile const unsigned char	rcvd_frame_type	0 1 3 5 9 2f/37 6f/111 43/67 f/15 f/15 63/99 67/103 87/135 e7/224 ff/255 ff/255	info rr ui rnr rej sabm sabme disc dm sarm ua sio frmr si1 other unknown Line Setup configured for emulate mode only.

Table 80-2 (continued)

Type	Variable	Value (hex/decimal)	Meaning
extern volatile const unsigned char	rcvd_frame_cntrl_byte_1	*(actual value of control byte—Line Setup configured for emulate mode only)*	

extern volatile const unsigned char	rcvd_frame_pf	0
		10/16
			pf=0
			pf=1
			Line Setup configured for emulate mode only.

extern volatile const unsigned char	rcvd_frame_bcc_type	1 2 3	good bad abort Line Setup configured for emulate mode only.
extern volatile const unsigned char	rcvd_frame_nr	*0–7 (MOD 8)*	Line Setup configured for emulate mode only.
extern volatile const unsigned char	rcvd_frame_ns	*0–7 (MOD 8)*	Line Setup configured for emulate mode only.
extern volatile unsigned short	rcvd_frame_buff_seg		Inter-layer message buffer number (actually, an IAPX-286 segment number) in a received frame. This segment number can be converted to a pointer by shifting it left 16 bits. Line Setup configured for emulate mode only.
extern volatile unsigned short	rcvd_frame_sdu_offset		Offset to where the service data unit begins in an inter-layer message buffer in a received frame. Add to buffer segment number (converted to pointer) to point to first byte in frame. Line Setup configured for emulate mode only.
extern volatile unsigned short	rcvd_frame_sdu_size		Size of service data unit in a received frame. Line Setup configured for emulate mode only.
extern volatile unsigned short	l2_current_window_edge		When equal to upper edge, window is full; when equal to lower edge, window is empty; when not equal to upper edge, window is not full; and when not equal to lower edge, window is not empty. Line Setup configured for emulate mode only.
extern volatile unsigned short	l2_lower_window_edge		see l2_current_window_edge
extern volatile unsigned short	l2_upper_window_edge		see l2_current_window_edge
extern volatile unsigned short	l2_resend_edge		When resend edge is not equal to lower window edge, there is more to resend; when resend edge is equal to lower window edge, there is no more to resend. Line Setup configured for emulate mode only.

Table 80-2 (continued)

Type	Variable	Value (hex/decimal)	Meaning
extern unsigned char	l2_enhance	0 1 4 5 8 9 12/18	normal reverse low reverse low blink reverse blink blink low Line Setup configured for emulate or monitor mode.
extern unsigned char	l2_suppress	0 1	off on Line Setup configured for emulate or monitor mode.

Using spreadsheet tokens, the same test needs two CONDITIONS/ACTIONS blocks:

```
CONDITIONS: DTE BDBCC
ACTIONS: COUNTER bad_bcc INC
CONDITIONS: DCE BDBCC
ACTIONS: COUNTER bad_bcc INC
```

When the user selects DTE or DCE on the first rack of softkeys for Layer 2 conditions, a second rack appears from which he must select a particular frame type. A DTE INFO condition, for example, when translated, includes two C variables, one event variable and one status variable:

```
{
 dte_frame && (m_frame_type == 0)
}
```

In C, the programmer does not need to specify a frame type. To include all frames in a condition, use the event variable only:

```
CONDITIONS:
{
 dte_frame
}
```

2. *Emulate mode only.* Some events may be detected in emulate mode only. The event variables are *rcvd_frame*, *invalid_frame*, *l2_T1*, *bcc_error*, *nr_error*, *ns_error*, and *frame_sent*.

   If you try to use one of these variables in monitor mode, you may be returned to the main program menu. When you go to the Protocol Spreadsheet and search for errors, a message like the following may be displayed: "*Error 140: Unresolved reference rcvd_frame.*"

   When the user selects RCV on the first rack of softkeys for Layer 2 conditions, a second rack appears from which he must select a particular frame type. When the translator converts a RCV INFO condition into C, it will include two C variables, one event variable and one status variable:

```
{
 rcvd_frame && (rcvd_frame_type == 0)
}
```

The C programmer does not have to specify a frame type. To include all received frames in a condition, use the event variable only:

```
CONDITIONS:
{
 rcvd_frame
}
```

Error detecting may be accomplished via *bcc_error*, *nr_error*, *ns_error*, and *invalid_frame*. These variables equate to the softkey tokens bearing similar names.

One of the emulate-mode variables monitors an emulate action. The event variable *frame_sent* will not come true until the frame actually has been passed to the layer below.

(B) Status Variables

Status variables are those in Table 80-2 that do not include *event* in the Type column. Their associated event variables guarantee that they are updated and tested.

