Figure 51-1 Block diagram of INTERVIEW 7000 Series hardware architecture.
51 Data Flow

Figure 51-1 is a block diagram showing the components on each of the six types of logic board in the INTERVIEW 7000 Series. The components on the TIM (Test Interface Module) also are shown. Figure 51-2 indicates the flow of data among the various functional components of the unit.

51.1 Two Types of CPU

The brain of the INTERVIEW is the Motorola 68010 processor on the CPM (Central Processing Module). See Figure 51-1. The 68010 processor controls operations in the unit not directly under control of the user program. 68010 operations include fetching power-up software and initialization routines from the EPROM, controlling disk I/O, and maintaining setup and statistics screens. The operating system in the 68010 is pSOS.

An Intel 80286 processor controls the operation of the MPM (Main Processor Module). The MPM does all higher level processing of receive data. The board also generates the transmit data to be sent out in emulate mode. The 80286 uses a basic, multitasking real–time executive operating system.

An INTERVIEW 7000 and 7200 TURBO may have from one to three MPMs, each with its own 80286 CPU. The INTERVIEW 7500 and 7700 TURBO always have three MPMs.
INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

Figure 51-2 INTERVIEW 7000 Series functional diagram.
The 80286 operates on software located in the DRAM on the MPM. See Figure 51-1. This software is the user program—setups, trigger menus, protocol spreadsheet, and protocol state machines (layer packages)—translated and compiled by the CPM and loaded into the MPM. The program will tell the MPM how to process the data, what trigger conditions to look for in the data stream, etc.

The CPM polls the MPM continuously to see if data is available to be output to the printer or the plasma display. This data includes character data, trace data, prompts, and values to be posted to the statistics screens.

While the CPM accesses the MPM on a regular basis, there is no access in the reverse direction. That is, the user program running on the MPM has no direct access to the CPM. The user cannot write to one of the menu screens, for example.

### 51.2 Front-End Buffer

Note in Figure 51-2 that the front-end buffer (FEB) lies squarely between the line interface and (1) the recording medium and (2) the program logic on the MPM. This means that control leads may or may not be recorded and may or may not be seen by the trigger-menu and spreadsheet conditions—depending on the FEB setup (see Section 7).

Once control leads and time ticks (that is, the original timing values) are recorded alongside character data, they are locked in. Since the FEB is not on the playback path for character data, FEB selections do not apply.

Bit-image data, however, does pass through the FEB during playback. Except for the Idle Suppress field, FEB selections apply. This means that control leads and time ticks, if recorded with the data, must be enabled in order for the program logic to detect them.

Not only characters but also leads and time ticks, if enabled in the FEB setup, are captured automatically in the display buffer (that is, the screen buffer or character RAM).

Data, time ticks, and control leads are encoded in a special storage format by a data-encoder chip on the FEB board. See Figure 51-1. The encoded data is buffered to be sent to the PCM (Peripheral Control Module) for recording and to the MPM for processing.

*The encoding process is driven by clock pulses on the line interface.* This means that in the absence of external clock (or, if the INTERVIEW is emulating DCE, in the absence of internal clock), neither line data, time ticks nor EIA leads will be recorded or presented to the receivers and to the program logic.
52 Program Main

Softkey-selectable programming "tokens" entered by the user on the Protocol Spreadsheet are translated automatically into C during the initial compiler phases after \texttt{[H]} is pressed. Trigger Menu setups also are translated into C. When the translation is complete, the compiler takes over and converts the C code into object code. The C variables and routines used by the translator are documented throughout this volume.

Briefly, the translator makes the following conversions: it turns TESTs into tasks; STATE names into labels; STATES into \texttt{waitfor} clauses; CONDITIONS into \texttt{waitfor} expressions that include event variables; and ACTIONS into statements and routines, also inside of \texttt{waitfor} clauses.

Then the translator creates a program \texttt{main} function that calls every task in the program.

52.1 Translating a Simple Test into C

Suppose that the following simple program, intended to sound the INTERVIEW's alarm at 1 P.M., has been entered on the Protocol Spreadsheet.

\begin{verbatim}
STATE: sample1
CONDITIONS: TIME 1300
ACTIONS: ALARM
\end{verbatim}

When the user presses \texttt{[H]}, roughly the following C coding (with some extraneous code removed for clarity) is generated and then compiled:

```c
extern fast_event fevar_time_of_day;
extern volatile unsigned short crnt_time_of_day;
task
{
  main ()
  {
    state_sample1:
      waitfor
      {
        fevar_time_of_day && (crnt_time_of_day == 1300):
        {
          sound_alarm ():
        }
      }
    }
  } dtest_0;
  main ()
  {
    dtest_0 ():
  }

52-1
```
Note that the translator has assigned state_sample1 to a default TEST named dtest_0. It converted the TEST into a task and placed state_sample1 inside of the task. Then it created a program main function and used the program main to call every test/task in the program. The tasks appear in the task list in the same order in which they appear in the spreadsheet program. In this instance there was only one task to call.

If you try to enter the program above on the spreadsheet entirely in C, in the first place you will have to surround it with a pair of curly braces. Then it will not compile. The translator does not look inside of curly braces (except to expand constants). It simply lifts up the braced C regions and places them intact into its translation of the softkey portion of the program, before adding a program main—even when, as in this instance, a program main already is included in a C region. The two main functions conflict here, and the compiler issues the error message, “Error 109: Function main redefined.”

If we were to remove the main function from our C version, the program would compile but it still would not work. Here’s why. When the translator looks at a program made up entirely of C code, it doesn’t see anything. So it creates a program main with a task-list that is empty. The task that is declared in the program above (dtest_0) is never called.

The rule, then, is that a Protocol Spreadsheet program containing tasks written in C must always have at least one softkey STATE (with its implied task) that calls all the tasks.

52.2 A Minimum of One Softkey State

Here is a Protocol Spreadsheet test that works and yet has the minimum number of softkey tokens—one. Note that we have given the task dtest_0 a new name, since the translator will declare the task-name dtest_0 as the default test for our new softkey state, task_list.

```c
{ extern fast_event fevar_time_of_day; extern volatile unsigned short crnt_time_of_day; task
  { main ()
   { state_sample1:
     waitfor
     { fevar_time_of_day && (crnt_time_of_day == 1300):
       { sound_alarm ();
       }
     }
   } c_test;
 } c_test;
} STATE: task_list
  { c_test ();
  }
```
And here is the program as it is actually compiled. Note that the translator has added a program main that calls `dtest_0` (which in turn calls `c_test`).

```c
extern fast_event fevar_time_of_day;
extern volatile unsigned short crnt_time_of_day;
task
{
    main()
    {
        state_sample1:
        waitfor
        {
            fevar_time_of_day && (crnt_time_of_day == 1300):
            {
                sound_alarm();
            }
        }
    } c_test;
task
    {
        main()
        {
            state_task_list:
            {
                c_test();
                waitfor
                {
                } /* This empty waitfor is automatically generated in any state that does not contain a waitfor. */
            }
        }
    } dtest_0;
    main()
    {
        dtest_0();
    }
}
```

52.3 Writing the Test Entirely in C

The INTERVIEW is equipped with tools—namely, the `#pragma hook 0` preprocessor directive and linkable-object (LOBJ) files—that make it possible to write a version of the test completely in C.

NOTE: For more information on `#pragma hook` directives, see Section 56.4. Refer also to Section 13.3(P) on linkable-object files.

Write the following C code to an ASCII file (`hook_cctest.s`) using the Protocol Spreadsheet editor's WRITE/U command. Then delete the code from the spreadsheet. Go to the File Maintenance screen and and create a linkable-object file (`hook_cctest.o`) using the Compile command.
#pragma hook 0 "c_test();"
extern fast_event fevar_time_of_day;
extern volatile unsigned short crnt_time_of_day;
task
{
    main()
    {
        state_sample1:
        waitfor
        {
            fevar_time_of_day && (crnt_time_of_day == 1300):
            {
                sound_alarm();
            }
        }
    }
} c_test_task;
c_test()
{
c_test_task();
}

Notice that the "hook" is a call to the routine c_test. This routine's only purpose is to start the task, c_test_task. A task name is always local to a linkable-object file and never directly copied from it. If you try to call the task directly in the #pragma hook 0 directive, therefore, the spreadsheet program (shown below) will not compile. Since the task name is local to the file, the following error message will be displayed: "Error 140: Unresolved reference c_test_task." The rule for including tasks in a linkable-object file, then, is to let the #pragma hook 0 directive call a routine which starts the task(s).

NOTE: Since task names are local to a file, the definition of c_test_task also cannot be located in a referenced LOBJ file different from the one in which it is called.

The Protocol Spreadsheet program required to execute the test consists of a single line:

OBJECT: "hook_ctest.o"

When translated, the program looks like this:

    #pragma object "hook_ctest.o"
    main()
    {
        c_test();
    }

Notice that the routine c_test is located within the top-level program main. The hook text from a #pragma hook 0 directive is always put at the end of main's task list. At this point, since c_test has not been previously declared, it is assumed to be an extern function (not a task) that returns an int. The linkable-object file(s) referenced in the spreadsheet program will be searched for the routine's definition.
53 Regions in Spreadsheet

C language can be embedded in a Protocol Spreadsheet program at several access points. A C region can be opened at the top of the program, or in an OBJECT, CONSTANTS, LAYER, TEST, STATE, CONDITIONS, or ACTIONS block.

At these points, simply begin the C region with an opening curly brace. Make your entry and terminate it with a closing curly brace.

The remainder of this section describes C code blocks related to the spreadsheet components, from largest to smallest.

53.1 Layer and Test

The main function of a task is the highest level function that may be programmed by the user of the INTERVIEW 7000 Series. The keyword task in a C region corresponds to the TEST: softkey token on the Protocol Spreadsheet. Typing TEST: keyboard_alarm on the spreadsheet is the equivalent of the following C coding:

```c
task
{
    #pragma layer 1
    main()
    {
        /* declarations, state-labels, and statements go here */
    }
}
layer_1_test_keyboard_alarm;
```

The INTERVIEW is multitasking, so more than one task/test may be defined. All tasks/tests run concurrently if they are included in the task list created by the translator when it generates the program main function. See Section 52, Program Main, for an explanation of how this automatic program main is created.

Layers have no existence in C independent of the tasks that they contain. When a user enters the LAYER: token on the spreadsheet followed by a layer number, the C translator prefixes that number to the name of each task that follows. Note in the example above that the test name keyboard_alarm was given a layer_1_test prefix.

The C translator also issued the preprocessor directive #pragma layer 1. The compiler uses this layer declaration to distribute tasks efficiently among 80286 processors. This pragma is an optimizing feature and is not strictly required in the body of the task.

The C translator does nothing else with the layer number other than convert it into a prefix to the task name and construct the #pragma directive.
The layer number does, of course, determine many of the branching softkey selections that will be available to the user who is not programming in C. The C programmer will find that none of the variables or routines mentioned in this manual is specific to a particular layer. A variable or routine that is supplied, for example, by the X.25 Layer 3 personality package (at the time that the package is loaded in via the Layer Setup screen) will still be available inside of a task that nominally belongs to Layer 1 or Layer 2.

```
Test ----> task
{   main()
    {    static label current_state;
        States declared here
    }
} layer_n_test_name;
```

Figure 53-1 C equivalent of a spreadsheet test.

### 53.2 State, Enter State, and Next State

A STATE on the Protocol Spreadsheet is a label in C, used as a target of a goto statement. Typing STATE: alarm_on on the spreadsheet is the equivalent of this C coding, placed inside of the braces that follow the task main:

```
static label current_state;
state_alarm_on:
    current_state = state_alarm_on_loop;
{   /* statements go here */
    goto (current_state);
    state_alarm_on_loop:
    waitfor
    {       /* condition clauses go here, each comprised of expression, colon(:), and statements */
    }   goto (current_state);
}
```

Note that the C translator has taken STATE: alarm_on on the Protocol Spreadsheet and produced two state labels, state_alarm_on and state_alarm_on_loop. The first state label is followed by statements that will be executed immediately upon entering the state. The “loop”-state label always introduces a waitfor construction. Both states end in a statement to goto (current_state).
The translator's version of a state includes overhead to cover all cases, including special cases. The loop state is not strictly required, and a streamlined version of the basic state coding that eliminates the extra state will work in most instances:

```
static label current_state;
state_alarm_on:
{
    /* declarations and statements go here */
    waitfor
    {
        /* condition clauses go here, each comprised of expression, colon(:), and statement(s) */
    }
    goto (current_state);
}
```

Note these points about states created entirely by the programmer:

- A `goto` statement cannot be used inside of a `waitfor` construction.
- You must use a `break` statement to exit the `waitfor` construction.
- You may dispense with the `current_state` variable and `goto` a state label, in which case the opening and closing parens may be omitted.

(A) Declaring States

The state name followed by the colon (:) is itself a label declaration and does not require an additional declaration.

(B) Enter State

The C translator puts a `waitfor` construction into every "loop" state. If you want a statement to be executed immediately without waiting for an event, you may place that statement in the nonloop state, outside of the `waitfor` statement. The following is an example of a state in which the `sound_alarm` routine is executed immediately.

```
static label current_state;
state_alarm_on:
current_state = state_alarm_on_loop;
{
    sound_alarm();
    goto (current_state);
    state_alarm_on_loop:
    waitfor
    {
        goto (current_state);
    }
}
```

The example above is the equivalent of this spreadsheet entry:

```
STATE: alarm_on
CONDITIONS: ENTER_STATE
ACTIONS: ALARM
```

A hybrid version also may be created:

```
STATE: alarm_on
{
    sound_alarm();
}
```
The `sound_alarm` function is executed immediately, since the translator places it above the `waitfor`. When you enter a `CONDITIONS:` block on the spreadsheet, you move inside a `waitfor`—unless you place your C region immediately following an `ENTER_STATE`.

An `ENTER_STATE` condition may cause the translator to generate an `if` statement in the nonloop state (above the `waitfor` state). Here is a spreadsheet example:

```
STATE: alarm_on
CONDITIONS: ENTER_STATE
COUNTER anyname EQ 3
ACTIONS: ALARM
```

This is the C version:

```c
static label current_state;
state_alarm_on:
current_state = state_alarm_on_loop;
{
    if (counter_anyname.current == 3) sound_alarm();
goto (current_state);
state_alarm_on_loop:
waitfor
}
goto (current_state);
}
```

And here is a hybrid version:

```
STATE: alarm_on
{
    if (counter_anyname.current == 3) sound_alarm();
}
```

(C) Next State

The C translator supplies the statement "goto (current_state)" at the bottom of every state that it codes. If `current_state` has been redefined and if the program reaches the bottom of the state, the `goto` statement will redirect the program toward a new state label. That is how the program is redirected into `state_alarm_on_loop` in this translator's version of `STATE: alarm_on`:

```c
static label current_state;
state_alarm_on:
current_state = state_alarm_on_loop;
{
    goto (current_state);
state_alarm_on_loop:
waitfor
}
goto (current_state);
}```
If the user wants to redefine \texttt{current\_state}, he may do so in the nonloop state, in which case the loop (\texttt{waitfor}) state will be bypassed:

```c
static label current_state;
state_alarm_on:
current_state = state_alarm_on_loop;
{
    current_state = state_alarm_off;
    goto (current_state);
    state_alarm_on_loop:
    waitfor
    {
    }
    goto (current_state);
}
state_alarm_off:
/* etc. */
```
The example above is the equivalent of this spreadsheet entry:

```
STATE: alarm_on
CONDITIONS: ENTER_STATE
NEXT_STATE: alarm_off
STATE: alarm_off
```

The following hybrid code also will produce the same result. No `break` is necessary, since the translator will place the C region above the `waitfor`.

```
STATE: alarm_on
{  current_state = state_alarm_off;
}
STATE: alarm_off
```

Or the user may redefine `current_state` in the `waitfor` statement itself, inside the loop state. The only way out of a `waitfor` statement is a `break`, so the translator must furnish a `break` whenever it converts a `NEXT_STATE` action into C (unless, as in the example above, the condition that triggered the `NEXT_STATE` action was `ENTER_STATE`, and consequently the program never entered the `waitfor` loop).

The following example uses `NEXT_STATE`:

```
STATE: alarm_on
CONDITIONS: KEYBOARD * *
ACTIONS: ALARM
  PROMPT "press space bar--alarm now disabled"
NEXT_STATE: alarm_off
STATE: alarm_off
CONDITIONS: KEYBOARD * *
ACTIONS: PROMPT "press space bar--alarm is activated"
NEXT_STATE: alarm_on
```

Here is the C version:

```c
static label current_state;
state_alarm_on:
current_state = state_alarm_on_loop;
{
goto (current_state);
state_alarm_on_loop:
waitfor
{
  keyboard_new_any_key && (keyboard_any_key == ' '):
  {
sound_alarm();
display_prompt("press space bar--alarm now disabled");
current_state = state_alarm_off;
break;
  }
}  
goto (current_state);
}
```

53-6
state_alarm_off:
current_state = state_alarm_off_loop;
{
    goto (current_state);
    state_alarm_off_loop:
    waitfor
    {
        keyboard_new_any_key && (keyboard_any_key == ' '):
        {
            display_prompt ("press space bar--alarm is activated");
            current_state = state_alarm_on;
            break;
        }
    }
    goto (current_state);
}

Various hybrid versions are possible. Here is one:

STATE: alarm on
CONDITIONS:
{
    keyboard_new_any_key && (keyboard_any_key == ' ')
}

ACTIONS:
{
    sound_alarm();
    display_prompt ("press space bar--alarm now disabled");
    current_state = state_alarm_off;
    break;
}

STATE: alarm off
CONDITIONS:
{
    keyboard_new_any_key && (keyboard_any_key == ' ')
}

ACTIONS:
{
    display_prompt ("press space bar--alarm is activated");
    current_state = state_alarm_on;
    break;
}

53.3 Conditions and Actions

When a condition is translated into C code by the INTERVIEW, the resulting expression is enclosed in braces at the top of a waitfor statement. The only exception to this rule is the ENTER_STATE condition—see Section 53.2(B), above.

The conditional expression is followed by a colon and then by the statement that constitutes the action to be taken when the condition is true. If more than one action is coded, braces must be used to form a statement block. See Figure 53-3.

Typing CONDITIONS: KEYBOARD " " on the spreadsheet is the equivalent of this C coding, placed inside of the braces that follow the reserved word waitfor:

keyboard_new_any_key && (keyboard_any_key == ' '):
{
    /* action-statements or routines go here */
}
(A) Multiple Condition Clauses

Following the semicolon that terminates the statement (or following the statement block), you may enter another condition clause. These clauses correspond to triggers on the Trigger menus or conditions-and-actions blocks inside a state on the Protocol Spreadsheet. Multiple condition clauses may be placed inside of one waitfor construction. (There is only one waitfor statement per state.)

Here is an example of a state with two "triggers":

```
STATE: keyboard_prompt
CONDITIONS: KEYBOARD "1"
  ACTIONS: ALARM
  PROMPT "You have pressed the 1 key."
CONDITIONS: KEYBOARD "2"
  ACTIONS: ALARM
  PROMPT "You have pressed the 2 key."
```

A version in C would have two condition clauses:

```
state_keyboard_prompt:
  waitfor
  {
    keyboard_new_any_key && (keyboard_any_key == '1'):
    {
      sound_alarm();
      display_prompt ("You have pressed the 1 key.");
    }
    keyboard_new_any_key && (keyboard_any_key == '2'):
    {
      sound_alarm();
      display_prompt ("You have pressed the 2 key.");
    }
  }
```

If you are mixing spreadsheet tokens with C, place condition clauses inside of STATE: blocks. Any C region at the top of a State block is placed above the automatic waitfor statement. You must therefore supply your own waitfor word, since a condition clause is syntactically valid only inside of a waitfor. An example follows.
A word of warning is in order. When your program executes this code, it will find itself stuck in a waitfor statement beneath the label state_keyboard_prompt. If you want to exit this waitfor, you must execute a break in a statement block in one of the condition clauses. Once you have broken outside of the waitfor, you may goto another state.

If you add softkey CONDITIONS, ACTIONS, or NEXT_STATE blocks to the state above, they will be placed inside a different waitfor statement, the one that is created automatically inside a state called state_keyboard_prompt_loop. See Section 53.2 (particularly Figure 53-2). What may look like a single state on the spreadsheet really will be two different states which never are active at the same time.

(B) Multiple Expressions

Expressions may be logically anded (&&) or ored (||) together inside a condition clause. Here is the spreadsheet version of a CONDITIONS block with two expressions:

```plaintext
CONDITIONS: KEYBOARD "2"
FLAG keyboard_disabled 0
ACTIONS: PROMPT "You have pressed the 2 key."
```

Inside the condition clause in C, the translator supplies a double ampersand (&&) to connect the keyboard expressions with the flag expression:

```c
keyboard_new_any_key && (keyboard_any_key == '2')
&& (flag_keyboard_disabled.current == 0):
{
  display_prompt ("You have pressed the 2 key.");
}
```
Inside a CONDITIONS block, the translator is able to and a softkey condition correctly with a C expression. Note that the user types the C expression without a terminating colon. The translator will supply one later:

```
CONDITIONS: KEYBOARD "2"
{   
flag_keyboard_disabled.current == 0
}
ACTIONS: PROMPT "You have pressed the 2 key."
```

The anding is also successful when the C expression is placed above the softkey condition inside the CONDITIONS block:

```
CONDITIONS:
{   
flag_keyboard_disabled.current == 0
}
KEYBOARD "2"
ACTIONS: PROMPT "You have pressed the 2 key."
```

If you want to insert a comment into a Conditions block, remember that the translator does not look inside of C regions (except to expand constants). It will take the comment and and it with the rest of the expressions in the Conditions block. Since a comment is not a C expression, the program will not compile: see Section 53.3(D). Note in the following example that a 1 has been inserted inside the C region along with the comment in order to make the code compile and in order to make the expression "true."

```
CONDITIONS:
{   
/* This comment will be anded with the keyboard expression. */ 1
}
KEYBOARD "2"
ACTIONS: PROMPT "You have pressed the 2 key."
```

(C) Event Variables

The translator converts most Conditions blocks on the Protocol Spreadsheet into two or more expressions linked by the logical and operator (&&). The keyboard condition in the examples above was typical: KEYBOARD "2" on the spreadsheet became a pair of expressions logically anded in C.

The first expression, keyboard_new_any_key, is an event variable. Event variables are very important in the INTERVIEW implementation of C, and the programmer should observe the following rules of thumb:

1. **An event variable usually is paired with a nonevent variable.** At the moment an event variable comes true in a waitfor construction, all nonevent (or "status") variables attached to that event variable are evaluated for truth or falsity. Whenever any keyboard key is struck, the event variable keyboard_new_any_key comes true. At that moment, the nonevent expression keyboard_any_key == '2' is evaluated to determine whether it is true or false.
2. A waitfor statement must include at least one event expression. A waitfor statement without an event variable will not compile. There must be some event that might transpire to cause the nonevent expressions to be evaluated.

3. An event variable may appear alone in an expression. It is possible (though unusual) to have an event expression that is not anded with a nonevent expression. When the translator converts CONDITIONS: DTE GOOD_BCC into C, for example, the resulting expression is this simple event variable:

   {evar_md_bcc_id;}

4. A nonevent variable also may appear alone. It also is possible (though the translator does not do this inside of waitfor statements) to have a nonevent expression that is not anded with an event expression—as long as there is an event expression somewhere in the waitfor construction. The following program will compile and work:

   {
       extern fast_event keyboard_new_any_key;
       extern volatile unsigned short keyboard_any_key;
   }

   STATE: keyboard_prompt
   CONDITIONS:
       {
           keyboard_new_any_key && (keyboard_any_key == '1')
       }
   ACTIONS: PROMPT "You have pressed the 1 key."
   CONDITIONS:
       {
           keyboard_any_key == '2'
       }
   ACTIONS: PROMPT "You have pressed the 2 key."

In this example, keyboard_any_key == '2' is not anded with an event variable. As a result, it is attached automatically to the event variable keyboard_new_any_key in the Conditions block above. If there had happened to be other event variables in the state, it would have been attached to them as well; so that when any event in the state came true, keyboard_any_key == '2' would be evaluated.

   NOTE: Other event variables in the state would cause
   keyboard_any_key to be evaluated, but would not necessarily
   cause it to be updated. Event variables are guaranteed to update
   only their associated nonevent variables. In the example above,
   keyboard_any_key is an associated nonevent variable for the event
   variable keyboard_new_any_key.

5. Two event variables may not be combined. Two event variables may never be combined in a condition clause, since two events never are simultaneous. Since all spreadsheet conditions have event variables associated with them—counter conditions have the counter_name_change event variable, for example—it might seem impossible to combine a counter with another
condition in a single CONDITIONS block. In fact, in the case of a few special combinable conditions—buffer-full, counter, flag, and E1A are examples—the translator will sometimes omit the event variable. When two or more combinable conditions are combined, the translator uses a first come, first served rule that is explained in Section 54.3, Programming Considerations.

(D) Evaluating Nonevent Expressions

Nonevent expressions are true if they have a nonzero value. In the following program, the "trigger" will sound the alarm when any keyboard key is struck because all of the nonevent expressions are nonzero:

```c
{ extern fast_event keyboard_new_any_key;
  } STATE: boolean
  CONDITIONS:
  { keyboard_new_any_key && 1 && 99 && 10003
  } ACTIONS: ALARM
```

This version never will sound the alarm, because one of the anded components is zero:

```c
{ extern fast_event keyboard_new_any_key;
  } STATE: boolean
  CONDITIONS:
  { keyboard_new_any_key && 1 && 0 && 10003
  } ACTIONS: ALARM
```

Relational expressions like `keyboard_any_key == '2'` and logical expressions connected by `&&` (like those above) and `||` are defined automatically to have the value 1 if true and 0 if false.

(E) Multiple Statements

Statements may be blocked together inside a condition clause. Here is the spreadsheet version of an ACTIONS block with two statements:

```
CONDITIONS: KEYBOARD "2"
ACTIONS: PROMPT "You have pressed the 2 key."
ALARM
```

The C version is a condition clause with two routines, `display_prompt` and `sound_alarm`, inside a block or compound statement:

```c
keyboard_new_any_key && (keyboard_any_key == '2'):
{ display_prompt ("You have pressed the 2 key.");
  sound_alarm();
}```
A hybrid version, part spreadsheet language and part C language, will work:

CONDITIONS: KEYBOARD "2"

ACTIONS: PROMPT "You have pressed the 2 key."

```c
{ sound_alarm();
}
```

The hybrid example as it stands will not allow you to declare routines and
variables, because the translator will place these declarations in a statement block
beneath the display prompt routine. For declarations, move the C region to the
top of the Actions block; or use double braces to open a new statement block
lower down, since declarations are legal following the left brace that introduces
any compound statement.

### 53.4 Example of Complete C Program

Some of the examples in the previous pages of this section were incomplete, in that
they included variables that were not declared, or they lacked a softkey STATE that
could generate a proper program main. The following is an extended example that
compiles and runs. It includes many of the pieces that formed the shorter examples
in this section. It is written for the Protocol Spreadsheet as completely as possible in
C. (See Section 52.3 on how to write a program completely in C.)

```c
{ extern fast_event keyboard_new_any_key;
extern volatile unsigned short keyboard_any_key;

main()
{
 static label current_state;
 state_alarm_on:
 current_state = state_alarm_on_loop;
 { goto (current_state);
  state_alarm_on_loop:

  waitfor
  {
   keyboard_new_any_key && (keyboard_any_key == ' '):
   { sound_alarm();
    display_prompt ("press space bar--alarm now disabled");
    current_state = state_alarm_off;
    break;
   }

   goto (current_state);
  }

  state_alarm_off:
  current_state = state_alarm_off_loop;
  { goto (current_state);
   state_alarm_off_loop:
   waitfor
   {
   keyboard_new_any_key && (keyboard_any_key == ' '):
```
{ 
    display_prompt ("press space bar--alarm is activated");
    current_state = state_alarm_on;
    break;
}
}
go to (current_state);
}
}
layer I_test_keyboard_alarm;

STATE: task_list
{
    layer I_test_keyboard_alarm();
}

53.5 Summary of C Regions

The translator removes the outer braces from a C region and places it into one of the six basic levels of source code shown in Figure 53-4.

(A) Declarations

Declare your variables and routines in a C region, delimited by curly braces { and }, at the top of your program or at the top of a Constants, Layer, Test, State, or Actions block. Declare a variable preceded by its type descriptors and followed by a semicolon, as in these examples:

{ 
    extern fast_event keyboard_new_key;
    extern fast_event keyboard_new_key;  
    extern fast_event fevar_time_of_day;
    short minutes;
}

We have not bothered to declare routines in most of the examples in the manual, since it is not necessary. In the absence of a declaration, the compiler assumes that the routine is external and that it returns an integer. In nearly all cases, this assumption works. In the few cases where a routine returns a long (get_65k_phys_addr is an example), it must be declared.