The softkey-generated condition for received Info frames is RCV INFO. The C version of the same condition should look like this:

```
CONDITIONS:
{
 rcvd_frame && (rcvd_frame_type == 0)
}
```

1. *Frame characteristics.* All status variables in Table 80-2 containing an *m_* prefix are valid in either emulate or monitor mode: *m_frame_addr_sapi*, *m_frame_addr_tei*, *m_frame_addr_cr*, *m_frame_type*, *m_frame_cntrl_byte_1*, *m_frame_pf*, *m_frame_bcc_type*, *m_frame_nr*, and *m_frame_ns*.

   All status variables in Table 80-2 containing a *rcvd_* prefix are valid in emulate mode only: *rcvd_frame_addr_sapi*, *rcvd_frame_addr_tei*, *rcvd_frame_addr_cr*, *rcvd_frame_type*, *rcvd_frame_cntrl_byte_1*, *rcvd_frame_pf*, *rcvd_frame_bcc_type*, *rcvd_frame_nr*, and *rcvd_frame_ns*.

   If you try to use an emulate-mode variable in monitor mode, you may be returned to the main program menu. When you go to the Protocol Spreadsheet and search for errors, a message like the following may be displayed: "*Error 140: Unresolved reference rcvd_frame_type.*"

2. *Frame buffers.* As BOP frames are received, they are automatically placed in IL message buffers to be passed up the layers. Three emulate-mode variables provide the user with access to the information in the frame that is located beyond the control byte. These variables are *rcvd_frame_buff_seg, rcvd_frame_sdu_offset,* and *rcvd_frame_sdu_size.* See Section 66.1 for a more detailed discussion of the buffer components to which these variables refer.

   Make a pointer to an IL buffer by casting *rcvd_frame_buff_seg* as a *long,* shifting it left sixteen bits, adding *rcvd_frame_sdu_offset,* and casting the result to a pointer. Increment the pointer twice (thereby adding two to the offset).

```
{
 unsigned char * ptr;
 ptr = (void *)(((long)rcvd_frame_buff_seg << 16) + rcvd_frame_sdu_offset);
 ptr+=2;
}
```

   It is now pointing at the first byte in the information field. You may continue to move through the frame for its entire length, indicated in *rcvd_frame_sdu_size.*

3. *Transmit window.* Four related variables test the status of the Layer 2 window. The particular values of these variables at any given time is not significant. What is significant is how they compare to each other. The softkey status condition on the left makes the variable comparison on the right:

WINDOW FULL	*l2_current_window_edge == l2_upper_window_edge*
WINDOW EMPTY	*l2_current_window_edge == l2_lower_window_edge*
WINDOW NOT_FULL	*l2_current_window_edge != l2_upper_window_edge*
WINDOW NOT_EMPTY	*l2_current_window_edge != l2_lower_window_edge*
MORE_TO_RESEND	*l2_resend_edge != l2_lower_window_edge*
NO_MORE_TO_RESEND	*l2_resend_edge == l2_lower_window_edge*

(C) Controlling Protocol Trace Display

To enhance or suppress particular frames on the Layer 2 Protocol Trace screen in emulate or monitor mode, assign a coded value to *l2_enhance* or *l2_suppress.* The possible values are listed in Table 80-2. To assign a value to either of these variables, place the statement in an ACTIONS block. For example, display RNR frames in reverse-video and suppress display of invalid frames:

```
CONDITIONS: RCV RNR
ACTIONS:
{
 l2_enhance = 1;
}
CONDITIONS: RCV INVALID
ACTIONS:
{
 l2_suppress = 1;
}
```

Check the value of these display-control variables in a CONDITIONS block

```
CONDITIONS: RCV INFO
{
 l2_enhance == 1
}
ACTIONS:
{
 l2_enhance = 0;
}
```

or an ACTIONS block:

```
CONDITIONS: RCV INFO
ACTIONS:
{
 if(l2_enhance == 1)
   l2_enhance = 0;
}
```

80.3 Routines

Use the following routines in emulate mode only. If you try to call one of these routines in monitor mode, you will be returned to the main program menu. When you go to the Protocol Spreadsheet and search for errors, a message like the following will be displayed: *"Error 140: Unresolved reference l2_give_data."*

(A) Receive

l2_give_data

Synopsis

```
extern void l2_give_data();
```

Description

The *l2_give_data* routine takes an interlayer message buffer associated with a received INFO frame, changes the SDU offset to point to higher-level data, and sends a DL_DATA IND primitive up to Layer 3 along with a reference to this buffer. The softkey equivalent of this routine is the GIVE_DATA action on the Protocol Spreadsheet.