1. Automatic declaration. In cases where the translator declares a variable automatically, the user does not have to declare the variable himself. For example, a KEYBOARD condition, when entered via softkey, will declare the variable keyboard_new_key automatically for the entire program. When a variable has been declared twice in a program block, the program may not run. Instead, the compiler will put up a message such as the following: Error 110: keyboard_new_key redeclared. In software version 5.00 and in earlier software, the compiler flagged double declarations and aborted the compilation.

Sometimes it is difficult to keep track of the exact version of a variable that the translator is declaring. Some external variables have been improved for the use of C programmers, and we have documented the newer version in
our tables and in many of our examples. The translator may still use an older version of the variable.

In an earlier software release, for example, the variable `extern event keyboard_new_key` was speeded up and renamed `extern fast_event keyboard_new_key`. The translator still uses the older name to declare the variable.

The variable `keyboard_new_any_key` is a still more recent improved version of `keyboard_new_key`—improved in that it detects the striking of non-ASCII keys as well as the ASCII set. The translator never declares `keyboard_new_any_key` automatically.

Similarly, the translator uses an older version of `extern fast_event fevar_eia_changed`. The older version is `extern event evar_eia_changed`. In the earlier software, compiler error messages such as "keyboard_new_key redeclared" and "Variable fevar_eia_changed undeclared" will inform you what the translator is doing in each instance.

2. **Legal declaration.** Declarations are legal following the left brace that introduces any compound statement. Figure 53-4 shows that when the user opens a braced C region following a TEST:, STATE:, or ACTIONS: keyword, the translator removes the outer braces from the C region and plants the C code just inside the left brace at Level 2, 4, and 6 of the source code. Declarations therefore are valid at the top of these regions.

Declarations should be grouped at the top of any region, since they are not allowed in a statement block below an executable statement. This program will not compile, because the `sound_alarm` routine precedes a declaration:

```c
{ extern fast_event fevar_eia_changed;
 }

STATE: lead_changes
CONDITIONS:
{ fevar_eia_changed
}

ACTIONS:
{ sound_alarm();
  int lead_changes;
  lead_changes ++;
}
```

Declarations never are legal at Level 5 (Figure 53-4)—that is, preceding the colon in a condition clause inside a `waitfor` statement. Declarations always are legal at Level 1, since there are no executable statements at that level.

The set of variables listed as `extern` cannot be declared below Level 1. `Extern` has a specialized meaning at the task level or lower: it is used to "forward-declare" a variable without actually reserving storage space. The variable must be declared again (but not as `extern`) in the body of the task.
<table>
<thead>
<tr>
<th>Level 1</th>
<th>Level 2</th>
<th>Level 3</th>
<th>Level 4</th>
<th>Level 5</th>
<th>Level 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Braced C region at top of spreadsheet, following an OBJECT block, following program CONSTANTS: , following first LAYER: number, or following first layer CONSTANTS: inserted here</td>
<td>Braced C region following TEST:name inserted here</td>
<td>static label current_state; state name: current state = state name_loop;</td>
<td>goto (current_state); state name_loop: waitfor</td>
<td>Braced C region following STATE:name inserted here</td>
<td>Braced C region following CONDITIONS: inserted here</td>
</tr>
<tr>
<td>task</td>
<td>#pragma layer 1 main()</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>layer_1_test_name;</td>
<td>}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Braced C region following subsequent LAYER: number or subsequent layer CONSTANTS: inserted here</td>
<td>main()</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 53-4 The translator removes the outer braces from a C region and places it into one of six basic levels of source code. The "telescoping" of the braces indicates the scope of declarations. A variable or routine declared for Level 1 is declared for the remainder of Level 1 and across all levels to the right.
3. **Scope.** The "telescoping" of the braces in Figure 53-4 indicates the scope of declarations. A variable or routine declared for Level 1 is declared for the remainder of Level 1 and across all levels to the right. This means that a variable or routine declared at the top of Level 1 will be global throughout the program. You can force a declaration to the top of Level 1 by placing it in braces (1) at the top of the Protocol Spreadsheet; (2) before or after an OBJECT block; (3) inside a CONSTANTS block above the Layer level; (4) inside the first LAYER block on the spreadsheet; or (5) inside the CONSTANTS block in the first LAYER block.

Here is an example of a global declaration:

```c
{
    extern fast_event fevar_eia_changed;
}
```

LAYER: 1
TEST: leads
STATE: init
CONDITIONS:
```c
    fevar_eia_changed
```
ACTIONS: PROMPT "Status of a lead has changed."

A variable or routine declared at Level 1 (Figure 53-4) is declared for subsequent layers and tests, whether the subsequent layer is higher or lower. The concept of higher and lower layers is relevant to softkey entry on the Protocol Spreadsheet, but is not carried over into the source code. To the compiler, a TEST in Layer 2 and a TEST in Layer 3 are simply concurrent tasks. The task that is first in the program is compiled first. That is the only meaning of "higher" and "lower" to the compiler.

A variable or routine may have its scope limited to a particular Test, State, or Actions block. A variable or routine also may be redeclared at different levels. Given more than one valid declaration, the lower or nearer one applies.

4. **Initialization.** A variable must be of the static storage class to pass its value into a waitfor statement. Declarations at Level 1 of the source code (Figure 53-4) are always static, whether or not they are declared so. A variable that is initialized at Level 4 (Figure 53-4) must be declared as static by the programmer if the initialized value is to be used inside a waitfor.

(B) **Statements**

Executable statements may occur at four levels (Figure 53-4) in the source code: at Level 2 of the program main function, where the function is defined; at Levels 3 and 4, where the task main function is defined; and at Level 6, inside a waitfor statement. The programmer has no access to Level 3. To access Level
4. The programmer may open a C region just beneath the STATE: name identifier. He may access Level 6 by opening a braced C region below the ACTIONS: keyword.

Levels 1 and 2 are reserved for declarations. The program main function executes statements at Level 2 (see the bottom of Figure 53-4), but this function is accessible only to the translator.
54 Events

In Run mode, the user program in the INTERVIEW moves from program STATE to program STATE. In each state a set of conditions is tested, with one or more actions the result of a particular condition coming true.

In the INTERVIEW's implementation of C, a "state" is a special control structure called a waitfor clause that is placed in the program directly following a label named for the state. Program movement is controlled by goto statements that reference these labels.

Each waitfor clause defines a set of interrupts ("events") that it is waiting for. When a waitfor clause is active and an interrupt/event occurs that is defined in that clause, the entire clause is processed. All of the conditions in the clause are tested and appropriate actions (statements, operations, routines) are executed.

The waitfor clause is a mechanism designed specifically for the data-communications testing environment, in which the program must interact at high speed with a variety of unpredictable inputs.

54.1 Example of Event: fevar_time_of_day

In the waitfor clause in an earlier example (Section 52 of this volume), the condition was this:

\[ \text{fevar\_time\_of\_day} \&\& (\text{crnt\_time\_of\_day} == 1300) \]

Once every minute, the CPM sends an interrupt to the MPM. This interrupt takes the form of a fevar_time_of_day event.

If the program includes a fevar_time_of_day condition, the interrupt each minute will cause the variable crnt_time_of_day to be updated.

If the current state includes a fevar_time_of_day condition, the interrupt each minute will satisfy that condition. At the same time all other conditions in the clause, including non-event (that is, non-interrupt-driven) conditions such as crnt_time_of_day == 1300, will be tested.

The relationship between an event variable such as fevar_time_of_day and its associated nonevent variable (in this case, crnt_time_of_day) can be summarized as follows: the event variable anywhere in the program causes the nonevent variable to be updated each time the event occurs. The event variable in the currently active waitfor loop causes the nonevent condition to be tested each time the event occurs.
Figure 54-1 illustrates this relationship, as well as the relationship between an event and a nonassociated variable. The figure shows, for example, how an EIA event might cause the time-of-day variable to be checked but not updated; and how a time-of-day event might cause the EIA-status variable to be checked but not updated. "Event" in the figure means event variable, while "variable" means nonevent variable.

```
+-------+            +-------+            +-------+
| event A occurs |                  | event B occurs |
| variable A     |                  | variable B    |
| updated;      |                  | updated;     |
| variable B     |                  | variable A    |
| not updated    |                  | not updated  |
```

Figure 54-1: This figure is meant to show the effect of event A on its associated variable (variable A) as well as its effect on a nonassociated variable (variable B).

### 54.2 Various Origins of `waitfor` Events

Interrupts sent to the MPM from the CPM include `fevar_time_of_day` and `keyboard_new_key`. Interrupts sent to the MPM by the SCC (Serial Communications Controller) chip in the FEB include `fevar_rcvd_char_td`, `fevar_sd_bcc_rd`, and `fevar_eia_changed`. Some interrupts are sent to the user program by the protocol state machines in the layer packages. Examples are `dce_frame` and `dte_packet`.

Interrupts also can be generated by the program itself. The program sends an interrupt in the form of a "signal." `counter_name_change` and `flag_name_change` are events that are signaled by the program itself, since the program is in charge of all counter and flag increments, decrements, and sets.
54.3 Programming Considerations

By itself in a waitfor clause, crnt_time_of_day == 1300 never can be true, since only
terupts/events cause the nonevent conditions in the clause to be processed. On the
other hand, counter_name_change && flag_name_change never can return true, since
two events cannot occur simultaneously.

Because two events never are simultaneous, the programmer (and the built-in
translator) has a decision to make whenever two nonevent conditions, such as
counter_name.current == 3 and flag_name.current == 5, are anded together. If the
programmer writes counter_name_change && (counter_name.current == 3) &&
(flag_name.current == 5), the condition may be true when counter_name.current
transitions to 3 but it never will be true when flag_name.current transitions to 5,
since there is no interrupt to cause the condition to be checked at that moment. If
an interrupt (flag_name_change) is tied to flag_name.current, then
counter_name.current transitioning to 3 will not be detected.

When the user combines a flag condition with a counter condition on a single Trigger
Setup menu, the translator solves the dilemma of which event to “wait for” by
generating a two-pronged waitfor condition that is approximately the following:

(counter_name_change && (counter_name.current == 3) &&
(flag_name.current == 5)) || (flag_name_change &&
(counter_name.current == 3) && (flag_name.current == 5));

On the Protocol Spreadsheet, the translator simply attaches the appropriate event
variable to the first softkey condition listed. If the user enters

CONDITIONS: COUNTER name EQ 3
FLAG name 101

the translator converts this to (counter_name_change && (counter_name.current ==
3) && (flag_name.current == 5)). The user is then free to repeat the combined
condition, reversing the order of the elements (and therefore invoking the
flag_name_change interrupt) the second time around.

NOTE: The examples in Section 54.3 above are somewhat
simplified. The actual translator versions are made more
complicated by the inclusion of counter_name.old and
flag_name.old variables that are explained in Section 62.
55 Receiving and Transmitting Data

As the INTERVIEW monitors the data source (line or disk), it signals the arrival of each character by an event variable (fevar_rcvd_char_rd or fevar_rcvd_char_td) and it stores each character momentarily in a variable (rcvd_char_rd or rcvd_char_td) accessible by the user. Data can be taken from the line in this form and copied into memory or into an interlayer message buffer. BOP-framed data is copied automatically into an interlayer ("IL") buffer.

The user transmits data from the INTERVIEW by creating a transmit-data structure and then referencing the structure in an il_transmit routine. Or the user may copy the data into an interlayer buffer (or simply reference the data in the buffer) and then call out the buffer in an il_il_transmit routine.

The IL buffers have several advantages as a storage medium for data. First, they are reusable. They are allocated dynamically and erased automatically unless the user takes steps to maintain them. Without these reusable buffers, data in Run mode would quickly eat up all of the memory in the unit.

Second, IL buffers support linked lists. There are routines that will start a list, insert data at the top of a list, and append data to the bottom of a list. Linked lists are well suited to layered-protocol transmissions, where the transmit string is built incrementally as the transmission moves down the layers.

55.1 Locating Data in an IL Buffer

When a BOP frame is placed automatically in an IL buffer, a data primitive is created automatically and the event variable m_lo_ph_prmtv is signaled. The segment number of the IL buffer is recorded in the variable m_lo_ph_il_buff. The offset from the start of the buffer to the start of the data is recorded in the variable m_lo_ph_sdu_offset. This offset is always 32 bytes. What is considered data at higher layers may have a larger offset, since each layer's data begins farther into the frame. See Figure 55-1 for an illustration of a gradually shrinking "service data unit" (SDU) and a gradually expanding SDU offset.

The first memory location in the first of the sixteen IL buffers is 03a80000, the first location in the second buffer is 03b00000, the first location in buffer #3 is 03b80000, buffer #4 starts at 03c00000, and so on through 04200000. Each of these addresses is 32 bits. The high-order 16 bits is the 80286 segment number (03a8, 03b0, 03b8, 03c0, etc., through 0420). This is the number that the software passes around when
it wants to identify an IL buffer, simply because 16 bits are easier and faster to pass around than 32 bits, the low-order 16 of which are always zero when we are discussing the starting location of each buffer.

When we want to look at data in the buffer, we need to reference not a 16-bit segment number but a 32-bit address. So we cast the segment number (always a short, 16 bits) into a long and move the number over to its high-order position, sixteen bits to the left. We add 32 to the number to bypass the header information for the buffer. Then we cast the new long as a character pointer. Here, for example, is m_lo_ph_il_buff converted into a pointer to the first byte in a frame:

\[
\text{char } \ast \text{ m_frame_ptr;} \\
\text{m_frame_ptr = (void*)(((long)m_lo_ph_il_buff } < 16) + 32); \\
\]

Figure 55-1 When an IL buffer is passed upward, the data offset changes and the data length changes, but the buffer itself does not change.
55.2 Monitor Path vs. Receive Path

The variables $m_{lo\_ph\_prmtv}$, $m_{lo\_ph\_il\_buff}$, and $m_{lo\_ph\_sdru\_offset}$ are part of a set of monitor services that handle IL buffers in both monitor and emulate modes. These variables are updated for data on either data lead. The layer packages use these variables to generate the protocol traces. The translator uses them to implement spreadsheet condition-tokens such as PH_TD_DATA IND and DTE INFO.

Another set of variables are maintained in emulate mode and are updated for data on the receive side only. These variables have names that reveal their obvious relationship to the monitor set: $lo\_ph\_prmtv$, $lo\_ph\_il\_buff$, $lo\_ph\_sdru$, etc. These receive-side variables are used by the translator to implement spreadsheet condition-tokens such as PH_DATA IND and RCV INFO.

Whenever a BOP frame is placed automatically in an IL buffer during an emulate run, events $m_{lo\_ph\_prmtv}$ and $lo\_ph\_prmtv$ both are signaled. The segment number of the same IL buffer is recorded in two variables, $m_{lo\_ph\_il\_buff}$ and $lo\_ph\_il\_buff$.

55.3 Passing a Buffer Upwards

Layer 1 stores data in IL buffers and passes these buffers to Layer 2 automatically, as we have seen. If a Layer 2 personality package is loaded in from the Layer Setup screen, the second data byte in the buffer (the 34th byte overall) is checked to determine the frame type. If the contents of the buffer is an Info frame, a data primitive is created automatically and the event variable $m_{lo\_dl\_prmtv}$ is signaled. The segment number of the IL buffer is recorded in the variable $m_{lo\_dl\_il\_buff}$. This is the same segment number that was stored previously in $m_{lo\_ph\_il\_buff}$.

The offset from the start of the buffer to the start of the data—Layer 2 or data link (DL) data—is recorded in the variable $m_{lo\_dl\_sdru\_offset}$. This offset is always 34 in MOD 8. This number represents the 32-byte buffer header plus a 2-byte frame header that is of no interest to Layer 3, which will use $m_{lo\_dl\_il\_buff}$ and $m_{lo\_dl\_sdru\_offset}$ to construct its packet trace.

The size of the data component in the buffer is stored in the variable $m_{lo\_dl\_sdru\_size}$. This number will be 2 bytes smaller than the variable $m_{lo\_ph\_sdru\_size}$.

If no layer packages are loaded, none of the buffer-handling services are provided automatically at Layer 2 or higher. The programmer can provide the services "manually" as indicated above.

If layer packages are loaded, monitor-path variables (those variables whose names begin with $m_\$) are updated automatically in order to drive the protocol traces. Receive-path variables such as $lo\_dl\_prmtv$, $lo\_dl\_il\_buff$, and $lo\_dl\_sdru$ are...
generated as needed by GIVE_DATA actions entered by the user on the Protocol Spreadsheet. Otherwise it is up to the C programmer to maintain these variables. For example, the user passing an IL buffer up to Layer 3 might write this code:

```c
lo_dUil_buff = lo_ph_il_buff;
lo_d_u_sdu = (lo_ph_sdu + 2);
pdu_ptr->data_length = (pdu_ptr->data_length - 2);
signal (lo_dl_prmv);
```

The same updates of variables and the same signal would be generated if the user called a `send_dl_prmv_above` routine, as follows:

```c
_set_maint_buf_bit (lo_ph_il_buff, &il2_relay_baton);
send_dl_prmv_above (lo_ph_il_buff, il2_relay_baton, lo_ph_sdu + 2, pdu_ptr->data_length - 2, 0x45);
```

The `send_dl_prmv_above` routine requires an SDU size value. There is no receive-path variable (equivalent to `m_lo_ph_sdu_size` on the monitor path) that maintains this value. Determine the SDU size from the `data_length` variable located in the `pdu`-structure. In the examples above, `pdu_ptr` is a structure pointer. The SDU size, therefore, is referenced as `pdu_ptr->data_length`. Refer to Section 63.1 for more information on the `pdu` structure.

**NOTE:** Do not use `m_lo_ph_sdu_size` for receive-path routines such as `send_dl_prmv_above`. It is not updated reliably at the same moment that other receive-path variables are updated.

0x45 is the code for a DL_DATA IND primitive.

### 55.4 Layer 1 Transmit

Line transmissions are accomplished through L1 transmit routines. Shown below is a program that ends in an `ll_il_transmit` routine. This routine puts the data contents (the service data unit or “SDU,” not the buffer header) of an IL buffer out onto the line.

Note that there is a set of routines leading up to the transmit routine. This set of routines is necessary to get a buffer, to start a linked list inside the buffer, and finally to insert several chunks of data into the list before it is transmitted.

```c
{
    unsigned short bufnum;
    unsigned short baton;
    unsigned short list_hd_offset;
    static unsigned char data[] = "(FOX)";
    static unsigned char pkt_hdr[3] = {0x10, 0x07, 0};
    static unsigned char frm_hdr[2] = {0x03, 0};
    int length;
    unsigned short transmit_tag = 1;
}
```
STATE: fo
CONDITIONS: 'KEYBOARD' "
ACTIONS:
{
    _get_il_msg_buff(bufnum, baton);
    _start_il_buff_list(bufnum, &list_hd_offset);
    length = sizeof(data) - 1;
    _insert_il_buff_list_cnt(bufnum, list_hd_offset, &data[0], length);
    _insert_il_buff_list_cnt(bufnum, list_hd_offset, &pkt_hdr[0], 3);
    _insert_il_buff_list_cnt(bufnum, list_hd_offset, &frm_hdr[0], 2);
    ll_il_transmit(bufnum, baton, USf_hd, transmit_tag);
}

The transmit string will look like this on the INTERVIEW's data display:
% THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG 0123456789

(A) Segment Number

The il_il_transmit routine required four arguments as input. First, it required
the segment number of the IL buffer that was intended to be transmitted. This
number was supplied by the _get_il_msg_buff routine, and we called the number
bufnum. There are a total of sixteen numbered IL buffers available to the
program.

(B) Relay Baton

The second argument was the number of the "relay baton" or "maintain bit."
This relay baton was supplied by the _get_il_msg_buff routine, and we called the variable that held the number baton. A relay baton is passed down automatically
with every send or transmit routine and serves to hold the buffer until it has
been processed by the next layer (or transmitted by Layer 1). Then the baton is
freed.

There are sixteen numbered relay batons available for each IL buffer. At the
moment that all sixteen batons (or maintain bits) are free, the buffer is returned
automatically to the pool of free IL buffers and its contents are no longer
available to the program.

In many applications—X.25 Layer 2 and Layer 3 personality packages, for
example—an extra maintain bit is reserved (via the _set_maint_buff_bit routine)
each time a buffer is sent down. This extra maintain bit is held onto in case a
frame or packet must be resent, and is not freed (in a free_il_msg_buff
routine) until the outstanding frame or packet has been acknowledged.

(C) List-Header Offset

In addition to buffer number and baton number, the il_il_transmit routine also
requires as input the offset from the start of the buffer to the linked-list header.
This offset is supplied at the moment the linked list is started by the
_start_il_buff_list routine. In the program above we called this offset
list_hd_offset.
When an IL buffer is passed downward, the data-start offset gives the location of the list header. This list header and the various pieces of the transmission (the list nodes) are threaded together.
Figure 55-2 illustrates how the list header ties the linked list together by identifying the offsets to the first and last nodes. A list node is created by each _insert_il_buff_list_cnt or _append_il_buff_list_cnt routine. The program in Section 55.4 has three _insert_il_buff_list_cnt routines. The IL buffer that is transmitted therefore has three list nodes.

(D) Transmit Tag

The fourth argument in the ll_il_transmit routine is a "transmit tag" that determines the type of BCC to be appended to the transmission. This variable is stored in the 32-byte header of each IL buffer. Refer to the structure il_buffer in the table of OSI structures, Table 63-1.

A transmit tag of 1 means a good BCC and 2 means a bad BCC. 3 causes an aborted transmission.

55.5 Passing a Buffer Between Tasks

At this point we need to modify our ll_il_transmit program to allow different layers—which are simply separate concurrent tasks in the programming architecture—to contribute list nodes to the IL buffer intended for transmission. The resulting transmit string will be the same as before, but three different tasks will have contributed data components to the transmitted buffer. In our new program, a Layer 4 task will provide the fox message, Layer 3 will provide the _insert_il_buff_list_cnt routine that references the 3-byte packet header, and Layer 2 will provide the insert routine that references the 2-byte frame header.

How do the separate layer tasks communicate with each other so that the right buffer is accepted at the moment it is handed down? They relay information in the same way that tasks always communicate, by signals that are detected throughout the program as event variables. When Layer 4 sends an IL buffer down in a send_n_prmtv_below routine, an event variable at Layer 3 (up_n_prmtv, not shown in the program below but implied nevertheless in the N_DATA REQ condition) comes true and at the same time updates the variables up_n_il_buff and up_n_sdu. Layer 3 can use these variables to identify the new IL buffer and to determine the offset to the list header in that buffer. With this information, Layer 3 can insert its own list node into the buffer before passing it down to layer 2.

Here is the program, followed by a few explanatory comments:

```c

unsigned short bufnum;
unsigned short l4_baton;
unsigned short l3_baton;
unsigned short l2_baton;
unsigned short list_hd_offset;
static unsigned char data[] = "(FOX)";
static unsigned char pkt_hdr[3] = {0x10,0x07,0};
```

55-7
static unsigned char frm_hdr[2] = {0x03,0};
int length;
extern volatile unsigned short up_n_il_buff;
extern volatile unsigned short up_dl_il_buff;
extern volatile unsigned short up_n_sdu;
extern volatile unsigned short up_dl_sdu;
}

LAYER: 4
STATE: fox
CONDITIONS: KEYBOARD * *
ACTIONS:
{
_set_il_msg_buff(&bufnum,&14_baton);
_start_il_buff_list(bufnum,&list_hd_offset);
length = sizeof(data) -1;
_insert_il_buff_list_cnt(bufnum,list_hd_offset,&data[0],length);
send_n_prmtv_below(bufnum,14_baton,list_hd_offset,0,0x64,0);
}

LAYER: 3
STATE: packet_header
CONDITIONS: N_DATA REQ
ACTIONS:
{
_insert_il_buff_list_cnt(up_n_il_buff,up_n_sdu,&pkt_hdr[0],3);
_set_maint_buff_bit(up_n_il_buff,&13_baton);
send_dl_prmtv_below(up_n_il_buff,13_baton,up_n_sdu,0,0x44,0);
}

LAYER: 2
STATE: frame_header
CONDITIONS: DL_DATA REQ
ACTIONS:
{
_insert_il_buff_list_cnt(up_dl_il_buff,up_dl_sdu,&frm_hdr[0],2);
_set_maint_buff_bit(up_dl_il_buff,&12_baton);
send_ph_prmtv_below(up_dl_il_buff,12_baton,up_dl_sdu,0,0x24,0);
}

In the send-primitive routines, the hex values 64, 44, and 24 identify the primitives as data requests. See, for example, the values of up_n_prmtv_code in Table 63-4.

Note that there is no longer an ll_il_transmit routine in the program. When Layer 2 executes a send_ph_prmtv_below routine, Layer 1 handles the transmit function automatically.

The send_ph_prmtv_below routine does not have a transmit-tag argument that allows us to specify the BCC. Since the ll_il_transmit routine, which has a transmit-tag input, is being handled automatically, it is not immediately clear how you would send the transmit string with a bad BCC. Here is one way. Instead of the send_ph_prmtv_below routine at Layer 2, use the ll_il_transmit routine as follows:

ll_il_transmit(up_dl_il_buff,12_baton,up_dl_sdu, 2);

The 2 in the argument represents the transmit tag for a bad BCC.
If it seems strange to be using an `Il_il_transmit` routine at Layer 2, remember that none of the variables or routines is really layer-specific. In C, layers are simply concurrent tasks.

A "realistic" implementation of this program might be made somewhat more complicated by two additional elements. One or more `_open_space_in_il_buff` routines might be used so that, as far as possible, text data could be copied into the buffer where it would then be erased when the buffer was freed. (One of the advantages of IL buffers is that the space inside them can be recycled.)

Another complication is that for the same transmission, more than one linked list might be started in a single buffer. The example under the `_insert_il_buff_list_cnt` routine in Section 63.3(A) shows Layer 2 accepting a buffer from Layer 3 and starting a new linked list. This allows Layer 3 to reconstruct its original linked list in case a packet-resend is needed.

### 55.6 Sample Transmit Program: Sync or Async Echo

This application monitors incoming data for text strings bounded by `~` and `;`. It copies these strings into an IL buffer and then echoes them back out onto the line, preceded by two ASCII sync characters. The program will work in most data formats as long as ASCII `~` and `;` are included.

The program may be modified for EBCDIC `~`, `;`, `;`, and `;`. Use received-character variables `fevar_rcvd_char_rd` and `rcvd_char_rd` for data received on RD.

```c
{ 
    extern fast_event fevar_rcvd_char_td;
    extern volatile unsigned short rcvd_char_td;
    unsigned short number, length;
    unsigned short il_buffer_number, relay_baton, data_start_offset;
    unsigned char echo_string[100] = {'~', ';'};
}

STATE: look_for_stx
CONDITIONS:
{ 
    fevar_rcvd_char_td && rcvd_char_td == '~'
} 
ACTIONS:
{ 
    number = 2;
    echo_string[number] = rcvd_char_td;
    number++;
} 
NEXT_STATE: construct_echo_string

STATE: construct_echo_string
CONDITIONS:
{ 
    fevar_rcvd_char_td
} 
ACTIONS:
{ 
    echo_string[number] = rcvd_char_td;
}
```
number++;  
if ((rcvd_char_td == 'e') || (rcvd_char_td == 'E'))  
{  
    length = number;  
}  
}  
CONDITIONS: RECEIVE GOOD_BCC  
NEXT_STATE: transmit_echo_string  
STATE: transmit_echo_string  
CONDITIONS: 'ENTER_STATE'  
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, echo_string, length);  
    _il_transmit(il_buffer_number, relay_baton, data_start_offset, 1);  
}  
NEXT_STATE: look_for_stx  

55.7 Sample Transmit Program: BOP Echo

When Format: **BOP** is selected on the Line Setup screen, every frame that is received at the line interface is placed in an IL buffer and passed up to Layer 2. This sample program makes a pointer to the I-field in the most recent IL buffer received at Layer 2, and then it echoes the data back out in the C equivalent of a SEND INFO action. If you try this program, be sure to load the X.25 or SDLC package at Layer 2.

```c
    {  
        char * data_ptr;  
        extern volatile unsigned short rcvd_frame_buff_seg;  
        extern volatile unsigned short rcvd_frame_sdu_offset;  
        extern volatile unsigned short rcvd_frame_sdu_size;  
        struct send_frame_structure  
        {  
            unsigned char addr_type;  
            unsigned char frame_type;  
            unsigned char nr_type;  
            unsigned char ns_type;  
            unsigned char pf_type;  
            unsigned char bcc_type;  
            unsigned char addr_value;  
            unsigned char ctrl_byte;  
            unsigned char nr_value;  
            unsigned char ns_value;  
        };  
        struct send_frame_structure frame;  
        unsigned short number, baton, offset;  
    }  
LAYER: 2  
STATE: echo  
CONDITIONS: RCV INFO  
ACTIONS:  
{  
    data_ptr = (void *)((long)rcvd_frame_buff_seg << 16) + rcvd_frame_sdu_offset;  
    _get_il_msg_buff(&number, &baton);  
    _start_il_buff_list(number, &offset);  
    _insert_il_buff_list_cnt(number, offset, data_ptr + 2, rcvd_frame_sdu_size - 2);  
    frame.bcc_type = 1;  
    send_frame(number, baton, offset, &frame);  
}  
```
** Protocol Spreadsheet **

** LAYER: 1 **
** TEST: bsc_one **
(static label prev_state;
  STATE: polling
  CONDITIONS: RECEIVE ONE_OF "\^{\textbf{\$\$}}"
  ACTIONS: SEND "\^{\textbf{\$\$}}" GOOD_BCC
  \{prev_state = state_polling;\}
  NEXT_STATE: ack0
  STATE: ack0
  CONDITIONS: RECEIVE ONE_OF "\^{\textbf{\$\$}}"
  ACTIONS: SEND "\^{\textbf{\$\$}}" GOOD_BCC
\{current_state = prev_state;
  break;
\}

** Figure 56-1 ** Using C to return to the previous state.
56 C Basics

C programming language as implemented in the INTERVIEW 7000 Series is based on the current ANSI recommendations. It contains several extensions to the language which enhance its utility in protocol testing, notably multi-tasking.