Example

Layer 3 wants access to the line in order to receive and send data. Assuming the LAPD personality package is loaded at Layer 2, enter the following program:

```
LAYER: 2
    STATE: datalink
       CONDITIONS: DL_CONNECT REQ
       ACTIONS: DL_CONNECT CONF
       CONDITIONS: DL_DATA REQ
       ACTIONS: SEND INFO "((DL_DATA))"
       CONDITIONS: RCV INFO
       ACTIONS:
       {
        l2_give_data();
       }
```

(B) Transmit

resend_frame

Synopsis

extern void resend_frame(pf, first_or_next);
unsigned char pf;
unsigned char first_or_next;

Description

The *resend_frame* routine will resend either the first or next frame in the window with the P/F bit set to a specified value. The softkey equivalent of this routine is the (PROTOCL) RESEND action on the Protocol Spreadsheet.

Inputs

The first parameter is the value of the P/F bit in the frame. It may be set to either 0 or 1.

The second parameter indicates whether the first frame in the window will be sent, or whether the next frame in the window will be sent. The first resend action will send the first frame in the window regardless of whether first or next has been selected. Legal entries are 0 (first) or 1 (next).

Example

Suppose you want to resend the entire transmit window if you receive a REJ frame.

```
LAYER: 2
    STATE: xfer
        /* Whatever conditions and actions send data precede the following condition. */

        CONDITIONS: RCV REJ RESP
        ACTIONS:
        {
         resend_frame(1, 0);
        }
        NEXT_STATE: recover
    STATE: recover
        CONDITIONS: FRAME_SENT
            MORE_TO_RESEND
        ACTIONS:
        {
         resend_frame(1,1);
        }
        CONDITIONS: FRAME_SENT
            NO_MORE_TO_RESEND
        NEXT_STATE: xfer
```

reset_nr

Synopsis

```
extern void reset_nr();
```

Description

This routine resets the N(R) field in information and supervisory frames to zero. The softkey equivalent of this routine is the (PROTOCL) RSET_NR action on the Protocol Spreadsheet.

Example

When a link is established, reset N(R).

```
LAYER: 2
    STATE: reset
        CONDITIONS: ENTER_STATE
        ACTIONS: SEND SABM
        CONDITIONS: RCV UA
        ACTIONS:
        {
         reset_nr();
        }
```

reset_ns

Synopsis

```
extern void reset_ns();
```

Description

The N(S) field in information frames is reset to zero and the transmit window is cleared. The softkey equivalent of this routine is the (PROTOCL) RSET_NS action on the Protocol Spreadsheet.

Example

When a link is established, reset N(S).

```
LAYER: 2
     STATE: reset
        CONDITIONS: ENTER_STATE
        ACTIONS: SEND SABM
        CONDITIONS: RCV UA
        ACTIONS:
        {
         reset_ns();
        }
```

send_frame

Synopsis

```
extern void send_frame(il_buffer_number, relay_baton, data_start_offset, transmit_frame_ptr);
unsigned short il_buffer_number;
unsigned short relay_baton;
unsigned short data_start_offset;
struct send_frame_structure
{
 unsigned char sapi_type;
 unsigned char tei_type;
 unsigned char cr_type;
 unsigned char frame_type;
 unsigned char nr_type;
 unsigned char ns_type;
 unsigned char p_f_type;
 unsigned char bcc_type;
 unsigned char sapi_value;
 unsigned char tei_value;
 unsigned char cntrl_byte;
 unsigned char nr_value;
 unsigned char ns_value;
};
struct send_frame_structure * transmit_frame_ptr;
```

Description

The *send_frame* routine adds a frame-level header to an interlayer message buffer and passes the buffer to Layer 1. The softkey equivalent of this routine is the SEND action on the Protocol Spreadsheet.

Inputs

The first parameter is the interlayer message buffer number. See Section 66.3(A), Layer-Independent OSI routines.

The second parameter is the maintain bit used to hold the buffer while the send operation is being performed. See Section 66.3(A).

The third parameter is the offset from the beginning of the buffer to the start of the service data unit. See Section 66.3(A).

The fourth parameter is a pointer to the frame structure to be sent. For a description of *send_frame_structure* see Table 80-1.

Example

Send an Info frame containing a canned fox message and a good BCC onto the line.

```
{
 static unsigned short il_buffer_number;
 static unsigned short relay_baton;
 static unsigned short data_start_offset;
 struct send_frame_structure
  {
   unsigned char sapi_type;
   unsigned char tei_type;
   unsigned char cr_type;
   unsigned char frame_type;
   unsigned char nr_type;
   unsigned char ns_type;
   unsigned char p_f_type;
   unsigned char bcc_type;
   unsigned char sapi_value;
   unsigned char tei_value;
   unsigned char cntrl_byte;
   unsigned char nr_value;
   unsigned char ns_value;
  };
 struct send_frame_structure transmit_frame;
 static char transmit_string [] = "((FOX)) ",
}
LAYER: 2
     STATE: send_a_frame
         CONDITIONS: KEYBOARD " "
         ACTIONS:
         {
          _get_il_msg_buff(&il_buffer_number, &relay_baton);
          _start_il_buff_list(il_buffer_number,&data_start_offset);
          transmit_frame.bcc_type = 1;
          _insert_il_buff_list_cnt(il_buffer_number, data_start_offset, &transmit_string[0],
                (sizeof(transmit_string) - 1));
          send_frame(il_buffer_number, relay_baton, data_start_offset, &transmit_frame);
         }
```

81 Q.931 Library

When the Q.931 package is loaded in via the Layer Setup screen, the following external variables become available for use by the programmer. Their use on the Protocol Spreadsheet is not limited to any particular layer, though normally they belong at Layer 3.

The variables approximate Q.931 Layer 3 spreadsheet-generated conditions and actions. Refer to Section 41 for more detailed explanations of the purposes of specific conditions and actions. Sometimes the name of the variable is sufficient for identifying its related spreadsheet token. When this is not the case, the information is provided below.

81.1 Structures

There are no *extern* structures associated exclusively with Q.931.

81.2 Variables

The variables discussed below apply when the Line Setup menu shows either emulate or monitor mode. Emulate mode, however, is not supported by emulate-only conditions and actions on the Protocol Spreadsheet.

(A) Monitoring Events

Q.931 Layer 3 event variables detect packets on either side of the line. See Table 81-1. They are valid in either emulate or monitor mode. The event variables are *dte_packet* and *dce_packet*.

When the user selects DTE or DCE on the first rack of softkeys for Layer 3 conditions, a second rack appears from which he must select a particular message type. A DTE INFO condition, for example, when translated, includes two C variables, one event variable and one status variable:

```
{
 dte_packet && (m_message_type == 0x7b)
}
```

As a C programmer, you do not need to specify a message type. To include all DTE messages in a condition, use the event variable only:

```
CONDITIONS:
{
 dte_packet
}
```

Table 81-1
Q.931 Variables

Type	Variable	Value (hex/decimal)	Meaning
extern event	dte_packet		True when a DTE packet is detected. Line Setup configured for emulate or monitor mode.
extern event	dce_packet		True when a DCE packet is detected. Line Setup configured for emulate or monitor mode.
extern volatile const unsigned char	m_packet_bcc_type	1 2 3	good bad abort Line Setup configured for emulate or monitor mode.
extern volatile const unsigned char	m_prot_disc	*00-ff/0-255*	Actual value of protocol discriminator—should be 8. Line Setup configured for emulate or monitor mode.
extern volatile const unsigned char	m_call_ref_flag	0 1	origination side destination side Line Setup configured for emulate or monitor mode.
extern volatile const unsigned char	m_message_type_defined	0	Actual value received is not a defined value for a LAPD message type.
		1	Actual value received is one of the following valid values for a LAPD message type: 1 alerting 2 call proceeding 5 setup 7 connect d/13 setup ack f/15 connect ack 20/32 user info 21/33 suspend rej 22/34 resume rej 25/37 suspend 26/38 resume 2d/45 suspend ack 2e/46 resume ack 40/64 detach 45/69 disconnect 48/72 detach ack 4d/77 release 5a/90 release complete 60/96 cancel 62/98 facility 64/100 register 68/104 cancel ack 6a/106 facility ack 6c/108 register ack

(m_message_type_defined continued on next page)

Table 81-1 (continued)

Type	Variable	Value (hex/decimal)	Meaning
	(m_message_type_defined continued)	70/112 72/114 74/116 79/121 7b/123 7d/125	cancel rej facility rej register rej congestion control info status Line Setup configured for emulate or monitor mode.
extern volatile const unsigned char	m_message_type	*00-ff/0-255*	Actual value of the message-type byte. Line Setup configured for emulate or monitor mode.
extern volatile const unsigned char	m_call_ref_len	*0-15*	Length of the call-reference value field. Line Setup configured for emulate or monitor mode.
extern volatile const unsigned char	m_info_element_len		Length of information element field. The total includes all information elements. Line Setup configured for emulate or monitor mode.
extern volatile const unsigned char *	m_ptr_to_call_ref		Pointer to the call-reference value field. Begins at the first byte, containing the call reference length. Line Setup configured for emulate or monitor mode.
extern volatile const unsigned char *	m_ptr_to_info_element		Pointer to the information element field. Begins at the first byte after the message-type byte. Line Setup configured for emulate or monitor mode.
extern unsigned char	l3_enhance	0 1 4 5 8 9 12/18	normal reverse low reverse low blink reverse blink blink low Line Setup configured for emulate or monitor mode.
extern unsigned char	l3_suppress	0 1	off on Line Setup configured for emulate or monitor mode.

(B) Status Variables

Status variables are those in Table 81-1 that do not include *event* in the Type column. Their associated event variables guarantee that they are updated and tested.