C is intended as an aid to INTERVIEW users who have advanced programming knowledge. A sophisticated programming tool, C can be applied to testing requirements which are not met by Protocol Spreadsheet selections. C is useful, for instance, in the analysis and “intelligent” manipulation of variable data strings anticipated within a complex protocol. Additional applications of C are the creation of customized protocol and program trace displays.

Figure 56-1 provides a means of returning to whatever state was the former state, without you the programmer knowing which state was previously active. This “go to previous state” function is not a standard spreadsheet feature. The example employs Bisync protocol to demonstrate the usefulness of this capability. The test begins in a state called polling. Here, an ACK1 is sent whenever the end of any received data is encountered, and the test passes to the state called ack0. This time when the end of received data is encountered, an ACK0 is sent, and the test returns to whatever state it was in formerly.

The first C region is the declaration of the variable prev_state, which allows the variable to be used anywhere within the test. In the second C region, the variable prev_state is initialized to the name of the active state. The third C region shows the transition of the test to the previously active state. Depending on the contents of the prev_state variable, the former state could be one of any number of states. This capability means that, as the programmer expands the simple test, the state ack0 can be used again and again as a utility state from which the test returns to the former state, removing the need for repetitive spreadsheet entry.

56.1 Notable Variations in C

The AR version of C varies in certain respects from the ANSI standard. Notable exceptions to the standard are outlined below. A full set of implementation-defined variations appear in Appendix K.

(A) Reserved Words

The following two reserved words, in addition to those covered in the ANSI standard, are included in C:

    task
    waitfor
**B** Predeclared identifiers

The following type identifiers are always predeclared. They are not defined in any `#include` files, nor are their definitions required in any program. Thus they are part of the INTERVIEW C lexicon, even though they are not reserved words and therefore do not appear in the language summary in Appendix K.

```
  event
  fast_event
  label
```

**C** Floating Point Notation

Since Floating Point Notation is not required in the protocol testing environment and since corresponding calculations could degrade processing speed, floating point notation is omitted from the AR implementation of C. Fixed point calculations, however, are performed.

**D** Values Returned from C Functions

Functions declared within AR's implementation of C may only return values for data types which are 1, 2, or 4 bytes long. Consequently, a function cannot legally return most structure or union types.

### 56.2 Editing a C Program

Entries in C are made on the Protocol Spreadsheet, accessed from the Main Program screen. All editing functions available on the spreadsheet can be applied to C coding. Refer to Section 26 for a description of these editing functions.

### 56.3 Error Reporting in C

Most syntax errors made on the Protocol Spreadsheet are indicated by strike-through of the text where the error occurs. This facilitates correction of entries as you create a test.

Errors which appear in C coding are not indicated by the editor. However, when the program is compiled (when you press ~), the errors will be noted. If there are errors in the program, the INTERVIEW will automatically revert to the Protocol Spreadsheet rather than run the program.

**A** Locating Errors

The cursor is automatically positioned near the first error when the INTERVIEW reverts to the Protocol Spreadsheet. A diagnostic message about the error will be displayed at the top (second line) of the screen. Errors pertaining to the
general syntax of the spreadsheet are explained in text. Errors noted by the C
pre-processor or compiler are displayed as numbers, with explanatory text if the
filename sys/error_text is accessible at the moment on a disk. (The file should
always be accessible in units with hard disks.) These numbered messages are
listed in Appendix A3.

Press GO-ERR again to move down through the spreadsheet to the next error.
When you press GO-ERR and there are no more errors, the message “No More
Errors” will be displayed.

56.4 Preprocessor Directives

The INTERVIEW supports preprocessor directives #define and #include. The full set
of ANSI preprocessor directives are supported on the INTERVIEW. Included among
these directives are #if, #else, #ifdef, #ifndef, and #undef. (Refer to the ANSI
Recommendation for a discussion of these directives.) Implementation-defined
#pragmas are also preprocessor directives. #pragma object and #pragma hook are
two of the AR #pragmas. As the name implies, preprocessor directives are processed
before the program in which they appear is compiled.

Preprocessor directives are easy to recognize, since they are always preceded by a
pound sign (#). Spaces are significant to the meaning of the directives, since other
delimiters are generally not used. Note also that a semi-colon cannot be used to
terminate a preprocessor directive. Instead, a directive is terminated by a hard
Carriage Return or some indicator of line continuation. Press \ to terminate the
directive (no indication of the Return will appear on the screen). Type \ (backslash)
and press \ at the end of the line on the screen to indicate that the directive
continues on the next line. You may also allow text to wrap to the next line by
continuing to type. (Wrapped lines are indicated on the screen by the highlighted
symbol •.)

(A) #define

The #define directive gives you the convenience of replacing frequently
referred items with a text string of any length.

1. Placement. A #define directive may be placed at the beginning of a logical
line anywhere in a legal C region. The eight valid positions for C regions on
the Protocol Spreadsheet are shown in Figure 53-4. The #define directive
may also be placed in a separate #include file. Use the #include directive as
explained in (B) to invoke the file and make the macro-substitutions it
indicates in your main program file.
2. **Format.** The directive follows this format:

```
#define identifier string
```

For example, if you enter the following line of code,

```
#define message The quick brown fox ....12345
```

the identifier `message` (wherever it appears exactly as written in the file being acted upon) is replaced in subsequent lines of code by the string `The quick brown fox ....12345`. The replacement, the macro-substitution, is performed before the code is compiled. When you enter the `#define` directive, leave a space between the directive (`#define`) and the identifier. There should be no spaces in the identifier. The space following the identifier indicates that the next ASCII character (or blank) starts the replacement string. Spaces are allowed and are considered part of the string. Terminate the string (and the directive) as described at the beginning of this sub-section.

3. **Nesting.** `#define` substitutions may be nested. Of course, the nested replacements must be described by a `#define` directive which precedes the `#define` for the replacement text which contains them.

There is one exception to nesting identifiers—the macro substitution will not be performed when the identifier occurs in a string. In the example below, the programmer tries to nest `MAXTRIES` within the definition of `MESSAGE`:

```
#define MAXTRIES 3
#define MESSAGE "Maximum retransmissions is MAXTRIES."
```

A call to `displayf(MESSAGE);` causes the following to be displayed:

`Maximum retransmissions is MAXTRIES.`

This is certainly not what the programmer intended.

(B) **#include**

`#include` files, when invoked in a program, are read into the program file before the program is compiled. As a result, your program has access to commonly used items such as subroutines (input/output and string operations, for example), global variables, constants, and structures without your having to enter or modify the required code repeatedly.

1. **Format.** The format for the directive is as follows:

```
#include <filename>
```

or

```
#include "filename"
```
#include files follow standard naming conventions. See Section 13.2(E). As an added convention, the suffix .h is appended to the end of the name (as in the filename stdio.h).

2. Search rules for #include files. The delimiters you use to surround the filename determine how the INTERVIEW searches its filing system for the file.

• The <> delimiters are intended for files which are supplied by AR. When these delimiters are used, the following directories—and only the following directories—are searched, in the order given:

1. /sys/include on current drive (indicated on File Maintenance screen)
2. The directory named as the current directory on the File Maintenance screen (provided that the current directory is not the root directory for FD1, FD2, or hard disk)
3. /usr/include on current drive (indicated on File Maintenance screen)
4. FD1/sys/include
5. FD2/sys/include
6. HRD/sys/include
7. FD1/usr/include
8. FD2/usr/include
9. HRD/usr/include

NOTE: The directory names are given in the format which the INTERVIEW interprets as the absolute path from the root directory of the disk named before the first slash. So HRD/sys/include means /sys/include on the hard disk.

• The " " delimiters are intended for user-created files. The same directories are searched for the filename, but they are searched in the following order:

1. The directory named as the current directory on the File Maintenance screen (provided that the current directory is not the root directory for FD1, FD2, or hard disk)
2. /usr/include on current drive (indicated on File Maintenance screen)
3. /sys/include on current drive (indicated on File Maintenance screen)
4. FD1/usr/include
5. FD2/usr/include
6. HRD/usr/include
7. FD1/sys/include
8. FD2/sys/include
9. HRD/sys/include
If you have used the same filename for an include file in more than one directory, the file which is actually read in as a result of an include directive will be from the first directory searched which contains that filename. The delimiters you use, then, can make a difference in the file selected for inclusion.

The filename enclosed in <> or " " delimiters may be a relative pathname. The highest directory in the pathname must reside in the current directory or in one of the include directories. In response to an include "disk_iol/stdio.h" directive, for example, the INTERVIEW first looks for a disk_io subdirectory in the current directory on the File Maintenance screen and then for an stdio.h file in that subdirectory. If the file is not found, the search for the relative pathname continues according to the sequence designated for " " delimiters.

If the file is not located in any of these directories, an error message is returned to the operator.

(C) #pragma object

Use the #pragma object directive to access the compiled routine definitions in a linkable-object file. The OBJECT block-identifier discussed in Section 24.4 may also be used for this purpose. (Also see Section 13.3(P) on creating a linkable-object file—displayed as type LOBJ in the directory listings on the File Maintenance screen).

1. Placement. Place the #pragma object directive inside any legal C region on the Protocol Spreadsheet. Except for those containing the static attribute, routine definitions from an LOBJ file always have global scope. It makes sense, therefore, to position the directive at the top of your spreadsheet program along with other global declarations and definitions.

2. Format. The format for the #pragma object directive is as follows:

   #pragma object "filename.o"

A #pragma object directive references only one LOBJ filename, but you may include as many directives as you wish.

The relative or absolute pathname of the linkable-object file is enclosed in quotation marks.

3. Search rules for linkable-object files. As your spreadsheet program compiles, the INTERVIEW's filing system is searched for the linkable-object files referenced in #pragma object directives.

   • If the referenced LOBJ filename begins with FD1/, FD2/, or HRD/, the INTERVIEW interprets it as the absolute pathname and makes only that one search.

   • Pathnames beginning with a / indicate that the root directory on each drive should be the beginning point of the search. The drives are searched in the following order: current drive, FD1, FD2, and HRD.
Otherwise, the name may be a one-word filename, or a relative pathname which includes the directories leading to the file. The highest directory in a relative pathname must reside in the current directory or in one of the /lib subdirectories. The following directories—and only the following directories—are searched, in the order given:

1. current directory on the current drive (indicated on the File Maintenance screen)
2. /usr/lib on the current drive
3. /sys/lib on the current drive
4. FD1/usr/lib
5. FD2/usr/lib
6. HRD/usr/lib
7. FD1/sys/lib
8. FD2/sys/lib
9. HRD/sys/lib

If the pathname is not located in any of these directories, the program will not compile and an error message will be returned to the operator.

4. How #pragma object works. When the source of code for the Compile command is FILE, the LOBJ which results usually defines user-created routines. These routine definitions may be "linked," or combined, as needed with your spreadsheet program. This means that routines called within your active program do not always have to be defined on the Protocol Spreadsheet or in #include files.

NOTE: An LOBJ file may also contain #pragma hook directives. See Section (D) below. If a #pragma object directive references an LOBJ file which contains #pragma hook directives, the "hooks" within that file are ignored. Since Compile always generates #pragma hooks, use the OBJECT block-identifier to reference the resulting LOBJ file.

(a) Referenced linkable-object files searched for routine definitions. If a spreadsheet program calls a routine for which no definition is provided, the LOBJ files referenced in #pragma object directives are searched in the order in which they appear on the Protocol Spreadsheet. If a routine is defined in more than one referenced LOBJ file, the definition in the first LOBJ file listed on the Protocol Spreadsheet will be used. If the routine definition is not found in the spreadsheet program or in any referenced linkable-object file, the compilation will abort. When you go to the Protocol Spreadsheet and look for error messages, the routine name will appear as an unresolved reference.
(b) **Compiled routine definition combined with compiled spreadsheet.** When the routine's definition is located, the compiled code is copied from the LOBJ file and combined with the compiled code of the spreadsheet program.

Routine definitions in an LOBJ file may reference additional routines not defined within the same file. If these indirectly-referenced routines also are not defined on the Protocol Spreadsheet, the LOBJ files are searched again.

Routine definitions containing the `static` attribute are local to the LOBJ file. A `static` routine will be copied from the file only if it is included in the definition of another routine.

**NOTE:** Use `#pragma object` directives in your active spreadsheet program only. Do not incorporate them in code that will be compiled and saved as an LOBJ file. Although the code will compile, no search for routine definitions in referenced LOBJ files will be performed.

(c) **Efficiently uses memory.** Using `#pragma object` to reference routines in linkable-object files, assists in using the INTERVIEW's memory and spreadsheet buffer efficiently.

- Only the definitions for routines actually called within the current spreadsheet program are copied into memory from the LOBJ file. All other code within the file is ignored.

- When commonly utilized routines are defined in linkable-object files, space in the spreadsheet buffer otherwise dedicated to this purpose can be used for additional programming.

- Since the code in LOBJ files has already been compiled, the INTERVIEW can support a larger program without a corresponding increase in compilation time.

**NOTE:** Additional `#pragma` preprocessor directives utilized by the INTERVIEW are discussed in other sections of the manual. Refer to Section 61 on Display Window and Trace, for example, for information on the `#pragma tracebuf` directive. Except for `#pragma hook` (below), these other `#pragmas` should be part of the active spreadsheet program, not part of a linkable-object file.
(D) #pragma hook

The #pragma hook directive allows compiled C code within a referenced linkable-object file to be automatically combined with the compiled code of an active spreadsheet program. There are eight types of #pragma hook directives—hook_types zero through seven. All types may be system-generated during the Compile operation when the source of code is SPREADSHEET, but the resulting linkable-object file always contains at least one hook_type zero.

The programmer also uses hook_type zero (#pragma hook 0). For this reason, #pragma hook 0 will be the focus of the following discussion. The primary purpose of #pragma hook 0 is to "force" a routine to be called and executed as part of a spreadsheet program, even though no explicit call to the routine is made on the Protocol Spreadsheet. The spreadsheet program may also call the routine, but keep in mind that it will be executed twice—once because of the call on the spreadsheet and once because of the call made via the #pragma hook 0 directive.

1. Format. Create hooks on the Protocol Spreadsheet and then write them to a file using the WRITE/U editor command. Before typing your hook on the spreadsheet, press $ to prevent the editor from placing a strike-through over the text.

The format for the #pragma hook 0 directive is as follows:

```
#pragma hook hook_type "routine_name();"
```

Follow the directive with a space and enter a decimal (not hexadecimal) constant to identify the hook_type.

After the hook_type, enter another space, and then the hook text—C code that calls the routine you want combined with your spreadsheet program. The call to the routine is placed inside quotation marks and includes required syntax—parentheses for the arguments and a semi-colon to complete statement punctuation.

NOTE: Task names are always local to a linkable-object file and never directly copied from it. The hook text, therefore, cannot reference a task. The rule for exporting tasks from a linkable-object file is to let the #pragma hook 0 directive call a routine which starts the task(s). See Section 5. following and Section 52 for examples.

More than one #pragma hook 0 directive may be present in a single LOBJ file, but each directive calls only one routine.
2. Routine definitions. Typically, the definition for the routine called in the directive is located within the same linkable-object file. It may, however, be in another LOBJ file as long as both files are referenced via OBJECT block-identifiers on the Protocol Spreadsheet.

The definition of the hook-text routine may also reference a task (which must be defined in the same file) or it may reference additional routines not defined within the same file. The rules in Section (C) above for indirectly referencing routines apply.

Definitions for most of the extern routines included in this manual are not strictly required.

3. Accessing hooks. If you want the hook text combined with your program, use the OBJECT block-identifier to reference the LOBJ file. If you use the #pragma object directive to reference the file, the "hooks" within that file will be ignored.

4. Hooks are added to task list of program main. As your program compiles, referenced linkable-object files are searched for hooks. When a hook_type zero directive is found in the file, the hook text is automatically added to the bottom of the task-list in the top-level main. If a referenced LOBJ file contains more than one "hook," they will be added to the task list in the order in which they appear in the file.

   NOTE: The order of tasks and hooks in the task-list indicates the order in which main initiates tasks and executes hook routines. It does not necessarily indicate the order in which the actions in tasks or hooks are processed.

5. Execution of hooks. Recall that the main function is system-created during compilation. Refer to Section 52, Program Main. Because main simply initiates the execution of each task listed, the (hook-text) routine essentially runs concurrently with the tests in your spreadsheet program.

Since the hook text is a routine, and not a task, it must actually be executed by main, not simply started. The definition of the routine determines when, or whether, any subsequent hooks will be executed by main.

- If the routine's definition references a task, as in the example below, main returns quickly, leaving the routine to execute the task. Then main begins execution of the next hook in the task list.

```c
#pragma hook 0 "example();"
extern fast_event fevar_time_of_day;
extern volatile unsigned short crnt_time_of_day;
```
task
{
    main()
    {
        state_alarm_at_one:
        waitfor
        {
            var_time_of_day && (crnt_time_of_day == 1300):
            {
                sound_alarm();
            }
        }
    }
}
example_task;
example()
{
    example_task();
}

- If the routine's purpose is not to start a task (or tasks), then main has to execute all the code. The more code there is, the longer it will be before main can return to execute the next hook.

If the definition includes a waitfor, as in the following example, any subsequent hooks will never get executed. Instead, main will continue to wait for the specified event.

```
#pragma hook 0 "example();"
extern fast_event var_time_of_day;
extern volatile unsigned short crnt_time_of_day;
example()
{
    waitfor
    {
        var_time_of_day && (crnt_time_of_day == 1300):
        {
            sound_alarm();
        }
    }
}
```

56.5 Data Types

(A) Precisions

When a variable is declared, the compiler allocates space in memory according to the type declaration that precedes the variable name. There are three sizes (or precisions) of data allowable in 80286 memory, and three corresponding data types. A char is allotted one byte of memory. A short is given two bytes, while a long reserves four bytes of memory. Shorts and longs are varieties of int or integer, and the type descriptions short int and long int are permitted. The type int used by itself is the same as short int.
(B) Signed and Unsigned Types

All three precision types may be signed or unsigned. Signed and unsigned data types are stored identically, but treated differently in arithmetic operations. Specifically, they differ in the way they undergo type conversion, comparison, division, and right shifting.

1. Type conversion. The following declarations store the same value in memory:

```c
signed char a = -6;
unsigned char b = -6;
```

In both cases, the byte stored in memory will be the two’s complement of 00000110, or 11111010. (The two's complement is the one's complement + 1.) This bit pattern translates as hex fa or ASCII z. The `displayf` routine in the following program will write two z’s to the screen:

```c
{ signed char a = -6; unsigned char b = -6; 
  
  STATE: data_type 
  CONDITIONS: ENTER_STATE 
  ACTIONS: 
    { displayf("%c%c", a, b); 
      
    }
}
```

When you lengthen the chars to shorts, however, they behave differently. The unsigned char is left-padded with zeroes. The signed char, having a leftmost bit equaling 1, is left-padded with ones. This left-padding with ones is called “sign extension.”

A char is converted to a short automatically when a %d, %u, or %x conversion is applied to it, so the following example illustrates the difference between the conversion of signed and unsigned types:

```c
{ signed char a = -6; unsigned char b = -6; 
  
  STATE: data_type 
  CONDITIONS: ENTER_STATE 
  ACTIONS: 
    { displayf("%x%x", a, b); 
      
    }
}
```

The variable a will be seen to extend to hex fff9, which is fa left-padded with eight ones. The unsigned variable b will have been extended by eight zeroes and will appear unchanged as fa.

If the %x conversion specifiers in the example above are replaced by %d, the resulting signed-decimal conversion will show a equalling -6, b equalling 250. The signed char will have survived the type-lengthening with its original negative value intact.
Because they can be lengthened without changing their values, signed variables should be used for any arithmetic operations. Other differences between signed and unsigned variables, not reflected in Table 56-1, are the following:

2. **Comparison.** If the leftmost bit of a signed variable is 1, then the variable has a negative value and the expression `variable > 0` is false. If the leftmost bit of an unsigned variable is 1, the variable is positive and `variable > 0` is true.

3. **Division and modulus.** If the leftmost bit of a signed variable is 1, the two's complement of the variable rather than the stored value will be used in any division or modulus operation.

4. **Right shifting.** When a right–shift (>>) operator is used on a signed variable, a 1-bit is shifted in at the left. When the same operation is performed on an unsigned variable, a 0-bit is shifted in.

Table 56-1 shows the ranges of values that are produced by `displayf` and `printf` routines when the valid conversion specifiers—%c, %d, %ld, and so on—are applied to the various signed and unsigned data types. Frequently it makes no difference whether a variable is declared as signed or unsigned. When a variable undergoes type conversion, however, as in the case of a `char` given a decimal or hex conversion, there is a significant difference.
<table>
<thead>
<tr>
<th>type</th>
<th>char conversion (%c)</th>
<th>signed decimal conversion short (%d) long (%ld)</th>
<th>unsigned decimal conversion short (%u) long (%lu)</th>
<th>hex conversion short (%x) long (%lx)</th>
</tr>
</thead>
<tbody>
<tr>
<td>char¹</td>
<td>% to %</td>
<td>0 to 255</td>
<td>0 to 255</td>
<td>0 to ff</td>
</tr>
<tr>
<td>signed char¹</td>
<td>% to %</td>
<td>-128 to 127</td>
<td>0 to 127 and 65408 to 65535</td>
<td>0 to 7f and ff80 to ffff</td>
</tr>
<tr>
<td>unsigned char¹</td>
<td>% to %</td>
<td>0 to 255</td>
<td>0 to 255</td>
<td>0 to ff</td>
</tr>
<tr>
<td>Int</td>
<td>-</td>
<td>-32768 to 32767</td>
<td>0 to 65535</td>
<td>0 to ffff</td>
</tr>
<tr>
<td>signed int</td>
<td>-</td>
<td>-32768 to 32767</td>
<td>0 to 65535</td>
<td>0 to ffff</td>
</tr>
<tr>
<td>unsigned int</td>
<td>-</td>
<td>-32768 to 32767</td>
<td>0 to 65535</td>
<td>0 to ffff</td>
</tr>
<tr>
<td>short</td>
<td>-</td>
<td>-32768 to 32767</td>
<td>0 to 65535</td>
<td>0 to ffff</td>
</tr>
<tr>
<td>signed short</td>
<td>-</td>
<td>-32768 to 32767</td>
<td>0 to 65535</td>
<td>0 to ffff</td>
</tr>
<tr>
<td>unsigned short</td>
<td>-</td>
<td>-32768 to 32767</td>
<td>0 to 65535</td>
<td>0 to ffff</td>
</tr>
<tr>
<td>long</td>
<td>-</td>
<td>-2147483648 to 2147483647</td>
<td>0 to 4294967295</td>
<td>0 to fffffffff</td>
</tr>
<tr>
<td>signed long</td>
<td>-</td>
<td>-2147483648 to 2147483647</td>
<td>0 to 4294967295</td>
<td>0 to fffffffff</td>
</tr>
<tr>
<td>unsigned long</td>
<td>-</td>
<td>-2147483648 to 2147483647</td>
<td>0 to 4294967295</td>
<td>0 to fffffffff</td>
</tr>
</tbody>
</table>

¹ Through "integral promotion," char is converted automatically to int in a %d, %u, or %x conversion.
(C) Static Storage Class

A variable must be of the static storage class to pass its value into a waitfor statement. Declarations at the Program, Layer, or Test level (Level 1 in the source code diagram in Figure 52-4) are static even if they are not explicitly declared so. The same is true of a character array initialized by a string (see Section 56.7).

A variable that is initialized at the State level must be declared as static by the programmer if the initialized value is to be used inside a waitfor.

The following program will display a value of 8 on the prompt line when the operator presses the spacebar:

```c
STATE: pass_initialized_value
{
    static int initialized = 8;
    CONDITIONS: KEYBOARD • •
    ACTIONS:
    {
        displayf("%d", initialized);
    }
```

If you removed the word static from the declaration, the initialized value would not be passed into the condition clause and the program would display 0 or a "garbage" number instead of 8.

56.6 Operator Precedence

In an expression with more than one operator, operations are prioritized according to the ranking of operator precedence in Table 56-2. The operator with the highest precedence is at the top of the table. Precedence decreases as you move down.

Consider this example:

```c
STATE: precedence
{
    int a;
    a = 3 * 4 + 2;
    displayf("%d", a);
}
```

Because multiplicative operators (*, /, and %) have higher precedence than additive operators (+ and -), the 3 * 4 operation is performed first. Then 2 is added to the product of 3 and 4, and finally the sum is assigned to the variable a. (Assignment operators have very low precedence.) The result of the program is that a is displayed as 14. Compare this example:

```c
STATE: precedence
{
    int a;
    a = 3 * (4 + 2);
    displayf("%d", a);
}
```
Table 56-2
Operator Precedence

<table>
<thead>
<tr>
<th>Operator</th>
<th>Type of Operator</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>()</td>
<td>primary expression</td>
<td>left to right</td>
</tr>
<tr>
<td>[] . -&gt; ++ --</td>
<td>postfix</td>
<td>left to right</td>
</tr>
<tr>
<td>++ -- sizeof &amp; * + - !</td>
<td>unary</td>
<td>right to left</td>
</tr>
<tr>
<td>(type) * / %</td>
<td>multiplicative</td>
<td>left to right</td>
</tr>
<tr>
<td>+ -</td>
<td>additive</td>
<td>left to right</td>
</tr>
<tr>
<td>&lt;&lt; &gt;&gt;</td>
<td>bitwise shift</td>
<td>left to right</td>
</tr>
<tr>
<td>&lt; &gt; &lt;= &gt;=</td>
<td>relational</td>
<td>left to right</td>
</tr>
<tr>
<td>== !=</td>
<td>equality</td>
<td>left to right</td>
</tr>
<tr>
<td>&amp;</td>
<td>bitwise AND</td>
<td>left to right</td>
</tr>
<tr>
<td>^</td>
<td>bitwise exclusive OR</td>
<td>left to right</td>
</tr>
<tr>
<td></td>
<td>bitwise inclusive OR</td>
<td>left to right</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>logical AND</td>
<td>left to right</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>? :</td>
<td>conditional</td>
<td>right to left</td>
</tr>
<tr>
<td>= *= /= %= += -= &lt;&lt;= &gt;&gt;= &amp;= ^=</td>
<td>assignment</td>
<td>right to left</td>
</tr>
<tr>
<td>,</td>
<td>comma</td>
<td>left to right</td>
</tr>
</tbody>
</table>

Operators on the same line have the same precedence; rows are in order of decreasing precedence.

Here the additive operation is performed before the multiplicative, since the parentheses that denote a primary expression (see Table 56-2) have the highest precedence of all. The result of this program is that decimal 18 is displayed.

Given operations with the same precedence, left-to-right or right-to-left “associativity” (see the right column in Table 56-2) indicates which is performed first. This order of processing is significant for an expression such as 36 / 6 / 2, where the associativity is left to right.

Associativity is very important in assignment operations, which are always interpreted in a right-to-left direction. Consider this example:

```c
STATE: right_to_left_assocativity
{
    int a = 4;
    int b = 1;
    a = b;
    displayf ("%d", a);
}
```
The result of this program is that 1 is displayed, not 4. Right-to-left associativity also explains why the following program does not compile.

```c
STATE: right_to_left_associativity
{
    int a = 3;
    3 = a;
    display ("%d", a);
}
```

A constant never can have a value assigned to it, even if the value equals the constant.

### 56.7 Strings

A string is a sequence of characters enclosed in double quotes. This is an example of a string:

"hello"

A string is an expression of the type `pointer`, and may be used anyplace in the program that is appropriate for a pointer. For example, a pointer is appropriate as the argument of a `displays` routine:

```c
displays ("hello");
```

The string in this statement does two things during compilation: it writes the character string "hello" in memory, and it points to the first character in the string. The string "hello" becomes a 4-byte address that you can examine by displaying it as a `long` hexadecimal:

```c
display ("%lx", "hello");
```

(A) Using a String to Initialize an Array

Note that the pointer represented by "hello" in the examples above is not stored anywhere and therefore can be used only once. The string pointer "hello" could have been stored as a pointer to the first character in an array, as in this example:

```c
char string_array [] = "hello";
display (string_array);
```

Stored in this manner, the pointer can be used repeatedly.