The softkey-generated condition for DTE Info frames is DTE INFO. The C version of the same condition should look like this:

```
CONDITIONS:
{
 dte_packet && (m_message_type == 0x7b)
}
```

1. *Packet characteristics.* All status variables in Table 81-1 containing an *m_* prefix are valid in either emulate or monitor mode: *m_packet_bcc_type*, *m_prot_disc*, *m_call_ref_len*, *m_call_ref_flag*, *m_message_type*, *m_message_type_defined*, and *m_info_element_len*.

2. *Pointers.* Two pointers provide access to variable-length fields. *m_ptr_to_call_ref* is the pointer to the call-reference field. *m_ptr_to_info_element* is the pointer to the information-element field.

(C) Controlling Protocol Trace Display

To enhance or suppress particular packets on the Layer 3 Protocol Trace screen in emulate or monitor mode, assign a coded value to *l3_enhance* or *l3_suppress*. The values are listed in Table 81-1. To assign a value to either of these variables, place the statement in an ACTIONS block. For example, display Suspend messages in reverse-video and suppress display of Status messages:

```
CONDITIONS: DTE SUSPEND
ACTIONS:
{
 l3_enhance = 1;
}
CONDITIONS: DTE STATUS
ACTIONS:
{
 l3_suppress = 1;
}
```

Check the value of these display-control variables in a CONDITIONS block

```
CONDITIONS: DTE INFO
{
 l3_enhance == 1
}
ACTIONS:
{
 l3_enhance = 0;
}
```

or an ACTIONS block:

```
CONDITIONS: DTE INFO
ACTIONS:
{
 if(l3_enhance == 1)
   l3_enhance = 0;
}
```

81.3 Routines

There are no routines associated exclusively with Q.931.

82 SS#7 Layer 2 Library

When the SS#7 Layer 2 package is loaded in via the Layer Setup screen, most of the following external variables become available for use by the programmer. Their use on the Protocol Spreadsheet is not limited to any particular layer, though normally they belong at Layer 2.

The SS#7 Layer 1 variables shown in Table 82-2 are accessible only when the Layer 1 SS7_CMPRESN package is loaded in via the Layer Setup screen. They do not have related spreadsheet tokens. These Layer 1 variables are included in this section since they are associated with the Layer 2 event variables in Table 82-1.

The Layer 2 variables approximate SS#7 Layer 2 spreadsheet-generated conditions and actions. Refer to Section 45 for more detailed explanations of the purposes of specific conditions and actions. Sometimes the name of the variable is sufficient for identifying its related spreadsheet token. When this is not the case, the information is provided below.

82.1 Structures

There are no *extern* structures associated exclusively with SS#7.

82.2 Variables

The variables discussed below apply when the Line Setup menu shows either emulate or monitor mode. Emulate mode, however, is not supported by emulate-only conditions and actions on the Protocol Spreadsheet.

(A) Monitoring Events

SS#7 Layer 2 events include frames detected, good or bad BCC's, and aborts. All event variables in Table 82-1 containing a *dte_* or *dce_* prefix are valid in either emulate or monitor mode. These event variables are *dte_frame, dce_frame, dte_good_bcc, dce_good_bcc, dte_bad_bcc, dce_bad_bcc, dte_abort, dce_abort.*

You can use both *dte* and *dce* variables relating to the same event in one conditions block. Suppose you want to count all bad BCC's from either side of the line. Enter the following CONDITIONS/ACTIONS block:

```
CONDITIONS:
{
 dte_bad_bcc || dce_bad_bcc
}
ACTIONS: COUNTER bad_bcc INC
```

Table 82-1
SS#7 Layer 2 Variables

Type	Variable	Value (hex/decimal)	Meaning
extern event	dte_frame		True when a non-suppressed DTE frame is detected. Line Setup configured for emulate or monitor mode.
extern event	dce_frame		True when a non-suppressed DCE frame is detected. Line Setup configured for emulate or monitor mode.
extern event	dte_good_bcc		True when a non-suppressed good BCC is calculated for a DTE frame. Line Setup configured for emulate or monitor mode.
extern event	dce_good_bcc		True when a non-suppressed good BCC is calculated for a DCE frame. Line Setup configured for emulate or monitor mode.
extern event	dte_bad_bcc		True when a bad BCC is calculated for a DTE frame. Line Setup configured for emulate or monitor mode.
extern event	dce_bad_bcc		True when a bad BCC is calculated for a DCE frame. Line Setup configured for emulate or monitor mode.
extern event	dte_abort		True when an abort is detected for a DTE frame. Line Setup configured for emulate or monitor mode.
extern event	dce_abort		True when an abort is detected for a DCE frame. Line Setup configured for emulate or monitor mode.
extern volatile const unsigned char	m_unit_type	1 2 3	Fill-in Signal Unit (FI) Link Status Signal Unit (LSU) Message Signal Unit (MSU) Line Setup configured for emulate or monitor mode.
extern volatile const unsigned char	m_bib	0 *non-zero*	0 1 Line Setup configured for emulate or monitor mode.
extern volatile const unsigned char	m_fib	0 1	0 1 Line Setup configured for emulate or monitor mode.

Table 82-1 (continued)

Type	Variable	Value (hex/decimal)	Meaning
extern volatile const unsigned char	m_li	0	FI
		1-2	LSU
		3-3f/63	MSU
			Line Setup configured for emulate or monitor mode.
extern volatile const unsigned char	m_so0	0	out of alignment
		1	normal
		2	emergency
		3	out of service
		4	processor out
		5	busy
			Line Setup configured for emulate or monitor mode.
extern volatile const unsigned char	m_frame_bcc_type	1	good bcc
		2	bad bcc
		3	abort
			Line Setup configured for emulate or monitor mode.
extern unsigned char	l2_enhance	0	normal
		1	reverse
		4	low
		5	reverse low
		8	blink
		9	reverse blink
		12/18	blink low
			Line Setup configured for emulate or monitor mode.
extern unsigned char	l2_suppress	0	off
		1	on
			Line Setup configured for emulate or monitor mode.

When the user selects DTE or DCE on the first rack of softkeys for Layer 2 conditions, a second rack appears from which he must select a particular frame type. A DTE FILL_IN condition, for example, when translated, includes two C variables, one event variable and one status variable:

```
{
 dte_frame && (m_unit_type == 1)
}
```

The C programmer does not need to specify a frame type. To include all frames in a condition, use the event variable only:

```
CONDITIONS:
{
 dte_frame
}
```

(B) Status Variables

Status variables are those in Table 82-1 that do not include *event* in the Type column. Their associated event variables guarantee that they are updated and tested.

The softkey-generated condition for DTE Busy Link Status Signal Unit is DTE STATUS= B. The C version of the same condition should look like this:

```
CONDITIONS:
{
 dte_frame && (m_unit_type == 2) && (m_so0 == 5)
}
```

Status variables in Table 82-1 containing an *m_* prefix are valid in either emulate or monitor mode: *m_unit_type*, *m_bib*, *m_fib*, *m_li*, *m_so0*, and *m_frame_bcc_type*.

The Layer 1 variables listed in Table 82-2 are also status variables, valid in either emulate or monitor mode. Any of the Layer 2 event variables in Table 82-1 guarantee that they are updated and tested.

> NOTE: The SS#7 Layer 1 variables are updated frequently. If you want to track these variables for statistical purposes, we recommend that you copy their values into temporary variables.

(C) Controlling Protocol Trace Display

To enhance or suppress particular frames on the Layer 2 Protocol Trace screen in emulate or monitor mode, assign a coded value to *l2_enhance* or *l2_suppress*. The values are listed in Table 82-1. To assign a value to either of these variables, place the statement in an ACTIONS block. For example, display only Link Signal Units. Of these, display Emergency LSU's in reverse-video.

```
CONDITIONS:
{
 dte_frame && (m_unit_type != 2)
}
ACTIONS:
{
 l2_suppress = 1;
}
CONDITIONS:
{
 dte_frame && (m_unit_type == 2) && (m_so0 == 2)
}
ACTIONS:
{
 l2_enhance = 1;
}
```

Table 82-2
SS#7 Layer 1 Variables

Type	Variable	Value (hex/decimal)	Meaning
extern unsigned short	dte_frames_suppressed		Number of DTE Fill-In or Link Status Signal Units suppressed since the last non-suppressed frame. Line Setup configured for emulate or monitor mode.
extern unsigned short	dce_frames_suppressed		Number of DCE Fill-In or Link Status Signal Units suppressed since the last non-suppressed frame. Line Setup configured for emulate or monitor mode.
extern unsigned short	dte_flags		Number of DTE flags received since the last non-suppressed frame. Line Setup configured for emulate or monitor mode.
extern unsigned short	dce_flags		Number of DCE flags received since the last non-suppressed frame. Line Setup configured for emulate or monitor mode.

Check the value of these display-control variables in a CONDITIONS block

```
CONDITIONS:
{
 dte_frame && (m_unit_type == 2) && (m_so0 == 2) && (l2_enhance == 0)
}
ACTIONS:
{
 l2_enhance = 1;
}
```

or an ACTIONS block:

```
CONDITIONS:
{
 dte_frame && (m_unit_type == 2) && (m_so0 == 2)
}
ACTIONS:
{
 if(l2_enhance == 0)
   l2_enhance = 1;
}
```

82.3 Routines

There are no routines associated exclusively with SS#7.