An array like `string_array` that has been "initialized" by a string shares many of the traits of standard arrays, but it has unique characteristics as well.

1. **Data type.** A string may only initialize an array whose elements are of the type `char`.

2. **Null termination.** A string is always terminated by a null character. This null terminator is appended by the compiler, not the programmer.
3. **Size.** All arrays must declare their size, in any of three ways. The programmer may declare the length inside of brackets, as in this example:

```c
char array [5];
```

Or he may leave the brackets empty and provide a list of initializers, inside of curly braces, from which the compiler can determine the size of the array:

```c
char initializer_list_array [] = {'h', 'e', 'l', 0x6c, 'o'};
```

The third method of indicating size is to leave the brackets empty and initialize the array with a string, as in our original example of a string initializer:

```c
char string_array [] = "hello";
```

The compiler will add a terminating null-character to this string, and calculate an array size of six. To verify that the compiler counts one more character than the user has entered, you may try the following test. Note that the `sizeof` operator will return the length of any array:

```c
STATE: display_size_of_string
{
    char string_array [] = "hello";
    int compiler_count = sizeof(string_array);
    display("%d", compiler_count);
}
```

4. **One-dimensional array.** Whereas arrays in general can be multidimensional, a string-initialized array always has one dimension.

(B) **Valid Strings**

1. **ASCII and control.** With a few exceptions, all ASCII characters, including control characters, are valid in a string. The exceptions are \, \, "", and \. These characters are liable to be misinterpreted by the compiler. Null (\) and linefeed (\) will be taken to indicate a new logical line in the program. Double-quote (") will be mistaken for the end of the string. Backslash (\) will be misinterpreted as the start of an escape sequence.

If one of these characters is included in a string, the program may not compile. If not, you will be returned to the Protocol Spreadsheet. The following message will be displayed for nulls or linefeeds: "**Error 718: Newline inside string.**" For quotation marks, the message is "**Unclosed AR "C" region.**" Depending on their placement in the string, backslashes may or may not generate an error. Even when compilation succeeds, however, they will not be interpreted correctly.
## It

### Table 56-3

<table>
<thead>
<tr>
<th>Non-literal</th>
<th>Meaning</th>
<th>ASCII character</th>
<th>Hex character</th>
</tr>
</thead>
<tbody>
<tr>
<td>\a</td>
<td>bell</td>
<td>\r</td>
<td>0T</td>
</tr>
<tr>
<td>\b</td>
<td>backspace</td>
<td>\s</td>
<td>0b</td>
</tr>
<tr>
<td>\f</td>
<td>form feed</td>
<td>\t</td>
<td>0c</td>
</tr>
<tr>
<td>\n</td>
<td>linefeed †</td>
<td>\n</td>
<td>0a</td>
</tr>
<tr>
<td>\r</td>
<td>carriage return</td>
<td>\r</td>
<td>0o</td>
</tr>
<tr>
<td>\t</td>
<td>horizontal tab</td>
<td>\t</td>
<td>0n</td>
</tr>
<tr>
<td>\v</td>
<td>vertical tab</td>
<td>\t</td>
<td>0g</td>
</tr>
<tr>
<td>'</td>
<td>single quote</td>
<td>'</td>
<td>0g</td>
</tr>
<tr>
<td>&quot;</td>
<td>double quote †</td>
<td>&quot;</td>
<td>0g</td>
</tr>
<tr>
<td>\</td>
<td>backslash</td>
<td>\</td>
<td>000</td>
</tr>
<tr>
<td>##</td>
<td>octal representation</td>
<td>any ASCII character</td>
<td>00 0 - FF</td>
</tr>
<tr>
<td>###</td>
<td>hex representation</td>
<td>any ASCII character</td>
<td>00 0 - FF</td>
</tr>
</tbody>
</table>

† These characters require non-literal entries in INTERVIEW strings. The others may be entered as ASCII characters, non-literals, or hexadecimal characters.

2. **Non-literals.** Most characters in strings are interpreted literally. Each of the invalid characters listed above, therefore, needs a non-literal representation. Non-literals are preceded by a backslash. The compiler converts these non-literals to their one-byte numeric value.

To include a null (or any ASCII) character in a string, use the octal or hexadecimal representation shown in Table 56-3. Hex and octal numbers take up to three digits, so use leading zeroes if necessary. Otherwise, a subsequent digit may be interpreted as part of the value. Suppose, for example, you want to create the string “abc”. You initialize an array as follows:

```c
char string[] = "\x0abc";
```

The string will be stored as “abc” (hexadecimal characters \x0). The correct declaration was `char string[] = "\x000abc"`. In octal, the null would be written \000.

Please note that a string that has a null character somewhere other than at the end will be difficult to display or print completely. Display and print routines that take strings as input typically begin at the pointer position and continue until they encounter a terminating null. If, as in the last example, a null is encountered at the beginning of the string, execution of the routine will end before anything has been displayed or printed.

Provide precision to the %H conversion specifier to override null termination of a string while displaying a string in hex: see Section 60.3(C).
3. **Constants.** Spreadsheet constants may be included in strings. An example of a spreadsheet constant is the fox message represented as "(FOX)". See Section 25 on Constants.

The C translator expands constants both inside and outside of C regions before the code is preprocessed.

4. **Hexadecimal characters.** ASCII characters, including the control characters, may be entered in strings as hexadecimal characters via the \x key. Hex representation is considered literal. That is, you may not enter ASCII characters which require non-literal representation in strings as hexadecimal characters. The sequence of characters comprising a non-literal may be entered as hexadecimal characters. Double backslash (\\), for example, may be entered as %\__.

(C) **String Routines**

There are several C routines in the INTERVIEW that display or print strings. See Section 63 on “Print” and Section 60 on “Display Window and Trace” for detailed descriptions of the prints, displays, and traces routines, as well as other display and print routines that use the %s conversion specifier.

There is also a pair of routines, index and rindex, that search inside of strings for particular characters. These routines are defined (with examples) in Section 67.

56.8 **Recommended Sources**

The following sources provide accurate, in–depth information on C Programming Language:


   **NOTE:** When approved, the number for the ANSI document will change to: **ANSI Standard X3.159-198X.**


57 Variables

57.1 Creating or Accessing C Variables

Softkey-selectable programming "tokens" entered by the user on the Protocol Spreadsheet are translated automatically into C during the initial compiler phases after ~ is pressed. (Then the C code in turn is compiled into object code.) The C variables used by the translator are documented throughout this volume.

C regions available to the user at every level of spreadsheet programming (see Section 53) provide direct access to these variables.

An example of a user-accessible variable is keyboard_new_key, used in the following program to sound an alarm whenever any ASCII-keyboard key is pressed.

{  
  extern fast_event keyboard_new_key;
}

STATE: anykey
CONDITIONS:
  
  keyboard_new_key

ACTIONS: ALARM

The C regions also allow the user to work with variables of his own creation.

Here is an example of a user-created variable named minutes that is used to count minutes elapsed since the beginning of Run mode. The C program displays this "counter" on the prompt line of the Run-mode screen.

{  
  extern fast_event fevar_time_of_day;
  short minutes;
}

STATE: run_mode_minutes
CONDITIONS:
  
  fevar_time_of_day

ACTIONS:
  
  minutes++;
  pos_cursor (0,0);
  displayf ("Duration of run = %4d minutes", minutes);
The first C region in the example "declares" the variables `fevar_time_of_day` and `minutes`. The first of these variables is an event variable that is built into the system software. All event variables in an active State-block are polled constantly. Once every minute, `fevar_time_of_day` returns true.

The second variable, `minutes`, is created by the program itself—that is, by the user. The declaration in effect creates the variable: it causes 16 bits in memory ("short" = 16 bits) to be dedicated to information stored under the name `minutes`.

The second C region in the example is placed inside the Actions block. The statement `minutes++` causes the value that is stored in the 16 bits dedicated to `minutes` to increment. The function `pos_cursor (0,0)` places the cursor in the leftmost column on the second line of the display screen (the Prompt line). The `display` function writes a text message to the display screen, beginning at the current cursor position. In the text message itself, "%" will be replaced by the current value of the variable `minutes`. "4" means that four columns on the screen will be dedicated to the value, and "d" means that the value will be expressed in a decimal number.

57.2 Declaring Variables

Declare your variables and routines in a C region, delimited by curly braces { and }, at the top of your program or at the top of a Constants, Layer, Test, State, or Actions block. Declare a variable preceded by its type descriptors and followed by a semicolon, as in these examples:

```
{ 
extern fast_event keyboard_new_key;
extern fast_event keyboard_new_any_key;
extern fast_event fevar_time_of_day;
short minutes;
}
```

A variable may have its scope limited to a particular Test, State, or Actions block. A variable also may be redeclared at different levels. (In software revision 5.00 or earlier, it may not be redeclared at the same level.) Given more than one valid declaration, the lower or nearer one applies.

The rules governing the placement of variable declarations are laid out in detail in Section 53.5(A).

(A) Naming Variables

1. Legal names. The first letter of a variable name may be either a letter or an underscore. Following characters may be letters, numbers, underscores, or dollar signs.

   Reserved words (indicated in boldface type in Appendix K) may not be used as variable names.
2. **Naming conventions.** Generally speaking, variables that begin with `dte_` or `dce_` are used by the software to test DTE and DCE conditions. Variables that begin `rcvd_` are used to test RECEIVE (or RCV) conditions. Variables that begin `m_` are used by the layer packages to construct the protocol traces.

(B) **Modifiers**

1. **Data type.** The data type for each variable precedes the variable name in the declaration. All standard data types except `float` are supported in the INTERVIEW 7000 Series. Standard data types and their sizes and ranges are given in Table 56-1.

2. **Preassigned modifiers.** When you declare a user-accessible external variable, be sure to use the modifiers which precede the data type for that variable as listed in variable tables throughout this volume.

57.3 **Comparing a Variable to a Value**

User-accessible and user-created variables may be tested as part of any standard C expression.

The following is an example of a user-invented variable called `anykey` that is declared with a default value of zero, incremented by the operator pressing any ASCII-keyboard key, and checked for a value of 3 by an `if` statement after each depression of a key. An alarm will sound on the third keystroke.

```c
{extern fast_event keyboard_new_key;
 short anykey;
}

STATE: press_key
CONDITIONS:
{
 keyboard_new_key
}

ACTIONS:
{
 anykey++;
 if (anykey == 3) sound_alarm ();
}
```

The next example uses a built-in, user-accessible variable called `crnt_time_of_day` and checks it for a particular value. This 16-bit variable stores the time of day in hours and minutes. The Condition in the program (the event variable `fevar_time_of_day`) is true once per minute. The Action each time the condition is true is to check `crnt_time_of_day` for a value of 1129. At 11:29 AM, an alarm will sound.


```c
extern fast_event fevar_time_of_day;
extern volatile unsigned short crnt_time_of_day;

STATE: alarm clock
CONDITIONS:
{
  fevar_time_of_day
}
ACTIONS:
{
  if (crnt_time_of_day == 1129) sound_alarm();
}
```

### 57.4 Checking a Variable in a `waitfor` Clause

Please note that the following variation on the preceding example does not produce the same result. *The alarm will never sound* if this version of the program is run:

```c
extern volatile unsigned short crnt_time_of_day;

STATE: alarm clock
CONDITIONS:
{
  crnt_time_of_day == 1129
}
ACTIONS:
{
  sound_alarm();
}
```

Note that the time-of-day condition that was lodged in an `if` statement in the previous example has now been placed in a Conditions block. Conditions blocks on the Protocol Spreadsheet are converted to `waitfor` clauses (see Section 53.3), not `if` statements, when the program is translated automatically into C coding.

*Waitfor* clauses work very differently from `if` statements and other conditional control structures in C.

#### (A) Event vs. Nonevent Variables

Two kinds of variables may be used inside of these `waitfor` clauses—event variables and nonevent variables. When a state is active, event variables in that state are checked regularly during routine polling by the CPU. When an event variable (such as `fevar_time_of_day`) is polled and returns a value of true, conditional statements containing nonevent variables (such as `crnt_time_of_day`) also are checked for truth or falsity. *In the absence of an event variable being polled and returning a value of true, a statement about a nonevent variable inside of a Conditions block (waitfor clause) never can be true.*

Since there is no event variable in the Conditions block (`waitfor` clause) above, the nonevent variable `crnt_time_of_day` is never even checked.
(B) Translation of Softkey Tokens Into Variables

You could have written the "alarm clock" program using only softkey entries, as follows:

STATE: alarm clock
CONDITONS: TIME 1129
ACTIONS: ALARM

In this case, the C translator will convert the Conditions block into a waitfor clause that uses the event variable fevar_time_of_day to check the nonevent variable crnt_time_of_day once a minute. Here is the translator's version of the Conditions and Actions blocks:

```c
{ waitfor
  {fevar_time_of_day && (crnt_time_of_day == 1129):
    {sound_alarm();}
  }
}
```

(C) Example of A Nonevent Condition "Waiting For" An Event

The next example illustrates the interplay of event variables and nonevent variables in a waitfor clause.

```c
{ extern fast_event keyboard_new_key;
  short anykey;

  STATE: press_key
  CONDITIONS:
  {
    keyboard_new_key
  }
  ACTIONS:
  {
    anykey++;
  }
  CONDITIONS:
  {
    anykey == 3
  }
  ACTIONS: ALARM
}
```

This program looks similar to a previous one in which the operator hit three keys and the alarm sounded. Here, however, the alarm does not sound until the fourth keystroke. The variable anykey begins the test at zero, and increments (anykey++) with every keystroke. But remember what a condition such as anykey == 3 in a waitfor clause really means. It means that the condition will be true when the variable equals three and an event (such as a keystroke) occurs that causes the variable to be checked. On these terms, the condition is not satisfied until the fourth event.
(D) User-Created Event Variables

The user can create his own event variable simply by declaring a new variable with the modifiers \textit{extern event}. Once the event variable has been declared, he can use the \textit{signal} function to indicate that the event has occurred. Here is an example of an event variable called \textit{check_number} that causes the nonevent variable \textit{number} to be checked—and sounds the alarm when the value of \textit{number} satisfies the condition.

```c
{  short number = 3;
   extern event check_number;
}

STATE: user_created_event
{  signal (check_number);
}

CONDITIONS:
{  check_number && (number == 3)
}

ACTIONS: ALARM
```

(E) Rules and Cautions

To sum up the discussion of event and nonevent variables, here are a few rules of thumb:

1. \textit{If} statements, \textit{for} loops, \textit{while} loops, and other conditional control structures may not be used in Conditions blocks (that is, in \textit{waitfor} clauses). They may be used in State blocks, above (or in the absence of) Conditions blocks; and they may be used in Actions blocks.

   (Placing an \textit{if} statement at the top of the State block, above any \textit{waitfor} clauses, is how the translator converts \texttt{ENTER\_STATE} softkey conditions into C.)

2. Event variables are designed for use in Conditions blocks (\textit{waitfor} clauses) only. It makes no sense to use an event variable in an \textit{if} statement, \textit{while} loop, etc., since there is no possibility that the event will be true at the precise moment the statement is being processed.

3. A Conditions block (\textit{waitfor} clause) that lacks an event variable can never come true.

One other word of caution about the importance of event variables: please note that the following program will not sound the alarm even if the operator presses a key while the time is 11:29 AM.
57 Variables

{ extern fast_event keyboard_new_key;
  extern volatile unsigned short crnt_time_of_day;
}

STATE: alarm_clock
CONDITIONS:
{  keyboard_new_key && (crnt_time_of_day == 1129)
}
ACTIONS: ALARM

The reason this program doesn’t “work” is that all variables begin Run mode at zero. Often a particular event variable must return true before a particular nonevent variable will be updated. The nonevent variable crnt_time_of_day is updated only when the event variable fevar_time_of_day is entered in the waitfor clause and returns true. In the example above, the operator pressing the key will cause crnt_time_of_day to be checked; but in the absence of fevar_time_of_day, the value of crnt_time_of_day remains always at zero.

57.5 Checking and Displaying Equivalent Values of a Variable

Variables may be checked and displayed as octal, decimal, hexadecimal, and ASCII-character values. Decimal comparison and display is the default.

(A) Checking Equivalent Values

To compare a variable to an octal value, precede the value with a zero (0). No prefix is necessary to make a decimal comparison. To compare a variable to a hexadecimal value, precede the value with 0x or 0X. To check whether a variable matches an ASCII character, enter the character in between single quotes.

The alarm will sound in the example below, since all of the values entered to the right of the equal signs are equivalent.

{  char foxtrot = 'f';
}

STATE: compare_equivalent_values
{  if (((foxtrot == 0146) && (foxtrot == 102) && (foxtrot == 0x66) && (foxtrot == 'f')) sound_alarm();
}

Note that the data type char in the declaration simply means that the variable is composed of 8 bits. The designation char does not say anything about the comparison mode or the display mode. (Data types short and int = 16 bits; long = 32 bits.)
(B) Displaying Equivalent Values

Variables may be displayed in a variety of data formats via the `displayf` function. The full set of display conversions is given in Table 61-7. The program below generates a representative sample of display formats. When the program is run, the prompt line on the display screen will look like this: 152 106 6a 6A j 

```c
{
    char juliet = 'j';
}

STATE: display_equivalent_values
{
    displayf("\%o %d %x %X %c %#u ", juliet, juliet, juliet, juliet, juliet);
}
```

57.6 Isolating Bits from a Variable Value

Some variables are bit-oriented. That is, one bit (or perhaps a small field of bits) may have significance that is independent of the surrounding bit values. The variable `current_eia_leads` (refer to Table 60-1), for example, uses 7 bits to store the on/off status of seven separate EIA leads, plus an eighth bit to store the status of any lead that is patched to the UA input jack (see Section 10.3). If you want to check this variable to determine the status of DTR (for example) you need to determine whether the bit that represents DTR (the fifth bit from the right or the fifth least significant bit in the variable) is set to 1 (DTR off) or zero (DTR on). How can you isolate this bit from the surrounding bits in order to determine its status?

The tool for isolating a bit in a C variable is the “care mask,” a group of bits (usually expressed in hexadecimal) in which the bit(s) under scrutiny is set to 1 and all other bits to zero. The care mask for DTR is 0x10 (or 16 in decimal notation). The binary version, 00010000, shows that only the DTR bit is set to 1. When this care mask is anded (via the "&" operator) with the variable `current_eia_leads`, only two results are possible, depending on whether the DTR bit in `current_eia_leads` is 1 or 0.

With DTR on, suppose that the combination of all lead statuses gives `current_eia_leads` a value of e6 in hex—11100110 in binary. The effect of anding this variable with the care mask for DTR will be as follows:

```
11100110
& 00010000
00000000
```

Now turn DTR off, and the result of the anding will be this:

```
11110110
& 00010000
00010000
```
The seven "don't care" zeroes in the care mask guarantee seven zero-bits in the result (because 0 & 1 = 0 and 0 & 0 = 0). So the result of the anding must be either 0 if the DTR bit is 0 (on), or hex 10 (decimal 16, binary 00010000) if the DTR bit is 1 (off).

This C program will detect DTR on:

```c
extern fast_event fevar_eia_changed;
extern const volatile unsigned short current_eia_leads;

STATE: check_dtr_on
CONDITIONS:
{ fevar_eia_changed }
ACTIONS:
{ if ((current_eia_leads & 0x10) == 0) sound_alarm(); }
```

If you try to run this program, make sure of the following:

1. The Front-End Buffer Setup menu should be configured to buffer control leads.

2. If you are not connected to a device that provides clock, the Line Setup menu should be configured to provide internal clock. EIA leads are clocked through the front-end buffer before they reach the program logic.

3. After the program enters Run mode, use a single-wire patch cord to connect the +12V output pin on the test-interface module to the DTR lead. The alarm should sound as soon as the patch is made.

A slightly different condition inside of the if statement will detect DTR off:

```c
if ((current_eia_leads & 0x10) == 0x10) sound_alarm();
```

The DSR bit is the fourth least significant bit in the current_eia_leads variable, so the care mask for DSR is 0x08 (binary 00001000). The following if statement will detect DSR on:

```c
if ((current_eia_leads & 0x08) == 0) sound_alarm();
```

This if statement will detect DTR on and DSR on:

```c
if ((current_eia_leads & 0x18) == 0) sound_alarm();
```

This if statement will detect DTR off and DSR on:

```c
if ((current_eia_leads & 0x18) == 0x10) sound_alarm();
```

The last condition simply means that you care (1=care) about DTR and DSR and you want DTR to be 1 (off) and DSR to be 0 (on).
57.7 Pointing to an Address

Some routines require an address as input. The displays (display-string) routine, for example, requires a CPU memory address as its argument. When executed, the routine will begin to display characters that it finds at the specified address and at subsequent addresses, one by one, until a null is encountered. A memory address is four bytes (32 bits) and is declared as a long.

```
{
  long any_cpu_address;
}
```

Some routines require an address as input. The displays (display-string) routine, for example, requires a CPU memory address as its argument. When executed, the routine will begin to display characters that it finds at the specified address and at subsequent addresses, one by one, until a null is encountered. A memory address is four bytes (32 bits) and is declared as a long.

```
{
  long any_cpu_address;
}
```

Many of the important addresses needed by the user and by the program can be found inside of interlayer ("IL") message buffers. When BOP-framed data is monitored, it is copied automatically into IL buffers. Each time a frame is buffered, a data primitive is created automatically and the event variable m_lo_ph_prmtv is signaled. The segment number of the IL buffer is recorded in the variable m_lo_ph_il_buff. This segment number can be converted into an address.

Here, for example, is a program that looks for a DTE data packet, converts m_lo_ph_il_buff into a four-byte address that points to the first data position, and displays the data contents of the packet.

```
{
  long first_data_address;
  extern volatile unsigned short m_lo_ph_il_buff;
}
```

The IL buffer is illustrated in Section 63 of this manual, and the procedure for converting the buffer-segment number into a memory address is explained in detail in Section 63.1(C). Briefly, we have cast the segment number (a short, 16 bits) into a long and moved the number over to its high-order position in the CPU address, sixteen bits to the left. Then we added 37 to the number to bypass the header information for the buffer (32 bits) and the frame and packet headers (5 bits).

```
{
  long first_data_address;
  extern volatile unsigned short m_lo_ph_il_buff;
}
```

Each address in memory stores 8 bits, so the second byte in the data field of the data packet would be first_data_address + 1, the fourteenth byte would be first_data_address + 13, and so on.
57.8 Creating a Character Pointer

For most of the variables in a C program, the address is not important to the user or to the program. The user does not need to know the address in order to declare the variable, perform operations on it, and compare its value to other values. In general, addresses of variables are solely the concern of the compiler.

In the case of a routine such as *displays*, the address is what is important. The value that is stored at the address is not so important, since the routine will go to the address and begin displaying the data whatever the value (as long as the value is displayable).

There is another kind of variable for which both the address and the value stored at the address are important. These variables are called pointers. The user creates a pointer by typing an asterisk (*) just following the data type in a declaration, as in this example:

```
char * packet_type_ptr;
```

The variable `packet_type_ptr` is a four-byte memory address just as `first_data_address`, declared as a `long` in the previous example, was a four-byte address—even though `packet_type_ptr` is declared as a `char`. The data type `char` preceding the asterisk simply means that the amount of data pointed to is eight bits.

Once you use an asterisk to declare the variable a pointer, you can access the address directly as `packet_type_ptr` or you can access the value stored at that address as `* packet_type_ptr`. A *displays* routine would accept `packet_type_ptr` as input, while a *displaysc* or *displaysf* routine would expect `* packet_type_ptr`.

With the X.25 personality package loaded at Layers 2 and 3 (via the Layer Setup screen), the following program goes to the memory location pointed to by `packet_type_ptr` and checks its value to determine whether the packet in the buffer is a Clear request.

```c

{ extern volatile unsigned short m_lo_ph_il_buff;
  extern event dte_packet;
  char * packet_type_ptr;
 }

STATE: search_for_dte_clear
CONDITIONS:
{ dte_packet }
ACTIONS:
{ packet_type_ptr = (void *) (((long) m_lo_ph_il_buff << 16) + 36); 
  if (*packet_type_ptr == 0x13) sound_alarm();
}
```
The pointer `packet_type_ptr` is a `char`, but you could just as easily point to a `short` (16 bits) or a `long` (32 bits). If you increment an address, you get the next address, 8 bits farther in memory. If you increment a `char` pointer, you also get the next address. If you increment a `short` pointer, you add two increments to the memory address. In effect you move the pointer two places. If you increment a `long` pointer, you move the pointer by four addresses, 32 bits.

In the example above, the integer `m_lo_ph_il_buff` is cast as a pointer `(void *)` after it is cast as a `long`. This is to avoid a compiler error ("Warning 31: Illegal implicit integer-to-pointer conversion") when the new value of `m_lo_ph_il_buff` is assigned to `packet_type_ptr`.

### 57.9 Pointing with Subscripts

When it is preceded by an asterisk (*), the pointer `packet_type_ptr` returns the character value that it points to, as we have just seen. Another way to return this value is to omit the asterisk and add a subscript: `packet_type_ptr[0]`. This mechanism allows you to access an array of values without moving the pointer.

For example, the transmission header ("TH") in a FID2 SNA information field is six bytes long. If you establish a pointer to the first TH byte (TH0), you can use subscripts to access any other byte in the field without moving the pointer. The following program checks the values of two bytes in the TH field (corresponding to "DAF" and "OAF") before freezing the data display and sounding an alarm.

```c
{ extern volatile unsigned short m_lo_ph_il_buff;
  char * th;
}
```

```c
LAYER: 2
STATE: th pointer
CONDITIONS: DTE INFO
ACTIONS:
{  th = (void *) (*((long) m_lo_ph_il_buff << 16) + 34);
   {  ctl_capture_td (0x10);
      ctl_capture_rd (0x100);
      sound_alarm ();
   }
}
```

### 57.10 Creating a String

Strings are used in INTERVIEW programming mainly for transmissions and for messages to the operator ("prompts"). In the following program, the compiler decodes the string "QWERTYUIOP" from ASCII to hex, stores it in memory as a series of contiguous values, adds a null to it, returns the address of the first character, "Q," and then assigns this address to the variable `keyrow`:

```
57-12
```
The variable `keyrow` now is the four-byte address of "Q" in the string. You can see this address for yourself by using either "QWERTYUIOP" or `keyrow` as the argument in a `displayf` routine:

```c
displayf ("%lx ", "QWERTYUIOP");
```

or

```c
displayf ("%lx ", keyrow);
```

Either version will display a CPU address (hex 04400000) on the second line of the Run-mode screen.

The string can be displayed in a simple `displays` routine, since that routine expects a four-byte address as input:

```c
displays ("QWERTYUIOP");
```

or

```c
displays (keyrow);
```

If you want to access individual characters in the string, declare a pointer:

```c
char * keyrow = "QWERTYUIOP";
```

With a pointer you can display the entire string or a single character—the seventh character, "U," in this example:

```c
displays (keyrow);
displayc (keyrow[6]);
```

Declaring the string an array has virtually the same effect as declaring it a pointer:

```c
char keyrow [] = "QWERTYUIOP";
```

The name of the array still is the address of the first character in the string and so may be used in a `displays` routine; and individual characters still may be specified by a subscript:

```c
displays (keyrow);
displayc (keyrow[6]);
```

The only difference is that the array name is a constant whose value is assigned in a declaration and cannot be changed, while the pointer is a variable and may be incremented, assigned a new value, and so forth, while the program is running.
57.11 Comparing Strings

A string comparison in C may be conducted as follows. First, create a pointer in the manner described in Section 57.8, or else simply declare one of the pointers to line data that is provided in the set of user-accessible variables. Example: `extern volatile unsigned char * m_packet_ptr;`

Next, create an array that represents the search string you will try to match against the line data. For example:

```c
char search_string[] = "\xa";
```

Create a trigger to look for a line event (such as the event variable `dteyacket`) that will initialize the pointer.

```c
{ extern volatile unsigned char * m_packet_ptr;
  char search_string[] = { 0x10, 0x04, 0x0b }; 
  extern event dte_packet;
}
```

Compare the pointer-value with the first element of the search string. If a match is found, increment the pointer and compare the new value to the second element of the search string; and so on. If a match is found for every element of the string, take an appropriate action.

```c
ACTIONS:
{
  if (search_string[0] * m_packet_ptr)
  {
    m_packet_ptr ++;
    if (search_string[1] == * m_packet_ptr)
    {
      m_packet_ptr ++;
      if (search_string[2] == * m_packet_ptr) sound_alarm ();
    }
  }
}
```

Here is the same Actions block, only this time the variable `element` replaces the numeral in the subscript to `search_string`, and the same variable is added as a subscript to `m_packet_ptr`. This coding may be modified easily for any length string. For a 9-byte string, for example, simply change the 3 in the if statement to 9.
ACTIONS:
{
  element = 0;
  while (search_string [element] == m_packet_ptr [element])
  {
    if (search_string[element++] == 3)
    {
      sound_alarm();
      break;
    }
  }
}

57.12 Accessing a Variable Inside of a Structure

A structure is a mechanism that makes repetitive declarations of similar variables unnecessary. For example, there are twelve variables associated with any given counter created in the program. One variable is the current value of the counter, one is the last sampled value, another is the highest sampled count, another the total of all the sampled values, another the number of samples taken, and so forth. If the user creates four counters via the spreadsheet softkeys, the C translator does not declare 48 separate variables (4 x 12). Instead the translator declares a structure for counters—called `counter STRUCT—that declares each of the twelve variables once, as follows:

```
{  
  struct counter STRUCT
  { 
    unsigned long current;
    unsigned long last;
    unsigned long maximum;
    unsigned long minimum;
    unsigned short sample_count;
    unsigned long total_high;
    unsigned short total_low_low;
    unsigned short total_low_high;
    unsigned short out_of_range;
    unsigned short changed;
    unsigned long prev;
    unsigned long old;
  );
```

Then the translator declares each of the user's four counters as having the structure `counter STRUCT`:

```
struct counter STRUCT dte_good_bcc, dte_bad_bcc, dce_good_bcc, dce_bad_bcc;
```

In effect the translator has declared all 48 variables. Suppose the user wants to access one of these variables. He may wish to display the total value of a counter whose current value no longer is the total value (since the counter may have been sampled—and therefore cleared—several times). As long as the total is less than 65,536, the entire number will reside in the seventh variable in the `counter STRUCT` structure, `total_low_low`. If the counter in question is `dce_good_bcc`, he will access this "total" variable under the name `dce_good_bcc.total_low_low`. 