83 SS#7 Layer 3 Library

When the SS#7 Layer 3 package is loaded in via the Layer Setup screen, the following external variables become available for use by the programmer. Their use on the Protocol Spreadsheet is not limited to any particular layer, though normally they belong at Layer 3.

The variables approximate SS#7 Layer 3 spreadsheet-generated conditions and actions. Refer to Section 46 for more detailed explanations of the purposes of specific conditions and actions. Sometimes the name of the variable is sufficient for identifying its related spreadsheet token. When this is not the case, the information is provided below.

83.1 Structures

There are no *extern* structures associated exclusively with SS#7.

83.2 Variables

The variables discussed below apply when the Line Setup menu shows either emulate or monitor mode. Emulate mode, however, is not supported by emulate-only conditions and actions on the Protocol Spreadsheet.

(A) Monitoring Events

SS#7 Layer 3 event variables detect Message Signal Units on either side of the line. See Table 83-1. They are valid in either emulate or monitor mode. The event variables are *dte_packet* and *dce_packet.*

When the user selects DTE or DCE on the first rack of softkeys for Layer 3 conditions, a second rack appears from which he must select a particular MSU type. A DTE NETM condition, for example, when translated, includes two C variables, one event variable and one status variable:

```
{
 dte_packet && (m_sio_si == 0)
}
```

As a C programmer, you do not have to specify an MSU type. To include all DTE Message Signal Units in a condition, use the event variable only:

```
CONDITIONS:
{
 dte_packet
}
```

Table 83-1
SS#7 Layer 3 Variables

Type	Variable	Value (hex/decimal)	Meaning
extern event	dce_packet		True when a DCE packet is detected. Line Setup configured for emulate or monitor mode.
extern event	dte_packet		True when a DTE packet is detected. Line Setup configured for emulate or monitor mode.
extern volatile const unsigned char	m_sio_ni	0	International 0
		40/64	International 1
		80/128	national 0
		c0/192	national 1
			Line Setup configured for emulate or monitor mode.
extern volatile const unsigned char	m_sio_priority	0	priority=0
		10/16	priority=1
		20/32	priority=2
		30/48	priority=3
			Line Setup configured for emulate or monitor mode.
extern volatile const unsigned char	m_sio_si	*0-7*	User Part:
		0	netm
		1	ntr
		2	nts
		3	sccp
		4	tup
		5	isdn
		6	dup0
		7	dup1
		8-f/8-15	spare
			Line Setup configured for emulate or monitor mode.
extern volatile const unsigned char	m_code_type		Test headers:† (high 4 bits not defined)
		1	ltm
		2	lta

(m_code_type continued on next page)

† The high four bits in test headers are not defined. To check the value of *m_code_type* for test headers, *and* *m_code_type* with 0x0f:

header = m_code_type & 0x0f;

For LTM's, *header* equals 1; for LTA's, *header* equals 2.

Table 83-1 (continued)

Type	Variable	Value (hex/decimal)	Meaning
(m_code_type continued)			SCCP headers:
		1	cr
		2	cc
		3	cref
		4	rlsd
		5	rlc
		6	dt1
		7	dt2
		8	ak
		9	udt
		a/10	udts
		b/11	ed
		c/12	ea
		d/13	rsr
		e/14	rsc
		f/15	err
		10/16	it
			NETM headers:
		11/17	coo
		12/18	eco
		13/19	rct
		14/20	tfp
		15/21	rsp (US format only)
		15/21	rst (CCITT format only)
		16/22	lin
		18/24	dlc
		21/33	coa
		22/34	eca
		23/35	tfc
		24/36	tcp (US format only)
		25/37	rsr (US format only)
		25/37	rst (CCITT format only, national option)
		26/38	lun
		28/40	css
		34/52	tfr
		35/53	rcp (US format only)
		36/54	lia
		38/56	cns
		44/68	tcr (US format only)
		45/69	rcr (US format only)
		46/70	lua
		48/72	cnp
		51/81	cbd
		54/84	tfa
		56/86	lid
		61/96	cba
		64/100	tca (US format only)
		66/102	lfu
		76/118	lli
		86/134	lri

(m_code_type continued on next page)

Table 83-1 (continued)

Type	Variable	Value (hex/decimal)	Meaning
(m_code_type continued)			TUP headers:
		6	anu
		10	reserved
		11/17	iam
		12/18	gsm
		13/19	grq
		14/20	acm
		15/21	sec
		16/22	anc
		17/23	rlg
		18/24	mgb
		19/25	cfm
		21/33	iai
		24/36	chg
		25/37	cgc
		26/38	ann
		27/39	blo
		28/40	mba
		29/41	cpm
		31/49	sam
		32/50	cot
		35/53	nnc
		36/54	cbk
		37/55	bla
		38/56	mgu
		39/57	cpa
		41/65	sao
		42/66	ccf
		45/69	adi
		46/70	clf
		47/71	ubl
		48/72	mua
		49/73	csv
		55/85	cfl
		56/86	ran
		57/87	uba
		58/88	hgb
		59/89	cvm
		65/101	ssb
		66/102	fot
		67/103	ccr
		68/104	hba
		69/105	crm
		75/117	unn
		76/118	ccl
		77/119	rsc
		78/120	hgu
		79/121	cll
		85/133	los
		88/136	hua