57-15
Here is a sample trigger that displays this variable whenever the operator presses $\mathbb{Q}$:

```
STATE: dispaly total_dce_good bcc
CONDITIONS: KEYBOARD 'T'
ACTIONS:
{    displayf("Total DCE good BCC's = %d", dce_good_bcc.total_low_low);
}
```

Refer to Section 62.1 for more detail on the structure of counters.

### 57.13 Creating a Structure Pointer

We have just seen how a structure can be created to store and access data conveniently. A structure can also be used as a multibyte pointer that is superimposed on data that has been stored previously.

In our example we will declare the structure of an IL buffer and then point this structure at a newly received IL buffer.

The precise structure of an IL buffer is given in the following declaration. Note that there are 32 bytes devoted to header information and the remaining 4K bytes are available for data.

```c
struct il_buffer
{
    unsigned short lock;
    unsigned short maintain_bits;
    unsigned short buffer_size;
    unsigned short transmit_tag;
    unsigned short receive_tag;
    unsigned long char_buff_frame_start;
    unsigned long char_buff_frame_end;
    unsigned short tick_count_high;
    unsigned short tick_count_mid;
    unsigned short tick_count_low;
    unsigned short available_space_offset;
    unsigned short bytes_remaining;
    unsigned long bcc_indicator;
    unsigned char data [4064];
};
```

The next step is to create a pointer that has the structure of `il_buffer`. First, declare the structure of `il_buffer`, as indicated above. Then declare `buffer_ptr` as a structure-pointer, as follows:

```c
struct il_buffer * buffer_ptr;
```

The next step is to wait for an INFO frame to be monitored. When the the frame data has been buffered and `m_lo_ph_il_buff` has been updated with the new buffer-segment number, assign the first address of this buffer to `buffer_ptr`.

```
buffer_ptr = (void *) ((long) m_lo_ph_il_buff << 16);
```
Now a structure has been created around the most recent upward-moving IL buffer. This means that rather than moving a pointer around in the IL buffer, you can access elements in the buffer directly. The `tick_count_low` variable, for example, would be called `buffer_ptr->tick_count_low`. (The \texttt{->} operator is used in place of the dot operator in structure-pointers.)

The first element of the `data` string would be called `buffer_ptr->data[0]`. Here is a program that displays on the prompt line the fifth data element (the packet-type byte) in the IL buffer for Info frames monitored on DTE.

```c
{ extern volatile unsigned short m_lo_ph_il_buff;
  struct il_buffer
  {
    unsigned short lock;
    unsigned short maintain_bits;
    unsigned short buffer_size;
    unsigned short transmit_tag;
    unsigned short receive_tag;
    unsigned long char_buff_frame_start;
    unsigned long char_buff_frame_end;
    unsigned short tick_count_high;
    unsigned short tick_count_mid;
    unsigned short tick_count_low;
    unsigned short available_space_offset;
    unsigned short bytes_remaining;
    unsigned long bcc_indicator;
    unsigned char data[4064];
  }
  struct il_buffer * buffer_ptr;
}

LAYER: 2
STATE: monitor Il buffers
CONDITIONS: DTE INFO
ACTIONS:
  { buffer_ptr = (void *) ((long) m_lo_ph_il_buff <<16);
    pos_cursor (0,0);
    displayf("%02x", buffer_ptr->data[4]);
  }

57-17
58 Routines

This manual documents the C routines that are "external" to the C program—that is, defined elsewhere than in the program. Most of these routines are used by the C translator when it converts softkey-selectable programming "tokens"—most commonly those tokens that are appropriate to Actions blocks—entered by the user on the Protocol Spreadsheet. Some, like the Disk I/O routines, are associated with no spreadsheet conditions or actions and can be accessed only in C regions on the spreadsheet.

58.1 Declarations

In most of the examples in the manual, we have not bothered to declare routines since it is not necessary. In the absence of a declaration, the compiler assumes that the routine is external and that it returns an integer. In nearly all cases, this assumption works. In those rare cases when the routine returns another data type (the stats-display routine `get_68k_phys_addr`, for example, returns a `long`) it must be declared.

58.2 Arguments

An argument is an input that the user provides when he calls a routine. Arguments are placed inside of parentheses just following the routine name, as in this call to the `pos_cursor` routine: `pos_cursor (1,5);

This routine requires two arguments in order to position the cursor in one of 1,088 possible character positions. The first argument selects one of the seventeen horizontal rows. The second argument selects one of the sixty-four vertical columns.

Many routines in the INTERVIEW library have arguments whose names end in the letters `ptr` or `pointer`. If you look at the synopsis for the `displays` routine, for example, you will see that the only argument is something called `string_ptr`. This is an address argument. The user enters a four-byte address as argument when he calls the displays routine, and the routine goes to this address and begins displaying data until a null (or other nondisplayable character) is encountered.

Pointers are four-byte addresses. The following call to the `displays` routine will go to the location of `m_packet_info_ptr` (the first byte of user data in a packet) and begin displaying data until a nondisplayable character is encountered:

`displays (m_packet_info_ptr);`
Array names also are four-byte addresses. The following example will display the characters in the array string:

```c
char string[] = "QWERTY";
displays(string);
```

A string of characters declared inside of double-quotations marks is really a four-byte address that points to the first character in the string. In the function call `displays("qwertyuiop")`, "qwertyuiop" qualifies as a string pointer and therefore satisfies the formal definition of the routine.

Many routines have no arguments and are called with empty parentheses:

```c
sound_alarm();
```

Do not omit the parentheses. Without them, `sound_alarm` is a variable instead of a routine.

### 58.3 Returns

In addition to performing various operations, many routines include a `return` function that, at the end of the routine, stores a user-defined value in a memory location. As an example, we will look at an X.25 routine called `13_window_full`.

The `13_window_full` routine is declared automatically by the translator after the user has made a WINDOW FULL softkey entry. The synopsis for `13_window_full` shows how it is declared:

```c
extern unsigned char 13_window_full (path_number);
```

The routine is declared as a `char` because at the end of the routine, a return function will store a `char`-sized value (8 bits) in memory. If the packet window is full, the stored value will be nonzero. If the packet window is not full, the value will be zero.

The stored value is accessed any time you call the routine in your program. If you want to test for the window being full, you can enter this line of code:

```c
if (13_window_full(path_number) != 0) sound_alarm();
```

Here is a simpler coding for the same test:

```c
if (13_window_full(path_number)) sound_alarm();
```

This coding works for the same reason that `if (1) sound_alarm();` or `if (0) sound_alarm();` will sound the alarm. Nonzero constants, variables, and expressions are true in C and cause statements to be executed inside of `if`, `while`, and other control constructions. Constants, variables, and expressions that equal zero are false and prevent statements in control structures from being executed.
If a routine is declared as a short, a short will be set aside in memory and any value returned by the routine (via a return function) will be stored there. If the routine is declared a long, a long will be reserved. If the routine is declared void, no space will be reserved in memory and a call to return a value will not be successful.

58.4 User-Defined Routines

The following coding will blank out the prompt line near the top of the INTERVIEW run-mode display.

```c
pos_cursor(0,0);
displays ("");
```

If you code these two routines each time you display a user-prompt, you can always be sure that the prompt line will be blank and that each prompt will overwrite the previous prompt completely. The only problem is that the two routines are laborious to type in.

A better way is to declare a routine that executes the two "subroutines" automatically.

Declare a routine with its arguments inside parentheses and its body—the list of statements or subroutines that the routine is intended to perform—inside a pair of curly braces.

```c
void blank_prompt_line()
{
    pos_cursor(0,0);
displays ("");
}
```

Now you can blank out the line simply by typing this:

```c
blank_prompt_line();
```

Suppose you wanted a routine that blanked the prompt line and generated a new prompt. The new prompt will be the argument for the routine:

```c
void new_prompt (string_pointer)
char string_pointer [];
{
    pos_cursor(0,0);
displays ("");
pos_cursor(0,0);
displays (string_pointer);
}
```

Now you can generate a prompt against a blank background with this simple routine:

```c
new_prompt ("This prompt will overwrite any previous prompt");
```
NOTE: User routines may be declared and defined outside of the current spreadsheet program—in include files or linkable-object files. See Section 56.4.

58.5 Example Routines

We will provide three examples that will help illustrate how routines are created.

(A) Example Routine: Temporary Prompt

Here is a user-defined routine that blanks the prompt line, displays a new user-defined prompt, and then waits a user-defined interval before blanking the prompt line again. The routine is called temporary_prompt. The two inputs are 1) the new prompt, and 2) the number of seconds that you want the prompt to remain on the display.

The routine incorporates one external routine, timeout_restart_action, discussed in Section 69.3 of the section titled "Other Library Tools," and one internal routine, blank_prompt_line, discussed above.

```c
#include <stdio.h>

struct {
    int event_id;
    int event_id_uid;
} timeout_prompt;

void blank_prompt_line()
{
    pos_cursor(0, 0);
    displays(" ");
}

void temporary_prompt (string_pointer, seconds)
char string_pointer [];
char seconds;
{
    blank_prompt_line();
    pos_cursor(0, 0);
    displays(string_pointer);
    timeout_restart_action (&timeout_prompt, seconds * 1000, blank_prompt_line);
}

STATE: test_temporary_prompt
CONDITIONS: KEYBOARD * "
ACTIONS:
{
    temporary_prompt("This prompt will self-destruct in 4 seconds.", 4);
}
```

Note that the blank_prompt_line routine is embedded inside the timeout_restart_action routine, which in turn is embedded inside the temporary_prompt routine.
Note also:

The structure `timeout_prompt` is needed by the `timeout_restart_action` routine. The structure is explained in Table 69-1.

The two arguments in the `temporary_prompt` routine are declared outside the body of the routine (that is, outside of the curly braces). As a result, they are not redeclared each time the routine is called.

Timeout timers increment in milliseconds, so the user’s `seconds` entry is multiplied by 1,000.

(B) Example Routine: Display Binary Value of Byte

The next sample routine takes a user-defined 8-bit value as input and expands it into a binary display of ASCII 1's and 0's. The routine, called `display_binary`, uses the & ("and") operator to isolate each bit and turn it into a "1" or "0" in an ASCII string called `binary_string`. See Section 57.6 for a discussion of the & operator.

The condition-and-action program that follows the declaration of `display_binary` uses the routine to expand the packet-type byte in each DCE packet.

```c
{ extern volatile unsigned char * m_packet_ptr;
  extern event dce_packet;
  char binary_string[8];
  void display_binary (hex_value)
    char hex_value;
    { if ((hex_value & 0x80) == 0) binary_string[0] = '0';
      else binary_string[0] = '1';
      if ((hex_value & 0x40) == 0) binary_string[1] = '0';
      else binary_string[1] = '1';
      if ((hex_value & 0x20) == 0) binary_string[2] = '0';
      else binary_string[2] = '1';
      if ((hex_value & 0x10) == 0) binary_string[3] = '0';
      else binary_string[3] = '1';
      if ((hex_value & 0x08) == 0) binary_string[4] = '0';
      else binary_string[4] = '1';
      if ((hex_value & 0x04) == 0) binary_string[5] = '0';
      else binary_string[5] = '1';
      if ((hex_value & 0x02) == 0) binary_string[6] = '0';
      else binary_string[6] = '1';
      if ((hex_value & 0x01) == 0) binary_string[7] = '0';
      else binary_string[7] = '1';
      displayf("\n%8s", binary_string);
    }
  ST
  STATE: binary
  CONDITIONS: { dce_packet }
  ACTIONS:
    { display_binary (m_packet_ptr[2]);
  }
```
(C) Example Routine: Compare String Against Line Data

Here is a routine called `strcmp` that matches a user-entered string to line data, beginning at a point in the line data that the user specifies. The arguments are the string itself and a pointer to the beginning of the line data.

When the user enters his string inside double quotes, the compiler writes the string into memory, appends a zero (null), and returns a pointer to the first character in the string. The `strcmp` routine uses this zero to determine when the match is complete.

If a complete match is found, the `return(1)` routine breaks out of the while loop, so the `return(0)` never is executed. A routine that returns 1 (or nonzero) inside of an if condition will make the condition true.

The sample program that uses the `strcmp` routine looks on the DCE side for a data packet with a user-data field that begins “%PASSWORD.” This string occurs on the “HDLC/X.25 Data Sample” diskette, DSK-951-007-1, shipped with your INTERVIEW. Be sure to load in the Layer 2 and Layer 3 X.25 packages if you try out this program. The Layer 3 package will provide you with your line-data pointer (`m_packet_info_ptr`).

```c
extern volatile unsigned char *m_packet_info_ptr;
int element;
int strcmp (user_string_ptr, line_data_ptr)
char *user_string_ptr [];
char *line_data_ptr;
{
    element = 0;
    while (user_string_ptr[element] == line_data_ptr[element])
    {
        if (user_string_ptr[++element] == 0)
            return (1);  
    }
    return(0);
}
LAYER: 3
STATE: match user_data_field
CONDITIONS: DCE_DATA
ACTIONS:
{
    if (strcmp("\x0d\x0aPASSWORD", m_packet_info_ptr))
        sound_alarm();
}
```
59 Monitor/Transmit Line Data

The external variables and routines in this section are available for use by the programmer to monitor and transmit data. Their use on the Protocol Spreadsheet is not limited to any particular layer, though normally they belong at Layer 1.

The variables and routines approximate Layer 1 spreadsheet-generated conditions and actions. Refer to Section 28 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.

59.1 Structures

Use the structure `xmit_list`, shown in Table 59-1, when transmitting line data via the `ll_transmit` routine. Refer to `ll_transmit` in Section 59.3(B) for an example of how to use this structure.

<table>
<thead>
<tr>
<th>Structure Name:</th>
<th>xmit_list</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Variable</td>
</tr>
<tr>
<td>unsigned char *</td>
<td>string</td>
</tr>
<tr>
<td>unsigned short</td>
<td>string_length</td>
</tr>
</tbody>
</table>

Structure of a transmit list for `ll_transmit` routine. Declared as type `struct`. Reference member variables of the structure as follows: `xmit_list.string_length`. |

Table 59-1
Transmit Structures
59.2 Variables

(A) Monitoring Events

1. Emulate or monitor mode. Layer 1 events include characters received, good or bad BCC's, aborts, parity errors, and framing errors. All event variables in Table 59-2 containing a _td or _rd suffix are valid in either emulate or monitor mode. These event variables are fevar_rcvd_char_rd, fevar_rcvd_char_td, fevar_gd_bcc_rd, fevar_gd_bcc_td, fevar_bd_bcc_rd, fevar_bd_bcc_td, fevar_abort_td, fevar_abort_rd, fevar_parity_td, fevar_parity_rd, fevar parity_id, fevar_frm_error_rd, fevar_frm_error_td, and fevar_rcv_buffer_full. The variable fevar_frm_error_rd, for example, equates to DCE FRAMING_ERROR (or RECEIVE FRAMING_ERROR when you are emulating DTE).

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

CONDITIONS:
{  
  fevar_bd_bcc_td || fevar_bd_bcc_rd  
}

ACTIONS: COUNTER bad_bcc INC

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

CONDITIONS: DTE BAD_BCC
ACTIONS: COUNTER bad_bcc INC
CONDITIONS: DCE BAD_BCC
ACTIONS: COUNTER bad_bcc INC

Use fevar_rcv_buffer_full and its associated status variable, rcv_buffer_full, to monitor the status of the character buffer. The moment the buffer is full, fevar_rcv_buffer_full comes true and the value of rcv_buffer_full transitions from zero to a non-zero value. Then, new data begins to overwrite the old data. The softkey equivalent of fevar_rcv_buffer_full is the layer-independent condition BUFFER_FULL when it appears alone in a conditions block. When BUFFER_FULL is combined with another condition, in most cases the other condition will supply the event variable and only the status test will be used. See Section 27 for a discussion of this and other layer-independent conditions and actions.
### Table 59-2

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern fast_event</td>
<td>fevar_rcvd_char_rd</td>
<td>True for each character received on RD. Line Setup configured for emulate or monitor mode.</td>
<td></td>
</tr>
<tr>
<td>extern fast_event</td>
<td>fevar_rcvd_char_td</td>
<td>True for each character received on TD. Line Setup configured for emulate or monitor mode.</td>
<td></td>
</tr>
<tr>
<td>extern fast_event</td>
<td>fevar_gd_bcc_rd</td>
<td>True when a good BCC is calculated for an RD block or frame. Line Setup configured for emulate or monitor mode.</td>
<td></td>
</tr>
<tr>
<td>extern fast_event</td>
<td>fevar_gd_bcc_td</td>
<td>True when a good BCC is calculated for a TD block or frame. Line Setup configured for emulate or monitor mode.</td>
<td></td>
</tr>
<tr>
<td>extern fast_event</td>
<td>fevar_bd_bcc_rd</td>
<td>True when a bad BCC is calculated for an RD block or frame. Line Setup configured for emulate or monitor mode.</td>
<td></td>
</tr>
<tr>
<td>extern fast_event</td>
<td>fevar_bd_bcc_td</td>
<td>True when a bad BCC is calculated for a TD block or frame. Line Setup configured for emulate or monitor mode.</td>
<td></td>
</tr>
<tr>
<td>extern fast_event</td>
<td>fevar_abort_rd</td>
<td>True when an abort is detected in an RD frame. Line Setup configured for emulate or monitor mode.</td>
<td></td>
</tr>
<tr>
<td>extern fast_event</td>
<td>fevar_abort_td</td>
<td>True when an abort is detected in a TD frame. Line Setup configured for emulate or monitor mode.</td>
<td></td>
</tr>
<tr>
<td>extern fast_event</td>
<td>fevar_parity_rd</td>
<td>True when a parity error is detected for an RD byte. Line Setup configured for emulate or monitor mode.</td>
<td></td>
</tr>
<tr>
<td>extern fast_event</td>
<td>fevar_parity_td</td>
<td>True when a parity error is detected for a TD byte. Line Setup configured for emulate or monitor mode.</td>
<td></td>
</tr>
<tr>
<td>extern fast_event</td>
<td>fevar_frm_error_rd</td>
<td>True when an async framing error is detected for an RD byte. Line Setup configured for emulate or monitor mode.</td>
<td></td>
</tr>
<tr>
<td>extern fast_event</td>
<td>fevar_frm_error_td</td>
<td>True when an async framing error is detected for a TD byte. Line Setup configured for emulate or monitor mode.</td>
<td></td>
</tr>
</tbody>
</table>
**Table 59-2 (continued)**

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern fast_event</td>
<td>fevar_xmit_cmplt</td>
<td></td>
<td>True when the INTERVIEW puts a transmission out onto the link. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern fast_event</td>
<td>fevar_rcv_buffer_full</td>
<td></td>
<td>Returns true at the moment the character buffer fills with data and will begin to overwrite existing data. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>rcv_buffer_full</td>
<td>0</td>
<td>not full</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>full</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern unsigned short</td>
<td>rcvd_char_td</td>
<td></td>
<td>Most recent TD character is stored in this variable. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0-ff/0-255</td>
<td>data character (lower byte in 16-bit data word in data buffer)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100/256</td>
<td>good or bad BCC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>101/257</td>
<td>flag</td>
</tr>
<tr>
<td></td>
<td></td>
<td>102/258</td>
<td>sync</td>
</tr>
<tr>
<td></td>
<td></td>
<td>103/259</td>
<td>abort</td>
</tr>
<tr>
<td>extern unsigned short</td>
<td>rcvd_char_rd</td>
<td></td>
<td>Most recent RD character is stored in this variable. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0-ff/0-255</td>
<td>data character (lower byte in 16-bit data word in data buffer)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100/256</td>
<td>good or bad BCC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>101/257</td>
<td>flag</td>
</tr>
<tr>
<td></td>
<td></td>
<td>102/258</td>
<td>sync</td>
</tr>
<tr>
<td></td>
<td></td>
<td>103/259</td>
<td>abort</td>
</tr>
<tr>
<td>extern unsigned char</td>
<td>td_modifier</td>
<td></td>
<td>Most recent modifier byte for a TD data character. This is the upper byte in the 16-bit data word reserved for each data character in the data buffer. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>data—Initial value (always included in value of tdModifier)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>alternate code set</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>underline (rd character)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>reverse image</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10/16</td>
<td>hexadecimal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20/32</td>
<td>low intensity</td>
</tr>
<tr>
<td></td>
<td></td>
<td>40/64</td>
<td>blink</td>
</tr>
<tr>
<td></td>
<td></td>
<td>80/128</td>
<td>strike-thru (parity error)</td>
</tr>
</tbody>
</table>
Table 59-2 (continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern unsigned char</td>
<td>rd_modifier</td>
<td>Most recent modifier byte for an RD data character. This is the upper byte in the 16-bit data word reserved for each data character in the data buffer. Line Setup configured for emulate or monitor mode.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>data (always included in value of rd_modifier)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>alternate code set</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>underline (rd character)—initial value of rd_modifier</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>reverse image</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10/16</td>
<td>hexadecimal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20/32</td>
<td>low intensity</td>
</tr>
<tr>
<td></td>
<td></td>
<td>40/64</td>
<td>blink</td>
</tr>
<tr>
<td></td>
<td></td>
<td>80/128</td>
<td>strike-thru (parity error)</td>
</tr>
</tbody>
</table>

2. **Emulate mode only.** One variable is valid in emulate mode only, since it monitors an emulate action. “SENDing” a transmission means queuing a transmission to send. The layer protocol (the RTS–CTS handshake, for example, at Layer 1) may delay the actual transmission. The fast-event variable fevar_xmit_cmplt will not come true until the transmission actually has been sent. Use this condition to start accurate response–time measurements.

If you try to use fevar_xmit_cmplt in monitor mode, you will be returned to the main program menu. When you go to the Protocol Spreadsheet and search for errors, the following message will be displayed: “Error 140: Unresolved reference fevar_xmit_cmplt.”

(B) **Status Variables**

Status variables are those in Table 59-2 that do not include event in the Type column. Their associated event variables guarantee that they are updated and tested.

1. **Distinguishing character types.** Suppose you’re monitoring the DCE side of the link. Every time a character is detected, the event fevar_rcvd_char_rd comes true, regardless of whether or not the character will be stored in the character buffer. Not all characters are “data” characters. A character also may be a flag or the second byte in a block-check, for example. fevar_rcvd_char_rd (or fevar_rcvd_char_td) does not distinguish character types.

Character type is stored in the high byte of rcvd_char_rd or rcvd_char_td. For data characters, the high byte is zero. The low byte contains the actual value of the character.
For a “non-data” character, hereafter referenced as a special symbol, the high byte of \texttt{rcvd\_char\_rd} is a non-zero value. The low byte specifies a special symbol to be displayed on the data screen, overwriting or replacing the character. The special symbols are \texttt{}$/ (sync), \texttt{}$/ (good BCC), \texttt{}$/ (bad BCC), \texttt{}$/ (abort), and \texttt{}$/ (flag). See Table 59-2.

Notice on Table 59-2 that the value for good BCC and bad BCC is the same. Use \texttt{fevar\_gd\_bcc\_rd} and \texttt{fevar\_bd\_bcc\_rd} event variables to distinguish between good and bad BCC’s (or data BCC’s in DDCMP). Likewise, use \texttt{fevar\_gd\_bcc2\_rd} and \texttt{fevar\_bd\_bcc2\_rd} to differentiate between good and bad header BCC’s in DDCMP. Refer to Section 75 for DDCMP variables.

Aborts are not automatically reflected in \texttt{rcvd\_char\_rd} and \texttt{rcvd\_char\_td}. When seven consecutive 1-bits are received in 7E-framed protocols, the controller chip generates an interrupt. The bits, however, are not stored in memory. In this case, use \texttt{fevar\_abort\_rd} or \texttt{fevar\_abort\_td} to detect the interrupt. When this event variable transitions to true, it updates \texttt{rcvd\_char\_rd} (or \texttt{rcvd\_char\_td}) to indicate an abort.

Use \texttt{rcvd\_char\_td} and \texttt{rcvd\_char\_rd} to monitor received characters, independent of whether or not they will be buffered. The following condition detects RD data characters only:

\texttt{CONDITIONS:}
\begin{verbatim}
{
    fevar\_rcvd\_char\_rd && !(rcvd\_char\_rd & 0x100)
}
\end{verbatim}

2. \textit{Attributes}. Data characters and special symbols in the character buffer are available for normal or enhanced display on the data display-screen. Access the data display by pressing DATA on the first rack of Run-mode softkeys, or by selecting it as the initial Run-mode display on the Display Setup menu.

The current attributes for RD data are stored in \texttt{rd\_modifier}. Table 59-2 shows how the various attributes are coded. The initial value of \texttt{rd\_modifier} is always five. This value means that the character is data (1) on the RD (4) side. RD data is always underlined. TD data is never underlined. The initial value of \texttt{td\_modifier}, therefore, is one.

You may change some attributes by using spreadsheet tokens (or their equivalent C routines). The Layer 1 \texttt{ENHANCE} action allows you to control reverse-image, blink, hexadecimal, and low intensity enhancements. This action also updates \texttt{rd\_modifier}, \texttt{td\_modifier}, or both.

When an RD data character is written to the character buffer, the value of \texttt{rd\_modifier} is written to the high byte of a two-byte data event-word. The data character, found in \texttt{rcvd\_char\_rd}, is written to the low byte. See Section 59.3(C) on the format of character-buffer event words.
NOTE: The attributes in rd_modifier and td_modifier do not apply to special symbols. rd_modifier and td_modifier always reflect the attributes last assigned to data. Underlining applied to (RD) special symbols on the data display-screen comes from a bit in the special receive-event word. See Table 59-3.

59.3 Routines

Unless noted otherwise, the routines discussed below apply when the Line Setup menu shows either emulate or monitor mode.

(A) Controlling Data Display

ctl_enhance_td

Synopsis

extern void ctl_enhance_td(enhance_type_status);
unsigned short enhance_type_status;

Description

This routine turns various enhancements of the data display on and off on the DTE side. It also updates the variable td_modifier. The softkey equivalent of this routine is the ENHANCE DTE action on the Protocol Spreadsheet.

Inputs

There is one two-byte parameter. The high byte identifies the type of enhancement to be controlled: blink (40), low intensity (20), hexadecimal representation (10), and reverse image (08). The low-order byte indicates the status of the enhancement. To indicate a given enhancement is on, the second byte has the same value as the first. If the enhancement is to be turned off, the value of the second byte is zero. For example, if you want to turn blink on, the parameter value is Ox4040. To turn blink off, it is Ox4000.

Multiple enhancements can be controlled with one action by using hexadecimal addition of the parameters, as in the example for ctl_enhance_rd.

Example

Assume X.25 protocol for this example. You want to enhance the packet type byte on the DTE side with a blinking, reverse image.

LAYER: 1
STATE: enhance_packet_type
CONDITIONS: DTE STRING "HDRXXXXXXX0"R
ACTIONS:
{
    ctl_enhance_td(0x4040);
    ctl_enhance_td(0x0808);
}

59-7
CONDITIONS: DTE STRING "1EIfJ«XXXXXXXO»~·
ACTIONS:
{
    ctl_enhance_td(0x4000);
    ctl_enhance_td(0x0800);
}

cntl_enhance_rd

Synopsis
extern void ctl_enhance_rd(enhance_type_status);
unsigned short enhance_type_status;

Description
This routine turns various enhancements of the data display on and off on the DCE side. It also updates the variable rd_modifier. The softkey equivalent of this routine is the ENHANCED DCE action on the Protocol Spreadsheet.

Inputs
See ctl_enhance_td.

Example
Assume X.25 protocol for this example. You want to enhance the packet type byte on the DCE side with a blinking, reverse image.

LAYER: 1
STATE: enhance_packet_type
    CONDITIONS: DCE STRING "1EIfJ«XXXXXXXO»~·
    ACTIONS:
    {
        ctl_enhance_rd(0x4848);
    }
    CONDITIONS: DCE STRING "1EIfJ«XXXXXXXO»~·
    ACTIONS:
    {
        ctl_enhance_rd(0x4800);
    }

cntl_capture_td

Synopsis
extern void ctl_capture_td(status);
unsigned short status;

Description
This routine turns on and off the presentation of DTE data to the screen—that is, it stops or “freezes” the display—and capture of data to the screen buffer (character RAM). Unlike the Manual Freeze mode initiated by the Enter key,
however, the "capture off" action does not allow you to scroll through the buffer
while the test continues. The softkey equivalent of this routine is the CAPTURE
DTE action on the Protocol Spreadsheet.

Inputs
The only parameter is the status of capture, on (0x00) or off (0x10). Turning
capture off freezes the display.

Example
Assume X.25 protocol for this example. You want to turn capture off as soon
as the cause byte is displayed in a Clear packet on the DTE side. Capture will
be resumed when the spacebar is pressed.

LAYER: 1
STATE: find_cause
CONDITIONS: DTE STRING "FE(XXXXXXX0)XEN" 
ACTIONS:
{ 
  ctl_capture_td(0x10);
}
CONDITIONS: KEYBOARD " 
ACTIONS:
{ 
  ctl_capture_td(0x00);
}

capture_rd

Synopsis
extern void ctl_capture_rd(status);
unsigned short status;

Description
This routine turns on and off the presentation of DCE data to the screen—that
is, it stops or "freezes" the display—and capture of data to the screen buffer
(character RAM). Unlike the Manual Freeze mode initiated by the ESI key,
however, the "capture off" action does not allow you to scroll through the buffer
while the test continues. The softkey equivalent of this routine is the CAPTURE
DCE action on the Protocol Spreadsheet.

Inputs
The only parameter is the status of capture, on (0x00) or off (0x100). Turning
capture off freezes the display.

Example
Assume X.25 protocol for this example. You want to turn capture off as soon
as the cause byte is displayed in a Clear packet on the DCE side. Capture will
be resumed when the spacebar is pressed.
outsync_action

Synopsis

extern void outsync_action(side);
unsigned short side;

Description

The outsync_action routine applies to synchronous format only. This routine sends one of the receivers (TD or RD) out of sync and initiates a search for sync. The softkey equivalent of this routine is the (PROTOCL) OUT_SYN action on the Protocol Spreadsheet.

Inputs

The only parameter identifies which side of the line is to go out of sync, 0 for the DTE side, 1 for the DCE side.

Example

To display DTE-protocol information only, initiate sync each time a start-of-text character is found. The results of this routine are similar to turning capture off and on, but here the display does not have to be turned on again. It resumes automatically with sync.

(B) Transmitting

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 11 Il_transmit."
**Synopsis**

```c
extern void ll_transmit(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 `ll_transmit` routine sends a specified string 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 a fox message at Layer 1 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:

```c
{ 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
    };
}
INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

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

ll_il_transmit

Synopsis

extern void ll_il_transmit(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 onto the line.

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 ll_transmit. The softkey equivalent of this routine is the SEND action on the Protocol Spreadsheet. Refer to Section 63.3(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 [] = "\x000\x000\x000\x000(FOX)";
}
LAYER: 1
STATE: send_message
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, &message[0],
                         (sizeof(message) - 1));
  il_il_transmit(il_buffer_number, relay_baton, data_start_offset, 1);
}

idle_action

Synopsis

extern void idle_action(character);
unsigned char character;

Description

Only for format SYNC, the idle_action routine allows you to change the idle-line
condition applied by the INTERVIEW. The softkey equivalent of this routine is
the (PROTOCOL) IDLE_LN action on the Protocol Spreadsheet.

Inputs

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

Example

X.21 or X.21BIS idles different characters in various states, "r", "b",  
+, for
example. To signal a change in protocol state, you might change the idle
character to +:

LAYER: 1
STATE: change_idle_character
CONDITIONS: KEYBOARD "*
ACTIONS:
{
  idle_action("+");
}
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. In other contexts, the routine simply initiates and terminates a break.

Inputs

The first parameter is the mask that is anded 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

This program will generate a 250-millisecond break when the operator presses the [ ] key.

```
{  
  extern fast_event keyboard_new_any_key;
  extern volatile unsigned short keyboard_any_key;
}

STATE: generate_break
  CONDITIONS:
  {
    keyboard_new_any_key && (keyboard_any_key == 0x1e3)
  }
  ACTIONS: TIMEOUT break RESTART 0.250
  {
    set_tcr_b (0xf3, 0x08);
  }
  CONDITIONS: TIMEOUT break
  ACTIONS:
  {
    set_tcr_b (0xf3, 0x04);
  }
```
(C) Writing to Character RAM

For the sake of speed, the 64-Kbyte character buffer uses a shorter data word than the 32-bit word in the Display Window and traces. Refer to Table 61-4. A sixteen-bit event word is reserved for each character in the 64-Kbyte character buffer.

Table 59-3 shows the format of event words. Two kinds of event word should be distinguished: data and special receive.

1. *Data Event-Words.* Data event-words may contain enhancement attributes in the high byte. Whereas attributes comprise 24 bits of a *long* in the Display Window and the traces, in the character buffer they are contained in only 8 bits. Data words in the character buffer, therefore, include a less flexible set of attributes. Color attributes, for example, are not directly available in words written to the character buffer. See Section 16, Color Display, for an explanation of how reverse, blink, and low enhancements in the character buffer may be mapped to colors in the RGB output. Table 59-3 lists the available attributes.

The character is located in the low 8 bits. Its value can range from hexadecimal 0 through FF.

2. *Special-Receive Words.* The high byte in special-receive words determines the symbol (from the special graphic character font) that will overlay the character contained in the low byte. The symbols that may be written to the character buffer are good BCC's, bad BCC's, aborts, flags, and sync. One bit, the td/rd indicator, controls on which side the symbol will be displayed. Symbols on the RD side are underlined, as all RD data is. Notice in Table 59-3 that the td/rd indicator bit is the same one that controls the underline enhancement in data event-words.

The value in the low byte is meaningless in the context of special-receive words. The special symbol will overlay or replace the character. Its value, nevertheless, can range from hexadecimal 0 through FF.
**Table 59-3**  
**Character Buffer 16-Bit Word**

<table>
<thead>
<tr>
<th>Type</th>
<th>Mask (hex)</th>
<th>Input (hex)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>0100</td>
<td>0100</td>
<td>data-event word: the low byte contains data</td>
</tr>
<tr>
<td></td>
<td>0500</td>
<td>0000</td>
<td>add 0100 to the following: td.rd indicator:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0400</td>
<td>td character</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>rd character (underlined)</td>
</tr>
<tr>
<td></td>
<td>ff00</td>
<td>0000</td>
<td>enhancements: t† (enhancements apply to data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0200</td>
<td>indicated in low byte)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0400</td>
<td>normal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0800</td>
<td>alternate code set</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1000</td>
<td>reverse image</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2000</td>
<td>hexadecimal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4000</td>
<td>low intensity</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8000</td>
<td>blink</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>strike-thru (parity error on character)</td>
</tr>
<tr>
<td>special receive</td>
<td>8300</td>
<td>0200</td>
<td>special receive-event word: reserved</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8200</td>
<td>reserved</td>
</tr>
<tr>
<td></td>
<td>8700</td>
<td>0000</td>
<td>td.rd indicator:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0400</td>
<td>td character</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>rd character (underlined)</td>
</tr>
<tr>
<td></td>
<td>bf00</td>
<td>0800</td>
<td>special event: (symbols for these events overlay the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1000</td>
<td>data indicated in low byte)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1800</td>
<td>good CRC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2000</td>
<td>bad CRC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2800</td>
<td>abort</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3000</td>
<td>flag</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3800</td>
<td>sync</td>
</tr>
<tr>
<td>reserved</td>
<td>0700</td>
<td>0400</td>
<td>reserved</td>
</tr>
<tr>
<td>reserved</td>
<td>0f00</td>
<td>0800</td>
<td>reserved</td>
</tr>
</tbody>
</table>

† Selecting rd (0400) for the td/rd indicator results in the data being underlined. The underline enhancement shares the same bit. It has been omitted from the list of enhancements to avoid an error from double counting.
The routines for writing 16-bit event words to the character buffer are
add_event_to_buff and add_array_to_buff. These routines may be used when
the Line Setup menu shows either emulate or monitor mode.

add_event_to_buff

Synopsis

extern unsigned int add_event_to_buff(event_word);
unsigned int event_word;

Description

The add_event_to_buff routine writes the specified input to the 64-Kbyte
character buffer.

Inputs

The only input is a 16-bit event-word to be written to the buffer. Table 59-3
lists the coding of event words.

Returns

A one is returned if the event was successfully added to the character buffer. If
the routine failed, zero is returned.

Example

To display only SDLC frames with an address of hexadecimal c2, enter the
following spreadsheet program:

```
LAYER: 1
{
    extern unsigned short rcvd_char_td;
    extern unsigned short rcvd_char_rd;
}
STATE: init
    CONDITIONS: ENTER_STATE
    ACTIONS: CAPTURE BOTH OFF
    NEXT_STATE: address
STATE: address
    CONDITIONS: DTE STRING "c2"
    ACTIONS:
    {
        if(rcvd_char_td == 0xc2)
            {
            add_event_to_buff ((short)td_modifier << 8) + rcvd_char_td);
            ctl_capture_td(0x00);
            }
    }
    CONDITIONS: DTE STRING "c2"
    ACTIONS: CAPTURE DTE OFF
```
Conditions: DCE String "ФП".

Actions:

```c

if(rcvd_char_rd == 0xc2)
{
    add_event_to_buff(((short)rd_modifier << 8) + rcvd_char_rd);
    ctl_capture_rd(0x00);
}
```

Conditions: DCE STRING "ФП".

Actions: CAPTURE DCE OFF.

---

**add_array_to_buff**

**Synopsis**

```c
extern unsigned int add_array_to_buff(array_ptr, count);
unsigned short * array_ptr;
unsigned char count;
```

**Description**

The `add_array_to_buff` routine writes specified elements of an array to the 64-Kbyte character buffer.

**Inputs**

The first parameter is the location of the array to be written to the character buffer. The array consists of 16-bit shorts.

The second parameter is the number of elements in the array to be written. The number of elements which can be written to the buffer must be in the range 0–16. Elements in the array must adhere to the format of event words shown in Table 59-3.

**Returns**

The result of the `add_array_to_buff` routine is all or nothing. A one is returned when all requested elements of the array are successfully added to the character buffer. If the routine fails, zero is returned and nothing is written to the buffer.

**Example**

To display on the Data Screen only X.25 packets with an LCN of 004, enter the following spreadsheet program. (This program displays the DTE side of the line only. Additional programming similar to that entered would include DCE data.)
LAYER: 1
{*
    unsigned short dte_array [100];
    unsigned short lcn;
    extern unsigned short rcvd_char_td;
}*

STATE: init
CONDITIONS: ENTER_STATE
ACTIONS: CAPTURE BOTH OFF
NEXT_STATE: address

STATE: address
CONDITIONS: DTE STRING "[PA]"
ACTIONS:
{*
    dte_array [0] = (0x0100 + rcvd_char_td);
}*
NEXT_STATE: frame_type

STATE: frame_type
CONDITIONS: DTE STRING "(XXXXXXX0)"
ACTIONS:
{*
    dte_array [1] = (0x0100 + rcvd_char_td);
}*
NEXT_STATE: gfi

CONDITIONS: DTE STRING "(XXXXXXX1)"
NEXT_STATE: address

STATE: gfi
CONDITIONS: DTE STRING "[B]"
ACTIONS:
{*
    dte_array [2] = (0x0100 + rcvd_char_td);
    lcn = ((unsigned int)rcvd_char_td & 0x0f) << 8;
}*
NEXT_STATE: lcn

STATE: lcn
CONDITIONS: DTE STRING "[K]"
ACTIONS:
{*
    dte_array [3] = (0x0100 + rcvd_char_td);
    lcn += rcvd_char_td;
    if(lcn == 0x0004)
    {
        add_array_to_buff(dte_array, 4);
        cti_capture_td(0x00);
        current_state = state_eof;
    }
    else
    {
        current_state = state_address;
        break;
    }
}*

STATE: eof
CONDITIONS: DTE STRING "[E]"
ACTIONS: CAPTURE DTE OFF
NEXT_STATE: address
60 EIA

The Test Interface Module (TIM) located in the rear of the INTERVIEW determines the EIA leads available for monitoring and control (Section 10). The variables and routines in this section apply to RS-232, V.35, and RS-449 interface modules. The X.21 module is treated separately in Section 70.

To use the C variables and routines explained in this section, enable EIA leads by selecting Buffer Control Leads: YES on the FEB Setup menu. See Section 7.1(B). If no other source for clock is provided, use internal clock (Line Setup menu).

The variables and routines approximate Layer 1 EIA 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.

60.1 Variables

With an RS-232, V.35, or RS-449 TIM installed, you may monitor RI, DSR, DTR, CD, CTS, RTS, and UA. The lead names in RS-449 are slightly different: see Table 60-1.

The fast-event variable fevar_eia_changed detects a change in EIA leads. It does not establish which lead(s) has changed. Two associated variables, current_eia_leads and previous_eia_leads, indicate the status of the seven leads. These are two-byte (short) variables. Each lead is represented by a different bit in the short. Some bits are unused. Table 60-1 lists the mask that can be used to isolate each lead.

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, RTS for example, while disregarding its status, enter the following condition on the Protocol Spreadsheet:

CONDITIONS:
{
  fevar_eia_changed && (((current_eia_leads ^ previous_eia_leads) & 0x80) == 0x80)
}

Select a mask value from the list in Table 60-1 to indicate which lead you care about. Specify multiple leads with a mask derived via hexadecimal addition.
Table 60-1
EIA Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern fast_event</td>
<td>fevar_eia_changed</td>
<td></td>
<td>True when the status changes for an EIA lead (non-data). Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern const volatile unsigned short</td>
<td>current_eia_leads</td>
<td>4</td>
<td>RS-232/V.35: (RS-449)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>RI (IC)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10/16</td>
<td>DSR (DM)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20/32</td>
<td>DTR (TR)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>40/64</td>
<td>CD (RR)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>80/128</td>
<td>CTS (CS)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>200/512</td>
<td>RTS (RS)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>A value in this list, when anded (&amp;) with current_eia_leads, equals zero if the lead is on. Example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>STATE: rts_on { if ((current_eia_leads &amp; 0x80) == 0) sound_alarm(); }</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Note: This variable will store EIA status if (1) internal or external clock is supplied and (2) EIA leads are enabled on FEB Setup.</td>
</tr>
<tr>
<td></td>
<td>previous_eia_leads</td>
<td></td>
<td>Same values as current_eia_leads. Updated only after logic has had a chance to compare current and previous leads. Line Setup configured for emulate or monitor mode.</td>
</tr>
</tbody>
</table>

The mask for RTS is 0x80. In the example, the event fevar_eia_changed updated current_eia_leads. The new current_eia_leads was bitwise-exclusive-ORed with previous_eia_leads to identify all the leads that changed. Then the result was bitwise ANDed with the RTS mask to determine if RTS was among the leads that changed. If this result was equal to the mask, the lead changed.

(B) Masking For the Status of a Lead

You may also test the current status of a lead, independent of any change. **And** the mask with current_eia_leads, as in this if statement testing for RTS “on”:

```
STATE: test_for_rts_on
{
   if((current_eia_leads & 0x80) == 0) sound_alarm();
}
```
If the result is zero, the lead is on. If the result equals the mask, the lead is off. "On" means that a lead is more positive than +3 volts with respect to signal ground. "Off" implies only that a lead is not at or above the "on" threshold, not necessarily that a minus threshold has been attained.

(C) Detect Change and Current Status

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

CONDITIONS:
{
    (evar_eia_changed && (((current_eia_leads ^ previous_eia_leads) & 0x80) == 0x80) &&
    ((current_eia_leads & 0x80) == 0))
}

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

60.2 Routines

You may control RS-232 EIA leads in emulate mode only. When the Line Setup menu shows Mode: EMULATE DCE, you control CTS, CD, and DSR. An EMULATE DTE selection gives you control over RTS and DTR. Entries on the Interface Control menu may be used to set the leads' initial status (Section 10.6).

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 up to three of nine possible leads. Which leads you control depends on your emulation mode. The softkey equivalent of this routine is the EIA action on the Protocol Spreadsheet.

Inputs

The first parameter indicates which leads you want to turn on. Each bit in the parameter controls a given lead: RTS/CTS (01), DTR/DSR (02), CD (04), AUX0 (10), AUX1 (20), AUX2 (40), AUX3 (80). 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 Oxff will mean don’t care (no
action).

The second parameter indicates which leads you want in the “off” condition. Each
bit in the parameter controls a given lead: RTS/CTS (01), DTR/DSR (02), CD (04),
AUX0 (10), AUX1 (20), AUX2 (40), AUX3 (80). 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 CTS, DSR,
and CD leads. (An attempt to control the status of RTS or DTR will fail, since the
DTE controls these leads.) When RTS is raised, you want to turn CTS on; when RTS
drops, turn CTS off.

LAYER: 1
STATE: control cts
   CONDITIONS: EIA RTS ON
   ACTIONS:
   {   
      ctl_eia(0xfe, 0x00);
   }
   CONDITIONS: EIA RTS OFF
   ACTIONS:
   {   
      ctl_eia(0xff, 0x01);
   }
61 Display Window and Trace

The C structures, variables, and routines detailed in this section control the type and location of certain displays on the INTERVIEW. These displays can be grouped into three categories.

The first display area is the prompt line, the second line on all Run-mode screens.

The second type of display utilizes the Display Window, available as a selection on the Display Setup portion of the Line Setup menu, or conditionally accessible via softkey during Run mode. To write to the Display Window, use the pos_cursor (or restore_cursor) and displayc, displayf, or displays routines. When using Display Window, you may position the cursor before output is generated on the screen.

The third type of display utilizes one or a combination of the eight available trace buffers. Trace screens are conditionally accessible via softkey during Run mode. Seven user-traces appear as choices under the User Trace selection on the Display Setup menu. The remaining trace is Program Trace, also an option on Display Setup. Program Trace enables you to track any or all layers, one or all tests, and movement between states. To write to any of the eight trace-screens, use the tracec, tracef, and traces routines.

NOTE: You may not use the pos_cursor routine to position the cursor on any trace screen. New lines (or blank lines) may be generated via the "\n" specifier.

Attributes—color, underlining, and font, for example—may be assigned to characters in the Display Window and all of the Trace buffers.

NOTE: Color attributes are applied to the RGB output signal, not to the plasma screen.

61.1 Current Display Mode

A group of variables keeps track of softkey movement from one display screen to another (see Table 61-1). When you move from the Display Window to the Program Trace screen, for example, the fast-event variable display_screen_changed indicates the change of display. The coded value for Display Window now is stored in prev_display_screen, and the value for Program Trace can be found in crnt_display_screen.
These variables also distinguish between Run mode and Freeze mode. This distinction is important since some keys on the keyboard are mode-dependent. In Freeze mode, for instance, cursor keys automatically become operational for scrolling through the buffer. The programmer will want to avoid using these keys as user-input when `crnt_display_screen` indicates that the unit is in Freeze mode.

### Table 61-1
Current Display Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern fast_event</td>
<td>display_screen_changed</td>
<td></td>
<td>True when Run-mode display-screen is changed, or when Run/Freeze mode is changed. Value in <code>crnt_display_screen</code> is stored in <code>prev_display_screen</code>, and <code>crnt_display-screen</code> is updated. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern unsigned short</td>
<td>crnt_display_screen</td>
<td></td>
<td>Contains current display screen (low byte) and indicates whether unit is in Run mode or Freeze mode (high byte). Line Setup configured for emulate or monitor mode.</td>
</tr>
</tbody>
</table>

**display-screen**

- 0: no display
- 1: single-line data
- 2: dual-line data
- 3: single-line data with leads
- 4: dual-line data with leads
- 11/17: tabular statistics
- 12/18: graphic statistics
- 21/33: Display Window
- 31/49: Program Trace
- 41/65: Layer 1 Protocol Trace
- 42/66: Layer 2 Protocol Trace
- 43/67: Layer 3 Protocol Trace
- 44/68: Layer 4 Protocol Trace
- 45/69: Layer 5 Protocol Trace
- 46/70: Layer 6 Protocol Trace
- 47/71: Layer 7 Protocol Trace
- 51/81: User Trace 1
- 52/82: User Trace 2
- 53/83: User Trace 3
- 54/84: User Trace 4
- 55/85: User Trace 5
- 56/86: User Trace 6
- 57/87: User Trace 7
- 61/97: TIM package standard stats
- 62/98: TIM package aux
- 100/256: Run/Freeze mode (bit 9)
- 0: Freeze mode
- 0: Run mode

61-2
Table 61-1 (continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern unsigned short</td>
<td>prev_display_screen</td>
<td>Contains previous display screen (low byte) and indicates whether</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>unit was in Run mode or Freeze mode (high byte). Line Setup configured</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>for emulate or monitor mode.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>display-screen</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>single-line data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>dual-line data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>single-line data with leads</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>dual-line data with leads</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11/17</td>
<td>tabular statistics</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12/18</td>
<td>graphic statistics</td>
</tr>
<tr>
<td></td>
<td></td>
<td>21/33</td>
<td>Display Window</td>
</tr>
<tr>
<td></td>
<td></td>
<td>31/49</td>
<td>Program Trace</td>
</tr>
<tr>
<td></td>
<td></td>
<td>41/65</td>
<td>Layer 1 Protocol Trace</td>
</tr>
<tr>
<td></td>
<td></td>
<td>42/66</td>
<td>Layer 2 Protocol Trace</td>
</tr>
<tr>
<td></td>
<td></td>
<td>43/67</td>
<td>Layer 3 Protocol Trace</td>
</tr>
<tr>
<td></td>
<td></td>
<td>44/68</td>
<td>Layer 4 Protocol Trace</td>
</tr>
<tr>
<td></td>
<td></td>
<td>45/69</td>
<td>Layer 5 Protocol Trace</td>
</tr>
<tr>
<td></td>
<td></td>
<td>46/70</td>
<td>Layer 6 Protocol Trace</td>
</tr>
<tr>
<td></td>
<td></td>
<td>47/71</td>
<td>Layer 7 Protocol Trace</td>
</tr>
<tr>
<td></td>
<td></td>
<td>51/81</td>
<td>User Trace 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>52/82</td>
<td>User Trace 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>53/83</td>
<td>User Trace 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>54/84</td>
<td>User Trace 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>55/85</td>
<td>User Trace 5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>56/86</td>
<td>User Trace 6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>57/87</td>
<td>User Trace 7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>61/97</td>
<td>TIM package standard stats</td>
</tr>
<tr>
<td></td>
<td></td>
<td>62/98</td>
<td>TIM package aux</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100/256</td>
<td>Freeze mode (bit 9)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Run mode</td>
</tr>
</tbody>
</table>

61.2 Prompt Line

Access to the prompt line is always available via the display_prompt routine, or its softkey equivalent, the PROMPT action. Attributes may not be assigned to a prompt created via either of these methods. (To create a prompt with attributes, use the pos_cursor and displayf routines.) Prompts appear on whatever screen is active at the time the prompt is written, including trace screens. With one exception, movement to another display erases the prompt. The only screen which retains the most recent prompt is the Display Window.

You may also position the cursor to the prompt line in the Display Window via the pos_cursor routine. The initial position of the cursor in the Display Window is at the beginning of the prompt line—row zero, column zero. Anything written to this cursor
position in the Display Window will appear as a prompt on any one of the other
display screens (assuming one of them is active at the time the message is written). Position the cursor below the prompt line for messages intended for the Display Window only.

Trace buffers retain no record of prompts. When you write to a trace screen, the initial position of the cursor is the line immediately below the prompt line—row one. Since you may not position the cursor in trace buffers, all messages written to trace buffers are appended at the end of the buffer. You may never access the prompt line via tracef (or tracec or traces) routines.

61.3 Display Window

The Display Window preserves one screen, including the prompt line, of user-entered messages. When the end of the display screen is reached, the previous messages are overwritten, beginning at row one (the line below the prompt line).

NOTE: Use the keyboard variables and the send_key routine explained in Section 69, Other Library Tools, to program the Run-mode use of ▲ and ▼ in the Display Window. (For other Run-mode screens, these keys control the playback speed of disk data.)

(A) Variables

There are variables accessible to the user which provide information about the Display Window. Table 61-2 lists the variables and their possible values. Two variables indicate the current position of the cursor: current_line stores the row number and current_col stores the column number. To find out which attributes are active in the Display Window, check the values stored in window_color and window_modifier. Color is stored in the high byte of the two-byte variable window_color. Enhancements are stored in the low byte. The current font code can be found in window_modifier.

NOTE: Attributes assigned via the %m conversion specifier (refer to tracef-routine input) to characters in trace buffers will not alter the values of window_color and window_modifier. These variables refer to the Display Window only.

The variable display_window_buffer provides the user with access to the display-window buffer. This variable is an array of 1,088 longs. Each element in the array contains one byte of character data and three bytes of attributes. The attributes are determined by the current values of window_color and window_modifier.
### Display Window Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern unsigned short</td>
<td>current_line</td>
<td>0-10/0-16</td>
<td>Contains the current row number of the cursor position in the Display Window. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern unsigned short</td>
<td>current_col</td>
<td>0-31/0-63</td>
<td>Contains the current column number of the cursor position in the Display Window. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern unsigned short</td>
<td>window_color</td>
<td></td>
<td>Two-byte variable. Current color selections are indicated in the low byte. Current enhancements are indicated in the high byte. Written to by %m conversions. Attributes are copied into data words in Display Window. Line Setup configured for emulate or monitor mode.</td>
</tr>
</tbody>
</table>

Isolate bits of interest via bitwise anding (&) of mask with variable. Compare result to value column. For example, underline attribute mask = 0x100. Therefore `window_color & 0x100 equals 0 (underline off)` or `0x100 (underline on)`. Line Setup configured for emulate or monitor mode.

#### background color mask = 7 (bits 1-3):

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>black</td>
</tr>
<tr>
<td>1</td>
<td>blue</td>
</tr>
<tr>
<td>2</td>
<td>green</td>
</tr>
<tr>
<td>3</td>
<td>cyan</td>
</tr>
<tr>
<td>4</td>
<td>red</td>
</tr>
<tr>
<td>5</td>
<td>magenta</td>
</tr>
<tr>
<td>6</td>
<td>yellow</td>
</tr>
<tr>
<td>7</td>
<td>white</td>
</tr>
</tbody>
</table>

#### foreground color mask = 0x38 (bits 4-6):

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>black</td>
</tr>
<tr>
<td>8</td>
<td>blue</td>
</tr>
<tr>
<td>10/16</td>
<td>green</td>
</tr>
<tr>
<td>16/24</td>
<td>cyan</td>
</tr>
<tr>
<td>20/32</td>
<td>red</td>
</tr>
<tr>
<td>28/40</td>
<td>magenta</td>
</tr>
<tr>
<td>30/48</td>
<td>yellow</td>
</tr>
<tr>
<td>38/56</td>
<td>white</td>
</tr>
</tbody>
</table>
Table 61-2 (continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(window_color continued)</td>
<td>color blink mask = 0x40 (bit 7):</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>no blink</td>
<td>blink</td>
<td></td>
</tr>
<tr>
<td>40/64</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>no strike-thru</td>
<td>strike-thru</td>
<td></td>
</tr>
<tr>
<td>80/128</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>no overline</td>
<td>overline</td>
<td></td>
</tr>
<tr>
<td>100/256</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>no blank</td>
<td>blank</td>
<td></td>
</tr>
<tr>
<td>200/512</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>no underline</td>
<td>underline</td>
<td></td>
</tr>
<tr>
<td>400/1024</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>no reverse image</td>
<td>reverse image</td>
<td></td>
</tr>
<tr>
<td>800/2048</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>no hex</td>
<td>hex</td>
<td></td>
</tr>
<tr>
<td>1000/4096</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>no low intensity</td>
<td>low intensity (RS-170 output)</td>
<td></td>
</tr>
<tr>
<td>2000/8192</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>no monochrome blink</td>
<td>monochrome blink</td>
<td></td>
</tr>
<tr>
<td>4000/16384</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>no monochrome strike-thru</td>
<td>monochrome strike-thru</td>
<td></td>
</tr>
<tr>
<td>8000/32768</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Type | Variable | Value (hex/decimal) | Meaning
--- | --- | --- | ---
extern unsigned char | window_modifier | | Contains the current modifiers. Line Setup configured for emulate or monitor mode. *font mask = 7 (bits 1-3):*
| | | | ASCII
| | | special graphic character set (refer to Table 61-5)
| | 1 | | primary font—code selected on Line Setup
| | 2 | | alternate font—current implementation is for call-setup phase in X.21 (ASCII) hexadecimal
| | 7 | | 0
extern unsigned long | display_window_buffer [1088] | Array of 32-bit words that make up the one-screen Display Window. Each word contains three bytes of attributes and a one-byte character. Refer to Table 61-4. Line Setup configured for emulate or monitor mode.

(B) Structures

Once the data word is written to the buffer as an element in the `display_window_buffer` array, it can be accessed and written to—and therefore changed—the same as any other location in memory. There is an `extern` array, `display_window_index_buffer[17]`, which provides a method of informing the display controller on the CPM card that the display needs to be updated. The structure of this array is shown in Table 61-3.

Each element in the `display_window_index_buffer` array represents a horizontal row or line in the Display Window. Each element is a structure with two variables, `mpm` and `cpm`. The first variable in the structure, `mpm`, increments automatically whenever a line in the display-window buffer is updated by a display routine. (If you write to the buffer directly without using one of the display routines, you must increment this variable "manually.") Its particular value at any moment is not important. What is significant is whether or not the value of the second variable in the structure, `cpm`, is the same as `mpm`. The processor on the CPM compares these two variables (for each line) periodically to determine if a line in the Display Window needs to be rewritten. If the values of the two variables do not match, it means that a line updated in memory now needs to be updated by the CPM display-controller software. After the display is changed, the value of `mpm` is copied automatically into `cpm`.

61-7
Table 61-3
Display Window Buffer Structures

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structure</td>
<td>Name: display_window_index_buffer [17]</td>
<td>An array of structures used for detecting changes to the display-window buffer. There are seventeen elements in the array, one for each line in the Display Window. When a change is made to a line in the display-window buffer, the corresponding element in the array is accessed. If a display routine changes line 3, \text{display_window_index_buffer}[3] \cdot mpm is automatically incremented. The CPM detects the difference between \text{display_window_index_buffer}[3] \cdot mpm and \text{display_window_index_buffer}[3] \cdot cpm and updates line 3 in the Display Window. Declared as type \text{extern struct.}</td>
<td></td>
</tr>
<tr>
<td>unsigned char</td>
<td>mpm</td>
<td>0-ff/0-255</td>
<td>When the MPM updates a line in the display-window buffer, this variable is incremented.</td>
</tr>
<tr>
<td>unsigned char</td>
<td>cpm</td>
<td>0-ff/0-255</td>
<td>The CPM checks the value of this variable against the value of mpm. If they are different, the value in mpm is copied into cpm. The updated line in MPM is then presented on the display-window screen.</td>
</tr>
</tbody>
</table>

(C) Routines

You may position the cursor before output is generated on the screen via the pos_cursor and restore_cursor routines. The pos_cursor routine positions the cursor at the row and column you specify. The restore_cursor routine returns the cursor to a previous location. One routine, display, allows you to add attributes to messages in the Display Window, including the prompt line. These attributes are listed in Table 61-4.
displayc

Synopsis

extern void displayc(character);
const char character;

Description

The displayc routine outputs a single ASCII character to the Display Window screen. The placement of the output on the screen may be controlled via the pos_cursor routine. Attributes may not be used in displayc.

Inputs

The parameter value may be given as a hexadecimal, octal, or decimal constant; as an alphanumeric constant inside of single quotes; or as a variable. A hexadecimal value must be preceded by the prefix 0x or 0X; an octal value must be preceded by the prefix 0. If no prefix appears before the input, the number is assumed to be decimal. Valid numeric entries are 00 to 127, decimal. An alphanumeric character placed between single quotes will be output as is to the display.

Example

The displayc entries on the left output the character given on the right, at the cursor location on the Display Window screen:

displayc('a');   a
displayc(65);    A
displayc(0x65);  e
displayc(065);   5

displayf

Synopsis

extern int displayf(format_ptr, . . .);
const char * format_ptr;

Description

The displayf routine writes output to the Display Window screen, under control of the string pointed to by format_ptr that specifies how subsequent arguments are converted for output. If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated but otherwise ignored. The displayf routine returns when the end of the format string is encountered. The placement of the output on the screen may be controlled via the pos_cursor routine.
Inputs

The format is composed of zero or more directives: ordinary characters (not %), which are copied unchanged to the output stream; and conversion specifications, each of which results in fetching zero or more subsequent arguments. Each conversion specification is introduced by the character %. After the %, the following appear in sequence:

- Zero or more flags that modify the meaning of the conversion specification. The flag characters and their meanings are:
  - The result of the conversion will be left-justified within the field.
  - The result of a signed conversion will always begin with a plus or minus sign.

  space If the first character of a signed conversion is not a sign, a space will be prepended to the result. If the space and + flags both appear, the space flag will be ignored.

  # The result is to be converted to an “alternate form.” For d and i conversions, the flag has no effect. For o conversion, it increases the precision to force the first digit of the result to be a zero. For x (or X) conversion, a nonzero result will have 0x (or 0X) prepended to it. For u conversions, the argument is displayed in small hex characters. For example, displayf (uo/o#u", 258); yields °1 0 Z.

- An optional decimal integer specifying a minimum field width. If the converted value has fewer characters than the field width, it will be padded on the left (or right, if the left adjustment flag, described above, has been given) to the field width. The padding is with spaces unless the field width integer starts with a zero, in which case the padding is with zeros.

- An optional precision that gives the minimum number of digits to appear for the d, i, o, u, x, and X conversions, the maximum number of characters to be written from an array in an s conversion, or the number of characters to be written from an array in an H conversion (overriding the usual null-termination of strings). The precision takes the form of a period (.) followed by an optional decimal integer; if the integer is omitted, it is treated as zero. The amount of padding specified by the precision overrides that specified by the field width.
• An optional h specifying that a following d, i, o, u, x, or X conversion specifier applies to a short int or unsigned short int argument (the argument will have been promoted according to the integral promotions, and its value shall be converted to short int or unsigned short int before printing); or an optional l specifying that a following d, i, o, u, x, or X conversion specifier applies to a long int or unsigned long int argument. If an h or l appears with any other conversion specifier, it is ignored.

• A character that specifies the type of conversion to be applied. (Special AR extensions have been added.) The conversion specifiers and their meanings are:

d, i, o, u, x, X

The int argument is converted to signed decimal (d or i), unsigned octal (o), unsigned decimal (u), or unsigned hexadecimal notation (x or X); the letters abedef are used for x conversion and the letters ABCDEF for X conversion. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeros. The default precision is 1. The result of converting a zero value with a precision of zero is no characters.

c
The int argument is converted to an unsigned char, and the resulting character is written.

s
The argument shall be a pointer to a null-terminated array of 8-bit chars. Characters from the string are written up to (but not including) the terminating null character: if the precision is specified, no more than that many characters are written. The string may be an array into which output was written via the sprintf routine. (If the string pointed to is an array which has been written via the stracef routine, you must use %b rather than %s to display it.)

p
The argument shall be a pointer to void. The value of the pointer is converted to a sequence of printable characters, in this format: 0000:0000. There are always exactly 4 digits to the right of the colon. The number of digits to the left of the colon is determined by the pointer's value and the precision specified. Use this conversion to display 80286 memory addresses. The 16-bit segment number will appear to the left of the colon and the 16-bit offset to the right.

% A % is written. No argument is converted.

\n Displays \f. No argument is converted.

H displays a character array (pointed to by the argument) as small hex characters. If precision is specified, it is used as the length of the array (overriding the usual null-termination of strings).
b The argument shall be a pointer to an array of 32-bit words. Characters from the string are written up to (but not including) the terminating word containing a null character: if the precision is specified, no more than that many words are written. If the string pointed to is an array into which output was written via the \texttt{stracef} routine, you must use \texttt{%b} rather than \texttt{%s} to display it. (To display the information in an array written to via \texttt{printf}, use \texttt{%s}.)

m The argument is a \texttt{long} integer that indicates attributes to be assigned to subsequent characters. Attributes stay “on” until they are specifically turned “off” with another \texttt{%m} conversion specifier. The lowest-order byte contains primarily font code. The next higher byte is not used to set attributes. (In the display-window buffer, this second byte is reserved for character coding.) The third byte holds color information. The high byte indicates which enhancements should be invoked.

Attributes are written automatically to \texttt{window\_color} and \texttt{window\_modifier} variables, then copied into subsequent 32-bit data words in the Display Window. Table 61-4 shows the format of this 32-bit word.

Attributes may not be assigned as a one-byte value. Even if you want to alter only one attribute setting, color for example, you must include it as part of a \texttt{long}. Append an “L” at the end of the hexadecimal code specifying attributes to indicate the value is a \texttt{long}.

NOTE: If you are specifying an attribute in a lower-order byte of the \texttt{long}, color for example, and you want the high byte (or bytes) to be zero, you may omit the high byte provided you have the “L” appended at the end of the hexadecimal code. The high byte (or bytes) will be left-padded with zeroes. For example, \texttt{0x200000L} is converted to \texttt{0x000200000L}. Associated characters will be displayed on a color monitor as green on a black background, as dictated by the hexadecimal 20 in the third byte. Enhancements are controlled in the high byte, now assigned a value of zero. Any enhancements previously turned “on” will be turned “off.”

If a conversion specification is invalid, the behavior is undefined.

If any argument is or points to an aggregate (except for an array of characters using \texttt{%s} conversion or any pointer using \texttt{%p} conversion), the behavior is undefined.

In no case does a nonexistent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is expanded to contain the conversion result.
Returns

The `displayf` routine returns the number of characters displayed.

Example

To display a date and time in the form “Sunday, July 3, 10:02,” where weekday and month are pointers to strings:

```
LAYER: 1
{
  unsigned char weekday [10];
  unsigned char month [10];
  unsigned short day;
  unsigned char hour;
  unsigned char min;
}

STATE: output_to_display_window
CONDITIONS: KEYBOARD • *
ACTIONS:
{
  displayf( "%s, %s %d, %.2d:%.2d\n", weekday, month, day, hour, min);
}
```

`sprintf`

The `sprintf` routine is similar to the `displayf` routine. `displayf` writes output with or without attributes directly to the Display Window. `sprintf`, fully documented in Section 64.3, writes output to a character array in which attributes are not supported. This routine is useful for writing formatted output to a display, printer, or file.

See also `stracef` in Section 61.4(C).
### Table 61-4

Display Window/Trace Buffer 32-Bit Data Word

<table>
<thead>
<tr>
<th>Bit</th>
<th>Mask (hex) †</th>
<th>Input (hex) † †</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-3</td>
<td>00000007L</td>
<td>00000000L</td>
<td>Modifier attributes, font for example, are contained in the low byte of the 32-bit word. Font: ASCII special graphic character set (refer to Table 61-5). primary font—code selected on Line Setup. alternate font—current implementation is for call-setup phase in X.21 (ASCII) hexadecimal.</td>
</tr>
<tr>
<td>4</td>
<td>00000008L</td>
<td>00000000L</td>
<td>Special character indicator: only value in modifier in trace buffer header. Character is not displayable but contains control info used internally by the trace logic. When a &quot;\n&quot; is included in a tracef routine, for example, a new line is generated, but nothing is displayed on the trace screen. The tracef routine automatically sets this bit before the 32-bit word is written into <code>trace_buf.array</code>.</td>
</tr>
<tr>
<td>5-8</td>
<td>000000001L</td>
<td>00000000L</td>
<td>unused, but should be zero.</td>
</tr>
<tr>
<td>9-16</td>
<td>000000000L</td>
<td>00000000L</td>
<td>Character data is contained in the second byte of the <code>long</code> word. Input should be 00 in all <code>%m</code> conversions.</td>
</tr>
</tbody>
</table>

† Use the masks to change attributes of characters in the Display Window or trace buffer. In the Display Window, characters are represented in the second byte of the `longs` that comprise the 1,088 array elements in `display_window_buffer`. In the `trace_buf` structure, the characters are represented in the second byte of the `longs` that make up the `trace_buf.array`. To change one attribute of a character while leaving the others unchanged:

\[
\text{display_window_buffer[position]} = \left(\text{display_window_buffer[position]} \& \text{(~attribute-mask)}\right) \mid \text{input};
\]

To change only the font of the twenty-first character in the trace buffer from its current setting to the special graphic font, for example:

\[
\text{i2_trbuf.array[20]} = \left(\text{trace_buf.array[20]} \& \text{(~0x0000007L)}\right) \mid \text{0x00000001L};
\]

Anding the character with the mask will indicate the current setting of an attribute:

If `(i2_trbuf.array[20] & 0x00000007L)` equals 2, then the 21st character in the Trace 2 user–trace buffer is being displayed in the font selected on the Line Setup menu.

†† In displayf routines, the `%m` conversion specifier writes input to the `window_color` and `window Modifier` variables. These variables are copied into subsequent data words in the Display Window. In tracef routines, the `%m` conversion specifier writes input to `trace_buffer_header`. The header is then copied into each subsequent data word in the buffer. Combine attributes via hexadecimal addition.
Table 61-4 (continued)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Mask (hex)</th>
<th>Input (hex)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>17-19</td>
<td>00070000L</td>
<td>00000000L</td>
<td>Color is contained in the third byte of the long. Combine color attributes via hexadecimal addition.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00010000L</td>
<td>Background color:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00020000L</td>
<td>black</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00030000L</td>
<td>blue</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00040000L</td>
<td>green</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00050000L</td>
<td>cyan</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00060000L</td>
<td>red</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00070000L</td>
<td>magenta</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00080000L</td>
<td>yellow</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00090000L</td>
<td>white</td>
</tr>
<tr>
<td>20-22</td>
<td>00380000L</td>
<td>00000000L</td>
<td>Foreground color:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00080000L</td>
<td>black</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00100000L</td>
<td>blue</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00180000L</td>
<td>green</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00200000L</td>
<td>cyan</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00280000L</td>
<td>red</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00300000L</td>
<td>magenta</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00380000L</td>
<td>yellow</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00300000L</td>
<td>white</td>
</tr>
<tr>
<td>23</td>
<td>00400000L</td>
<td>00000000L</td>
<td>Color blink:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00400000L</td>
<td>no blink</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00400000L</td>
<td>blink</td>
</tr>
<tr>
<td>24</td>
<td>00800000L</td>
<td>00000000L</td>
<td>Color strike-thru:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00800000L</td>
<td>no strike-thru</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00800000L</td>
<td>strike-thru</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00800000L</td>
<td>Enhance attributes, underlining for example, are contained in the high byte of the long. Combine enhancements via hexadecimal addition.</td>
</tr>
<tr>
<td>25</td>
<td>01000000L</td>
<td>00000000L</td>
<td>Overline:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00000000L</td>
<td>(for monochrome and color)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>01000000L</td>
<td>no overline</td>
</tr>
<tr>
<td></td>
<td></td>
<td>01000000L</td>
<td>overline</td>
</tr>
<tr>
<td>26</td>
<td>02000000L</td>
<td>00000000L</td>
<td>Blank:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>02000000L</td>
<td>monochrome display, color display monochrome no display, color display</td>
</tr>
<tr>
<td>27</td>
<td>04000000L</td>
<td>00000000L</td>
<td>Underline:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>04000000L</td>
<td>(for monochrome and color)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00000000L</td>
<td>no underline</td>
</tr>
<tr>
<td></td>
<td></td>
<td>04000000L</td>
<td>underline</td>
</tr>
</tbody>
</table>
## Table 61-4 (continued)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Mask (hex)</th>
<th>Input (hex)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>08000000L</td>
<td>00000000L</td>
<td>Monochrome reverse image: no reverse image</td>
</tr>
<tr>
<td></td>
<td></td>
<td>08000000L</td>
<td>reverse image</td>
</tr>
<tr>
<td>29</td>
<td>10000000L</td>
<td>00000000L</td>
<td>Hex: no hex</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10000000L</td>
<td>hex</td>
</tr>
<tr>
<td>30</td>
<td>20000000L</td>
<td>00000000L</td>
<td>Monochrome low intensity: no low intensity</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20000000L</td>
<td>low intensity (RS-170 interface)</td>
</tr>
<tr>
<td>31</td>
<td>40000000L</td>
<td>00000000L</td>
<td>Monochrome blink: no blink</td>
</tr>
<tr>
<td></td>
<td></td>
<td>40000000L</td>
<td>blink</td>
</tr>
<tr>
<td>32</td>
<td>80000000L</td>
<td>00000000L</td>
<td>Monochrome strike-thru: no strike-thru</td>
</tr>
<tr>
<td></td>
<td></td>
<td>80000000L</td>
<td>strike-thru</td>
</tr>
</tbody>
</table>
Table 61-5
Special Graphic Character Set†

<table>
<thead>
<tr>
<th></th>
<th>Display Input (hex/decimal)</th>
<th>Display</th>
<th>Input (hex/decimal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>🆔</td>
<td>0</td>
<td>🆔</td>
<td>1a/26</td>
</tr>
<tr>
<td>🆕</td>
<td>1</td>
<td>🆕</td>
<td>1b/27</td>
</tr>
<tr>
<td>🆒</td>
<td>2</td>
<td>🆒</td>
<td>1c/28</td>
</tr>
<tr>
<td>🆓</td>
<td>3</td>
<td>🆓</td>
<td>1d/29</td>
</tr>
<tr>
<td>🆍</td>
<td>4</td>
<td>🆍</td>
<td>1e/30</td>
</tr>
<tr>
<td>🆕</td>
<td>5</td>
<td>🆕</td>
<td>1f/31</td>
</tr>
<tr>
<td>🆖</td>
<td>6</td>
<td>🆖</td>
<td>20/32</td>
</tr>
<tr>
<td>🆗</td>
<td>7</td>
<td>🆗</td>
<td>21/33</td>
</tr>
<tr>
<td>🆘</td>
<td>8</td>
<td>🆘</td>
<td>22/34</td>
</tr>
<tr>
<td>🆙</td>
<td>9</td>
<td>🆙</td>
<td>23/35</td>
</tr>
<tr>
<td>🆚</td>
<td>a/10</td>
<td>🆚</td>
<td>24/36</td>
</tr>
<tr>
<td>🆛</td>
<td>b/11</td>
<td>🆛</td>
<td>25/37</td>
</tr>
<tr>
<td>🆜</td>
<td>c/12</td>
<td>🆜</td>
<td>26/38</td>
</tr>
<tr>
<td>🆝</td>
<td>d,11/13,17</td>
<td>🆝</td>
<td>27/39</td>
</tr>
<tr>
<td>🆞</td>
<td>e/14</td>
<td>🆞</td>
<td>28/40</td>
</tr>
<tr>
<td>🆟</td>
<td>f/15</td>
<td>🆟</td>
<td>29/41</td>
</tr>
<tr>
<td>🆟</td>
<td>10/16</td>
<td>🆟</td>
<td>2a/42</td>
</tr>
<tr>
<td>🆟</td>
<td>12/18</td>
<td>🆟</td>
<td>2b/43</td>
</tr>
<tr>
<td>🆟</td>
<td>13/19</td>
<td>🆟</td>
<td>2c/44</td>
</tr>
<tr>
<td>🆟</td>
<td>14/20</td>
<td>🆟</td>
<td>2d/45</td>
</tr>
<tr>
<td>🆟</td>
<td>15/21</td>
<td>🆟</td>
<td>2e/46</td>
</tr>
<tr>
<td>🆟</td>
<td>16/22</td>
<td>🆟</td>
<td>2f/47</td>
</tr>
<tr>
<td>🆟</td>
<td>17/23</td>
<td>🆟</td>
<td>30/48</td>
</tr>
<tr>
<td>🆟</td>
<td>18/24</td>
<td>🆟</td>
<td>31/49</td>
</tr>
<tr>
<td>🆟</td>
<td>19/25</td>
<td>🆟</td>
<td>(space)</td>
</tr>
</tbody>
</table>

† Written to the Display Window or a trace buffer when low (modifier) byte of 32-bit data word = 0x01.
**Table 61-5 (continued)**

<table>
<thead>
<tr>
<th>Display</th>
<th>Input (hex/decimal)</th>
<th>Display</th>
<th>Input (hex/decimal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>¥</td>
<td>80/128</td>
<td>ropri</td>
<td>9a/154</td>
</tr>
<tr>
<td>･</td>
<td>81/129</td>
<td>サ</td>
<td>9b/155</td>
</tr>
<tr>
<td>･</td>
<td>82/130</td>
<td>シ</td>
<td>9c/156</td>
</tr>
<tr>
<td>･</td>
<td>83/131</td>
<td>ス</td>
<td>9d/157</td>
</tr>
<tr>
<td>､</td>
<td>84/132</td>
<td>セ</td>
<td>9e/158</td>
</tr>
<tr>
<td>･</td>
<td>85/133</td>
<td>ソ</td>
<td>9f/159</td>
</tr>
<tr>
<td>ヿ</td>
<td>86/134</td>
<td>タ</td>
<td>a0/160</td>
</tr>
<tr>
<td>ナ</td>
<td>87/135</td>
<td>チ</td>
<td>a1/161</td>
</tr>
<tr>
<td>フ</td>
<td>88/136</td>
<td>ツ</td>
<td>a2/162</td>
</tr>
<tr>
<td>ハ</td>
<td>89/137</td>
<td>テ</td>
<td>a3/163</td>
</tr>
<tr>
<td>ヒ</td>
<td>8a/138</td>
<td>ト</td>
<td>a4/164</td>
</tr>
<tr>
<td>ナ</td>
<td>8b/139</td>
<td>ナ</td>
<td>a5/165</td>
</tr>
<tr>
<td>ニ</td>
<td>8c/140</td>
<td>ニ</td>
<td>a6/166</td>
</tr>
<tr>
<td>ヌ</td>
<td>8d/141</td>
<td>ヌ</td>
<td>a7/167</td>
</tr>
<tr>
<td>ネ</td>
<td>8e/142</td>
<td>ネ</td>
<td>a8/168</td>
</tr>
<tr>
<td>ノ</td>
<td>8f/143</td>
<td>ノ</td>
<td>a9/169</td>
</tr>
<tr>
<td>ハ</td>
<td>90/144</td>
<td>ハ</td>
<td>aa/170</td>
</tr>
<tr>
<td>ヒ</td>
<td>91/145</td>
<td>ヒ</td>
<td>ab/171</td>
</tr>
<tr>
<td>フ</td>
<td>92/146</td>
<td>フ</td>
<td>ac/172</td>
</tr>
<tr>
<td>ヘ</td>
<td>93/147</td>
<td>ヘ</td>
<td>ad/173</td>
</tr>
<tr>
<td>ベ</td>
<td>94/148</td>
<td>ベ</td>
<td>ae/174</td>
</tr>
<tr>
<td>マ</td>
<td>95/149</td>
<td>マ</td>
<td>af/175</td>
</tr>
<tr>
<td>ミ</td>
<td>96/150</td>
<td>ミ</td>
<td>b0/176</td>
</tr>
<tr>
<td>ム</td>
<td>97/151</td>
<td>ム</td>
<td>b1/177</td>
</tr>
<tr>
<td>メ</td>
<td>98/152</td>
<td>メ</td>
<td>b2/178</td>
</tr>
<tr>
<td>モ</td>
<td>99/153</td>
<td>モ</td>
<td>b3/179</td>
</tr>
</tbody>
</table>
### Table 61-5 (continued)

<table>
<thead>
<tr>
<th>Display</th>
<th>Input (hex/decimal)</th>
<th>Display</th>
<th>Input (hex/decimal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>タ</td>
<td>b4/180</td>
<td>タ</td>
<td>ce/206</td>
</tr>
<tr>
<td>カ</td>
<td>b5/181</td>
<td>カ</td>
<td>cf/207</td>
</tr>
<tr>
<td>ユ</td>
<td>b6/182</td>
<td>ユ</td>
<td>d0/208</td>
</tr>
<tr>
<td>ル</td>
<td>b7/183</td>
<td>ル</td>
<td>d1/209</td>
</tr>
<tr>
<td>リ</td>
<td>b8/184</td>
<td>リ</td>
<td>d2/210</td>
</tr>
<tr>
<td>ル</td>
<td>b9/185</td>
<td>ル</td>
<td>d3/211</td>
</tr>
<tr>
<td>レ</td>
<td>ba/186</td>
<td>レ</td>
<td>d4/212</td>
</tr>
<tr>
<td>ロ</td>
<td>bb/187</td>
<td>ロ</td>
<td>d5/213</td>
</tr>
<tr>
<td>ワ</td>
<td>bc/188</td>
<td>ワ</td>
<td>d6/214</td>
</tr>
<tr>
<td>ジ</td>
<td>bd/189</td>
<td>ジ</td>
<td>d7/215</td>
</tr>
<tr>
<td>キ</td>
<td>be/190</td>
<td>キ</td>
<td>d8/216</td>
</tr>
<tr>
<td>プ</td>
<td>bf/191</td>
<td>プ</td>
<td>d9/217</td>
</tr>
<tr>
<td>シ</td>
<td>c0/192</td>
<td>シ</td>
<td>da/218</td>
</tr>
<tr>
<td>ユ</td>
<td>c1/193</td>
<td>ユ</td>
<td>db/219</td>
</tr>
<tr>
<td>エ</td>
<td>c2/194</td>
<td>エ</td>
<td>dc/220</td>
</tr>
<tr>
<td>ア</td>
<td>c3/195</td>
<td>ア</td>
<td>dd/221</td>
</tr>
<tr>
<td>へ</td>
<td>c4/196</td>
<td>へ</td>
<td>de/222</td>
</tr>
<tr>
<td>エ</td>
<td>c5/197</td>
<td>エ</td>
<td>df/223</td>
</tr>
<tr>
<td>イ</td>
<td>c6/198</td>
<td>イ</td>
<td>e0/224</td>
</tr>
<tr>
<td>イ</td>
<td>c7/199</td>
<td>イ</td>
<td>e1/225</td>
</tr>
<tr>
<td>イ</td>
<td>c8/200</td>
<td>イ</td>
<td>e2/226</td>
</tr>
<tr>
<td>イ</td>
<td>c9/201</td>
<td>イ</td>
<td>e3/227</td>
</tr>
<tr>
<td>イ</td>
<td>ca/202</td>
<td>イ</td>
<td>e4/228</td>
</tr>
<tr>
<td>イ</td>
<td>cb/203</td>
<td>イ</td>
<td>e5/229</td>
</tr>
<tr>
<td>イ</td>
<td>cc/204</td>
<td>イ</td>
<td>e6/230</td>
</tr>
<tr>
<td>イ</td>
<td>cd/205</td>
<td>イ</td>
<td>e7/231</td>
</tr>
</tbody>
</table>
### Table 61-5 (continued)

<table>
<thead>
<tr>
<th>Display</th>
<th>Input (hex/decimal)</th>
<th>Display</th>
<th>Input (hex/decimal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>⊲</td>
<td>e8/232</td>
<td>i</td>
<td>ed/237</td>
</tr>
<tr>
<td>⊱</td>
<td>e9/233</td>
<td>..</td>
<td>ee/238</td>
</tr>
<tr>
<td>⊲̅</td>
<td>ea/234</td>
<td>$</td>
<td>ef/239</td>
</tr>
<tr>
<td>¼</td>
<td>eb/235</td>
<td>*</td>
<td>fo/240</td>
</tr>
<tr>
<td>⊱̅</td>
<td>ec/236</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### displays

#### Synopsis

`extern void displays(string_ptr);`  
`const char * string_ptr;`

#### Description

The `displays` routine writes output to the Display Window screen, under control of the string that is pointed to by `string_ptr`. The `displays` routine returns when the end of the string is encountered. The placement of the output on the screen may be controlled via the `pos_cursor` routine. Attributes may not be used in `displays`.

#### Inputs

The input is a pointer to a string composed of zero or more ordinary characters. Octal or hexadecimal values also may be included in the string, with octal preceded by `\` and hex by `\x`. Pad each value to three integers with leading zeroes.

#### Example

The following entry

```c
pos_cursor(0, 0);
displays("End of test.");
```

produces the following output on the prompt line:

```
End of test.
```

The following coding produces the same output:

```c
pos_cursor(0, 0);
const char * string_ptr;
string_ptr = "End of test.";
displays(string_ptr);
```
display_prompt

Synopsis

extern void display_prompt (string_ptr);
const char * string_ptr;

Description

The display_prompt routine displays a designated string at the beginning of the
prompt line. The cursor is automatically positioned at row zero, column zero.
Once the prompt is written, the cursor is returned to its previous position. The
softkey equivalent of this routine is the PROMPT action. The prompt is visible on
whichever display screen is active at the time the prompt is written. The most
recent prompt is retained in the Display Window. Attributes may not be used in
display_prompt.

Inputs

The input is a pointer to a string composed of zero or more ordinary characters.
Octal or hexadecimal values also may be included in the string, with octal
preceded by \ and hex by \x. Pad each value to three integers with leading
zeroes.

Example

Refer to the example provided for the displays routine. The same string could
be output to the same position without calling the pos_cursor routine:

    display_prompt ("End of test.");

or

    const char * string_ptr;
    string_ptr = "End of test."
    display_prompt (string_ptr);

pos_cursor

Synopsis

extern unsigned int pos_cursor(row, column);
unsigned char row;
unsigned char column;

Description

This routine positions the cursor on the Display Window screen by row and
column numbers.

NOTE: The pos_cursor routine may not be used to position the
cursor on trace screens.
Inputs

The first parameter is the row number. Possible values: 0–16. (The top line of the screen is reserved for header information and cannot be written to.)

The second parameter is the column number. Possible values: 0–63.

Returns

The pos_cursor routine returns the previous cursor position in the form of an unsigned int. The high byte contains the row number; the low byte identifies the column number.

Example

To position the cursor at the far left edge of the prompt line on the Display Window, enter zero for both parameters.

LAYER: 4
STATE: write_to_display
CONDITIONS: KEYBOARD " "
ACTIONS:
{
    pos_cursor(0,0);
    displays("Display on prompt line.");
}

restore_cursor

Synopsis

extern void restore_cursor(position);
unsigned int position;

Description

The restore_cursor routine returns the cursor to a previous position.

NOTE: The restore_cursor routine may not be used to position the cursor on trace screens.

Inputs

The only input is an unsigned int in the same form that is used by the returned value of the pos_cursor routine. The high byte identifies the row number. The low byte identifies the column number.

Example

Suppose the cursor is located in the middle of the Display Window. You want to write a message to the prompt line, but return to your previous location on the screen to continue your display.
```c
{ int previous;

STATE: display
CONDITIONS: KEYBOARD *
ACTIONS:
{
    pos_cursor(8,0);
    displays("This line begins on row 8, column 0 of the Display Window.");
    previous = pos_cursor(0,0);
    displays("This sentence is on the prompt line.");
    restore_cursor(previous);
    displays("This sentence begins on row 8, column 58 of the Display Window, the position of the cursor at the time pos_cursor(0,0) was called.");
}

61.4 Program and User Traces

Unless their sizes are increased, Program Trace and the User Traces retain a maximum of 4096 characters, equivalent to four full screens when every character space is used. (See Section (B)2, below on increasing the size of trace buffers.) When a buffer’s limit is reached, new characters written to the end of the buffer force out the same number of characters from the beginning of the buffer. The prompt line is not part of these buffers. Messages are appended to the end of the buffers. In Freeze mode you may scroll through the buffer using the cursor keys.