(m_code_type continued on next page)

Table 83-1 (continued)

Type	Variable	Value (hex/decimal)	Meaning
(m_code_type continued)			*(TUP headers continued)*
		95/149	sst
		98/152	grs
		a5/165	acb
		a8/168	gra
		b5/181	dpn
		b8/184	sgb
		c5/197	mpr
		c8/200	sba
		d8/216	sgu
		e8/232	sua
		f5/245	eum
		f6/246	eam
			<u>ISDN headers</u>:
		1	iam
		2	sam
		3	inr
		4	inf
		5	cot
		6	acm
		8	fot
		9	anm
		a/10	ubm
		b/11	rel
		d/12	pau
		e/14	res
		f/15	rlsd
		10/16	rlc
		11/17	ccr
		12/18	rsc
		13/19	blo
		14/20	ubl
		15/21	bla
		16/22	uba
		17/23	grs
		18/24	cgb
		19/25	cgu
		1a/26	cgba
		1b/27	cgua
		1c/28	cmr
		1d/29	cmc
		1e/30	rcm
		1f/31	far
		20/32	faa
		21/33	frj
		22/34	fad
		23/35	fal
		25/37	csvr
		26/38	csvs
		27/39	drs
		28/40	pam
		29/41	gra
			Line Setup configured for emulate or monitor mode.

Table 83-1 (continued)

Type	Variable	Value (hex/decimal)	Meaning
extern volatile unsigned long	m_label_dpc	*0-3ffff/* *0-16383*	CCITT format (2 bytes)
		0-ffffff/ *0-16777215*	ANSI format (3 bytes)
			Line Setup configured for emulate or monitor mode.
extern volatile unsigned long	m_label_opc	*0-3ffff/* *0-16383*	CCITT format (2 bytes)
		0-ffffff/ *0-16777215*	ANSI format (3 bytes)
			Line Setup configured for emulate or monitor mode.
extern volatile const unsigned char	m_label_sls	*0-f/0-15*	CCITT format
		0-1f/0-31	ANSI format
			Line Setup configured for emulate or monitor mode.
extern volatile unsigned short	m_cic	*0-fff/0-4095*	TUP MSUs
		0-ffff/0-65535	ISDN MSUs
			Line Setup configured for emulate or monitor mode.
extern unsigned char	l3_enhance	0	normal
		1	reverse
		4	low
		5	reverse low
		8	blink
		9	reverse blink
		12/18	blink low
			Line Setup configured for emulate or monitor mode.
extern unsigned char	l3_suppress	0	off
		1	on
			Line Setup configured for emulate or monitor mode.

(B) Status Variables

Status variables are those in Table 83-1 that do not include *event* in the Type column. Their associated event variables guarantee that they are updated and tested.

The softkey-generated condition for NETM Message Signal Units on the DTE side of the line is DTE NETM. The C version of the same condition should look like this:

```
CONDITIONS:
{
 dte_packet && (m_sio_si == 0)
}
```

Most status variables in Table 83-1 contain an *m_* prefix: *m_sio_ni, m_sio_priority, m_sio_si, m_code_type, m_label_dpc, m_label_opc, m_label_sls,* and *m_cic.*

(C) Controlling Protocol Trace Display

To enhance or suppress particular packets on the Layer 3 Protocol Trace screen in emulate or monitor mode, assign a coded value to *l3_enhance* or *l3_suppress*. The values are listed in Table 83-1. To assign a value to either of these variables, place the statement in an ACTIONS block. For example, display only messages with NETM headers. Of these, display Transfer Restricted headers in reverse-video.

```
CONDITIONS:
{
 dte_packet && (m_sio_si != 0)
}
ACTIONS:
{
 l3_suppress = 1;
}
CONDITIONS:
{
 dte_packet && (m_sio_si == 0) && (m_code_type == 0x34)
}
ACTIONS:
{
 l3_enhance = 1;
}
```

Check the value of these display-control variables in a CONDITIONS block

```
CONDITIONS:
{
 dte_packet && (m_sio_si != 0) && (l2_suppress == 0)
}
ACTIONS:
{
 l2_suppress = 1;
}
```

or an ACTIONS block:

```
CONDITIONS:
{
 dte_packet && (m_sio_si != 0)
}
ACTIONS:
{
 if(l2_suppress == 0)
   l2_suppress = 1;
}
```

83.3 Routines

There are no routines associated exclusively with SS#7.