You write messages to the User Traces only by using C routines. The Run-mode softkeys for User Traces—USER TR, TRACE 1, TRACE 2, TRACE 3, TRACE 4, TRACE 5, TRACE 6, TRACE 7—appear when the buffers are used in a program.

(A) Variables

There are no extern variables associated exclusively with Traces.

(B) Structures

1. Declaring trace buffers. The trace routines that write to any of the trace buffers require a pointer to the appropriate trace buffer as input. To point to one of the trace buffers, you must first have declared it as a structure. The structure that is common to trace buffers is named trace_buf. This structure is already declared in a file called trace_buf.h located in the HRD/sys/include directory. The trace_buf structure contains another structure, trace_buffer_header, which also is declared in the trace_buf.h file. (These structures are explained in Table 61-6.) Use the #include pre-processor directive to include both declarations in your program.

There are eight trace buffers available (including the Program Trace), each one having its own display screen. All are instances of the trace_buf structure. Declare each one you use as an extern struct, as in this example:

    extern struct trace_buf tl_trbuf;

The names of all the trace buffers are listed in Table 61-6.
```
Table 61-6
Trace Buffer Structures

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structure Name:</td>
<td>trace_buffer_header</td>
<td></td>
<td>Structure of a header for trace buffers. Declared as type extern struct. Declared automatically if a softkey-entered TRACE action is taken. Contained in the structure of the trace buffer. Declaration contained in file named HRD/sysinclude/trace_buf.h. Written to by %m conversion specifier.</td>
</tr>
<tr>
<td></td>
<td>logical_end</td>
<td>0-fff/0-4095</td>
<td>end of data within the buffer. Maximum value is one less than the array_size.</td>
</tr>
<tr>
<td></td>
<td>logical_end_wrap_count</td>
<td>0 non-zero</td>
<td>trace buffer is not full. trace buffer is full. As new lines are written to the end of the trace buffer, lines at the beginning of the buffer are removed.</td>
</tr>
<tr>
<td></td>
<td>modifier</td>
<td></td>
<td>Special-character indicator bit and bit 8 must be zero. For other specific values and their meanings, see Table 61-4.</td>
</tr>
<tr>
<td></td>
<td>color</td>
<td>0-ff/0-255</td>
<td>For specific values and their meanings, see Table 61-4.</td>
</tr>
<tr>
<td></td>
<td>enhance</td>
<td>0-ff/0-255</td>
<td>For specific values and their meanings, see Table 61-4.</td>
</tr>
<tr>
<td></td>
<td>write_lock</td>
<td>0-fffffff/0-65535</td>
<td>prevents two processes from writing to the same buffer at the same time. Should not be altered by user or future access to the trace buffers may be locked out.</td>
</tr>
<tr>
<td></td>
<td>array_size</td>
<td>1000/4096</td>
<td>size of buffer; at present only one value</td>
</tr>
<tr>
<td></td>
<td>line_size</td>
<td>0-3f/0-63</td>
<td>number of characters in last line in buffer</td>
</tr>
<tr>
<td></td>
<td>spare</td>
<td>0</td>
<td>reserved for future use</td>
</tr>
<tr>
<td>Structure Name:</td>
<td>trace_buf</td>
<td></td>
<td>Structure of a trace buffer. Declared as type extern struct. Declared automatically if a softkey-entered TRACE action is taken. Declaration contained in file named HRD/sysinclude/trace_buf.h.</td>
</tr>
<tr>
<td></td>
<td>struct trace_buffer_header</td>
<td>hdr</td>
<td>structure of the trace-buffer header described above</td>
</tr>
<tr>
<td></td>
<td>unsigned long</td>
<td>array [4096]</td>
<td>array of data words in the buffer</td>
</tr>
<tr>
<td>Structure Name:</td>
<td>prog_trbuf</td>
<td>Value (hex/decimal)</td>
<td>Meaning</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------------------------------</td>
<td>---------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>struct trace_buffer_header</td>
<td>hdr</td>
<td></td>
<td>Structure of the Program Trace buffer, an instance of the trace_buf structure declared in file named HRD/sys/include/trace_buf.h. Declared as type extern struct trace_buf. Declared automatically if a softkey-entered TRACE action is taken. Writing to this buffer makes the Run-mode PROG TR softkey appear.</td>
</tr>
<tr>
<td>unsigned long</td>
<td>array [4096]</td>
<td></td>
<td>array of data words in the buffer</td>
</tr>
<tr>
<td>Structure Name:</td>
<td>l1_trbuf</td>
<td></td>
<td></td>
</tr>
<tr>
<td>struct trace_buffer_header</td>
<td>hdr</td>
<td></td>
<td>Structure of one of seven user trace buffers, an instance of the trace_buf structure declared in file named HRD/sys/include/trace_buf.h. Declared as type extern struct trace_buf. Writing to this buffer causes the Run-mode TRACE 1 softkey appear.</td>
</tr>
<tr>
<td>unsigned long</td>
<td>array [4096]</td>
<td></td>
<td>array of data words in the buffer</td>
</tr>
<tr>
<td>Structure Name:</td>
<td>l2_trbuf</td>
<td></td>
<td></td>
</tr>
<tr>
<td>struct trace_buffer_header</td>
<td>hdr</td>
<td></td>
<td>Structure of one of seven user trace buffers, an instance of the trace_buf structure declared in file named HRD/sys/include/trace_buf.h. Declared as type extern struct trace_buf. Writing to this buffer causes the Run-mode TRACE 2 softkey appear.</td>
</tr>
<tr>
<td>unsigned long</td>
<td>array [4096]</td>
<td></td>
<td>array of data words in the buffer</td>
</tr>
<tr>
<td>Structure Name:</td>
<td>l3_trbuf</td>
<td></td>
<td></td>
</tr>
<tr>
<td>struct trace_buffer_header</td>
<td>hdr</td>
<td></td>
<td>Structure of one of seven user trace buffers, an instance of the trace_buf structure declared in file named HRD/sys/include/trace_buf.h. Declared as type extern struct trace_buf. Writing to this buffer causes the Run-mode TRACE 3 softkey appear.</td>
</tr>
<tr>
<td>unsigned long</td>
<td>array [4096]</td>
<td></td>
<td>array of data words in the buffer</td>
</tr>
<tr>
<td>Structure Name:</td>
<td>l4_trbuf</td>
<td></td>
<td></td>
</tr>
<tr>
<td>struct trace_buffer_header</td>
<td>hdr</td>
<td></td>
<td>Structure of one of seven user trace buffers, an instance of the trace_buf structure declared in file named HRD/sys/include/trace_buf.h. Declared as type extern struct trace_buf. Writing to this buffer causes the Run-mode TRACE 4 softkey appear.</td>
</tr>
<tr>
<td>unsigned long</td>
<td>array [4096]</td>
<td></td>
<td>array of data words in the buffer</td>
</tr>
</tbody>
</table>
Table 61-6 (continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
</table>

**Structure Name:** 15_trbuf

- `struct trace_buffer_header hdr` - structure of the trace-buffer header described above
- `unsigned long array [4096]` - array of data words in the buffer

**Structure Name:** 16_trbuf

- `struct trace_buffer_header hdr` - structure of the trace-buffer header described above
- `unsigned long array [4096]` - array of data words in the buffer

**Structure Name:** 17_trbuf

- `struct trace_buffer_header hdr` - structure of the trace-buffer header described above
- `unsigned long array [4096]` - array of data words in the buffer

---

2. **Sizing trace buffers.** There is a preprocessor `#pragma` which allows the user to configure the size of the data array in each user trace buffer. The syntax is `TRACE-NUMBER SIZE TRACE-NUMBER`. Trace number 0 refers to the Program Trace buffer, and trace-number "*" allows all trace-buffer arrays to be set at once. All sizes are given in terms of four-byte array elements.

   The example below first sets all trace-buffer arrays to 16,000 elements, and then down-sizes array number 3 to 2,048 elements.

   ```
   #pragma tracebuf * 16000 3 2048
   ```

   When a trace buffer is declared, its array will have the size specified in the `#pragma tracebuf` directive. If the buffer was not referenced in a `#pragma tracebuf` directive, its array size will default to 4,096. The maximum size for a trace-buffer array is 16,381 elements. If you specify a size that is too small or too large, an error message will be displayed.
(C) Routines

The four trace routines are tracec, tracef, stracef, and traces. These routines are defined below. The softkey TRACE action is built on the tracef routine.

The first argument in three of the trace routines is the address of the trace buffer into which you want output written. Each time you call a trace routine, tracef for example, variables in the named trace-buffer structure are updated. Those variables which store attributes are updated when the %m conversion specifier is used in the tracef routine parameter. When %m is not present, the routine applies the attributes currently stored in the color, modifier, and enhance variables.

The second argument in all trace routines is the character, string, or format pointer to the data that will be written to the selected trace buffer.

The tracef routine allows you to add attributes to messages on the Program Trace screen and User Traces. These attributes are listed in Table 61-4.

Each trace operation appends output to the end of the trace buffer. You may not use the pos_cursor routine to position the cursor on any trace screen. New lines (or blank lines) may be generated via the \n nonliteral. Put the \n nonliteral at the end of the string to generate a leading blank line on the selected trace screen:

tracef(&prog_trbuf, "This trace message will generate a leading blank line.\n");

During real-time display, this line moves just ahead of the freshest trace message and continuously overwrites the oldest one. If you put the \n sequence at the beginning of the format string, no leading blank line will help you distinguish new messages from the old:

tracef(&prog_trbuf, \n"This message will not generate a leading blank line."\n");

tracec

Synopsis

extern void tracec(trace_buffer_ptr, character);
extern struct trace_buf *trace_buffer_ptr;
cnst char character;

Description

The tracec routine outputs a single ASCII character to the trace screen indicated.

Inputs

The first parameter is a pointer to the trace buffer into which the character will be written.

For the second parameter, see the displayc routine.
Example
In this instance, output will be written to the Program Trace screen.

```c
#include <trace_buf.h>
extern struct trace_buf prog_trbuf;
```

LAYER: 2
STATE: display_to_prog_tr
CONDITIONS: KEYBOARD
```
ACTIONS:
{
  tracec(&prog_trbuf, 'a');
  tracec(&prog_trbuf, 'n');
  tracec(&prog_trbuf, 0x65);
  tracec(&prog_trbuf, 'n');
  tracec(&prog_trbuf, 0x65);
  tracec(&prog_trbuf, 065);
}
```

When the user views the PROG TR screen, the output will look like this:

```
a
A
```

e
```
5
```

tracef

Synopsis
 extern int tracef(trace_buffer_ptr, format_ptr, . . . );
 extern struct trace_buf * trace_buffer_ptr;
 const char * format_ptr;

Description
The tracef routine writes output to a specified trace screen, under control of the string, pointed to by format_ptr, that specifies how subsequent arguments are converted for output. If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated but otherwise ignored. The tracef routine returns when the end of the format string is encountered.

Inputs
The first parameter is a pointer to the trace buffer into which the output will be written.

For the second parameter, see the displayf routine. Placement of "\n" in the format string of a call to tracef generates a blank new line on the selected trace screen. (In a displayf routine, the newline character does not blank the new line.)
Attributes are written via the %m conversion specifier to \texttt{trace_buf.hdr.modifier}, \texttt{trace_buf.hdr.color}, and \texttt{trace_buf.hdr.enhance}. The attributes are copied from these variables into subsequent 32-bit data words in the Program Trace and User Traces. Table 61-4 shows the format of this 32-bit word.

Returns

The \texttt{tracef} routine returns the number of characters displayed, or a negative value if the unit is in freeze mode.

Example

This program traces X.29 PAD-control messages in DTE and DCE data packets. The letters "DCE" are underlined in the DCE trace lines.

\begin{verbatim}
LAYER: 3
{
    #include <trace_buf.h>
    extern struct trace_buf\_t3\_trbuf;
    extern unsigned char * m\_packet\_info\_ptr;
    extern unsigned short m\_packet\_lcn;
    unsigned char pad\_ctrl\_msg;
}

STATE: packet\_type
CONDITIONS: DTE DATA Q= 1
ACTIONS:
{
    pad\_ctrl\_msg = m\_packet\_info\_ptr[0];
    tracef ( &t3\_trbuf, "DTE LCN:\%.3x PAD MSG: \%.2x\n", m\_packet\_lcn,
            pad\_ctrl\_msg); 
}
CONDITIONS: DCE DATA Q= 1
ACTIONS:
{
    pad\_ctrl\_msg = m\_packet\_info\_ptr[0];
    tracef ( &t3\_trbuf, "%mDCE%m LCN: \%.3x PAD MSG: \%.2x\n", 0x04000000L,
            0x00000000L, m\_packet\_lcn, pad\_ctrl\_msg); 
}
\end{verbatim}

\texttt{stracef}

Synopsis

\texttt{extern void stracef(array\_ptr, string\_ptr);}
\texttt{unsigned long array\_ptr;}
\texttt{const char * string\_ptr;}

Description

The \texttt{stracef} routine is similar to the \texttt{tracef} routine, except that \texttt{stracef} writes output to a \texttt{variable}, while \texttt{tracef} writes output to a trace screen. The output is under control of the string pointed to by \texttt{string\_ptr} that specifies how subsequent arguments are converted for output. If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated but otherwise ignored. The \texttt{stracef} routine returns when the end of the format string is encountered.
The \texttt{stracef} routine differs from \texttt{sprintf} in that it generates an array of \texttt{longs}, whereas \texttt{sprintf} generates an array of \texttt{chars}. When the \texttt{stracef} array is written to a trace buffer (or to the Display Window) it carries its predefined attributes along with it. An \texttt{sprintf} array, by contrast, will receive the attributes that are active in the buffer at the moment.

At the end of the output string, there will be a null character with the Special Character Indicator bit set in its \texttt{modifier} attribute-byte.

\textbf{Inputs}

The first parameter is a pointer to the variable into which output will be written. The array which will hold output must be declared as a \texttt{long}. By allocating 32 bits for each element, the array may accommodate attributes assigned via the \texttt{%m} conversion specifier. Attributes comprise 24 bits of the \texttt{long}. The preferred form of the declaration is \texttt{unsigned long name[100]}. The size and name of the array are user-determined.

For the second parameter, see the \texttt{displayf} routine.

\textbf{Example}

This program traces X.29 PAD-control messages for DTE and DCE data packets. The resulting trace is identical to the one generated by the example under \texttt{tracef}. Note that attributes that are turned on in an \texttt{stracef} array do not need to be turned off after the array has been brought, via the \texttt{%b} conversion specifier, into a \texttt{tracef} format string.

\texttt{LAYER: 3}

\{ 
    \#include <trace_buj.h>
    extern struct trace_buf13_trbuf;
    extern unsigned char * m_packet_info_ptr;
    extern unsigned short m_packet_lcn;
    unsigned char pad_ctrl_msg;
    unsigned long source[4];
\}

\texttt{STATE: packet_type}

\texttt{CONDITIONS: DTE DATA Q= 1}

\texttt{ACTIONS:}

\{ 
    \texttt{stracef (source, "%s", "DTE");}
\}

\texttt{NEXT_STATE: pad_msg_trace}

\texttt{CONDITIONS: DCE DATA Q= 1}

\texttt{ACTIONS:}

\{ 
    \texttt{stracef (source, "%m%s", 0x04000000L, "DCE");}
\}

\texttt{NEXT_STATE: pad_msg_trace}

\texttt{STATE: pad_msg_trace}

\texttt{CONDITIONS: ENTER STATE}

\texttt{ACTIONS:}

\{ 
    \texttt{pad_ctrl_msg = m_packet_info_ptr[0];}
    \texttt{tracef (&13_trbuf,"%b LCN:%.3x PAD MSG:%.2x\n", source, m_packet_lcn,
    pad_ctrl_msg);}
\}

\texttt{NEXT_STATE: packet_type}

61-30
traces

Synopsis
extern void traces(trace_buffer_ptr, string_ptr);
extern struct trace_buf trace_buffer_ptr;
const char * string_ptr;

Description
The traces routine writes output to a specified trace screen, under control of the string that is referenced by string_ptr. The traces routine returns when the end of the string is encountered.

Inputs
The first parameter is a pointer to the trace buffer into which the output will be written.
For the second parameter, see the displays routine.

Example
In this instance, output will be written to the TRACE 1 screen.
The following entry produces the following output on the TRACE 1 trace screen:

```
End of test.
```
The following coding produces the same output:

```
```
61.5 Attributes

Attributes are written to the Display Window and to the trace buffers in 32-bit words that include 8 bits of character data (the second-lowest byte) and 24 bits of attributes. The format of the 32-bit data word, given in Table 61-4, is the same for the Display Window and for the trace buffers.

In `displayf` routines, the `%m` conversion specifier writes input to `window_color` and `window_modifier` variables. These variables are then copied into data words written to the Display Window by string pointers in this and subsequent `displayf` routines. See Figure 61-1.

In `tracef` routines, the `%m` conversion specifier writes input to the `trace_buffer_header` structure for a particular user-trace buffer. The header is then copied into each data word written to the particular user buffer by string pointers in this and subsequent `tracef` routines. See Figure 61-2.

(A) Applying Attributes As Data Is Buffered

There are two ways an attribute may be assigned to a character in the Display Window. One way uses the `%m` conversion specifier to assign attributes to the `window_color` and `window_modifier` variables. This program, for example, includes a `displayf` routine that uses the `%m` conversion specifier to write underlined data to the Display Window:

```
STATE: apply_attribute_to_window_color_variable
CONDITIONS: ENTER_STATE
ACTIONS:
{
    pos_cursor (1,0);
    displayf("%mThis data is underlined in the Display Window.", 0x04000000L);
}
```

The chart in Table 61-4 shows the hex value `04000000L` in the "input" column alongside the underline attribute. This means that when the value `0x04000000L` is input to the conversion specifier `%m`, an underline attribute is applied to the current `displayf` string and any that follow until the attribute is turned off. The underline attribute actually is applied to the external `window_color` variable. See Table 61-2. The `window_color` and `window_modifier` variables lend their attributes to every character that is written in a format string to the Display Window. In Run mode if the user presses the softkey for DSP WND, he will see his underlined string. Subsequent characters or strings written to the Display Window also will be underlined.
Figure 61-1 When a `display` routine is called, the attributes assigned via the `%m` conversion specifier are stored in two `extern` variables, accessible to the user. Both color and enhance attributes are contained in `window_color`. The low byte in `window_color` indicates the color; the high byte contains enhancements. In this example, the following attributes will be assigned to characters written to the Display Window: reverse-image enhancement, green-on-black color, and ASCII font. Before a character is written to the Display Window, it is combined in a `long` with its attributes, as mapped in the figure.
tracef(&l1_trbuf, "%mDATA", Ox08100000L);

Figure 61-2 When a `tracef` routine is called, the attributes assigned via the `%m` conversion specifier are stored in three variables in the trace-buffer header of a designated buffer. In this example, `l1_trbuf.hdr` holds the following attributes: reverse-image enhancement, green-on-black color, and ASCII font. Before a character is written to the buffer, it is combined in a `long` with its attributes, as mapped in the figure.
The same attribute could be applied to a string in any of the user-trace buffers, as follows:

```
#include <trace_buf.h>
extern struct trace_buf ll_trbuf;
```

```
STATE: apply_attribute_to_header
CONDITIONS: ENTER_STATE
ACTIONS:
{
    tracef (&ll_trbuf, "This data is underlined.", 0x04000000L);
}
```

Only the header for the TRACE 1 display is affected by this conversion. Only the TRACE 1 buffer is written to. When other trace buffers are subsequently written to, the strings will not receive underlining as a result of the attributes applied above to the TRACE 1 header.

(B) Applying Attributes to Buffered Data

The Display Window is an array of 1,088 long integers, each including one byte of character data and three bytes of attributes. The character data is generated by strings in display routines. The attributes for each character are derived from the current window_color and window_modifier values at the time the character is written to the display-window buffer.

Once the data word is written to the buffer as an element in the array, it can be accessed and written to—and therefore changed—the same as any other location in memory. In the example that follows, a string is written to the Display Window without underlining. Then, as a result of a keyboard input from the operator, the first 32-bit word in the string (containing the first character, the letter “T”) is given a new value that includes the underline attribute.

```
{
    extern unsigned long display_window_buffer[1088];
    extern struct
    {
        unsigned char mpm;
        unsigned char cpm;
    } display_window_index_buffer[17];
}
```

```
STATE: apply_attribute_directly_to_display_window
CONDITIONS: ENTER_STATE
ACTIONS:
{
    pos_cursor(1,0);
    displayf ("This data is not underlined.");
}
```
CONDITIONS: KEYBOARD • *

ACTIONS:
{
    display_window_buffer[64] = ((display_window_buffer[64] & -0x04000000L) | 0x04000000L);
    display_window_index_buffer[1].mpm ++;
}

Incrementing `display_window_index_buffer.mpm` is necessary to alert the processor on the CPM card (containing the display-controller software) that the program has changed the contents of the Display Window. Refer to Table 61-3 for an explanation of this structure.

The bitwise `anding` and `oring` in the example are necessary if you want to change certain bits in the word without affecting others. Note that the value whose complement (-) is `anded` with `display_window_buffer` element #64 is the “mask” for the underline attribute in Table 61-4; and the value to the right of the `or` operator (!) is the “input” value for the underline attribute.
### Table 61-7
Conversion Specifiers

<table>
<thead>
<tr>
<th>Specifier</th>
<th>Argument type</th>
<th>Conversion Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>%b</td>
<td>integer-array pointer</td>
<td>array of long integers. 2nd byte of each long is displayed as character. 1st, 3rd, and 4th bytes interpreted as attributes. Array begins at pointer, ends at element containing null character and Special Character bit = 1.</td>
</tr>
<tr>
<td>%l</td>
<td>integer</td>
<td>signed decimal representing 15-bit value</td>
</tr>
<tr>
<td>%c</td>
<td>unsigned character</td>
<td>unsigned character</td>
</tr>
<tr>
<td>%#c</td>
<td>unsigned character</td>
<td>newline character displayed as 'r' rather than acted on</td>
</tr>
<tr>
<td>%d</td>
<td>integer</td>
<td>signed decimal representing 15-bit value</td>
</tr>
<tr>
<td>%ld</td>
<td>integer</td>
<td>signed decimal representing 31-bit value</td>
</tr>
<tr>
<td>%H</td>
<td>character-array pointer</td>
<td>character array indicated by argument appears as small hex characters. (Precision as to number of characters becomes length of the array, overriding usual null-termination of strings.)</td>
</tr>
<tr>
<td>%m</td>
<td>integer</td>
<td>long integer not displayed or printed, but written to attribute header-variable for Display Window or for one of the trace buffers</td>
</tr>
<tr>
<td>%o</td>
<td>integer</td>
<td>unsigned octal representing 16-bit value</td>
</tr>
<tr>
<td>%lo</td>
<td>integer</td>
<td>unsigned octal representing 32-bit value</td>
</tr>
<tr>
<td>%#o</td>
<td>integer</td>
<td>unsigned octal representing 16-bit value, preceded by 0</td>
</tr>
<tr>
<td>%#lo</td>
<td>integer</td>
<td>unsigned octal representing 32-bit value, preceded by 0</td>
</tr>
<tr>
<td>%p</td>
<td>integer</td>
<td>unsigned hexadecimal (lower-case letters) representing 32-bit value, with a minimum 5 digits displayed and a colon between the 4 right-hand digits and the 1-4 left-hand digits. Useful for displaying CPU segment number and offset.</td>
</tr>
<tr>
<td>%s</td>
<td>character-array pointer</td>
<td>array of characters beginning at pointer and ending at null terminator or at array-length precision, whichever occurs first</td>
</tr>
<tr>
<td>%#s</td>
<td>character-array pointer</td>
<td>newline character displayed as 'r' rather than acted on</td>
</tr>
<tr>
<td>%u</td>
<td>integer</td>
<td>unsigned decimal representing 16-bit value</td>
</tr>
<tr>
<td>%lu</td>
<td>integer</td>
<td>unsigned decimal representing 32-bit value</td>
</tr>
<tr>
<td>%#u</td>
<td>integer</td>
<td>hex characters (example: 9 F E S ) representing 16-bit value</td>
</tr>
<tr>
<td>%#lu</td>
<td>integer</td>
<td>hex characters (example: 9 F E S 30 1 :J ) representing 32-bit value</td>
</tr>
</tbody>
</table>
Table 61-7 (continued)

<table>
<thead>
<tr>
<th>Specifier</th>
<th>Argument type</th>
<th>Conversion Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>%x</td>
<td>integer</td>
<td>unsigned hexadecimal (lower-case letters) representing 16-bit value</td>
</tr>
<tr>
<td>%ix</td>
<td>integer</td>
<td>unsigned hexadecimal (lower-case letters) representing 32-bit value</td>
</tr>
<tr>
<td>%#x</td>
<td>integer</td>
<td>unsigned hexadecimal (lower-case letters) representing 16-bit value, preceded by 0x</td>
</tr>
<tr>
<td>%#ix</td>
<td>integer</td>
<td>unsigned hexadecimal (lower-case letters) representing 32-bit value, preceded by 0x</td>
</tr>
<tr>
<td>%x</td>
<td>integer</td>
<td>unsigned hexadecimal (upper-case letters) representing 16-bit value</td>
</tr>
<tr>
<td>%ix</td>
<td>integer</td>
<td>unsigned hexadecimal (upper-case letters) representing 32-bit value</td>
</tr>
<tr>
<td>%#x</td>
<td>integer</td>
<td>unsigned hexadecimal (upper-case letters) representing 16-bit value, preceded by 0x</td>
</tr>
<tr>
<td>%#ix</td>
<td>integer</td>
<td>unsigned hexadecimal (upper-case letters) representing 32-bit value, preceded by 0x</td>
</tr>
<tr>
<td>%\n</td>
<td>none</td>
<td>displays an \n</td>
</tr>
<tr>
<td>%%</td>
<td>none</td>
<td>displays a %</td>
</tr>
</tbody>
</table>

61.6 Protocol Trace Buffers

There are two Protocol Trace buffers, one dedicated to Layer 2 and the other to Layer 3 data. Run-mode softkeys for accessing these traces—PROTOCL, L2TRACE, and L3TRACE—appear when personality packages are loaded in at Layers 2 and 3. The prompt line is not part of these buffers.

The size of each Protocol Trace buffer is 65,536 bytes. Of this total, two bytes are dedicated to the buffer header and two bytes to the trailer. The usable size of a Protocol Trace buffer, therefore, is 65,532 bytes. When a buffer’s limit is reached, new characters written to the end of the buffer force out the same number of characters from the beginning of the buffer. In Freeze mode you may scroll through the buffer using the cursor keys.

You cannot write directly to the Protocol Trace buffers. Monitor the position within the buffers, as well as the wrap count, using the variables and structures discussed below.

(A) Variables

The addresses of the variables in Table 61-8 identify the physical location of the beginning and end of each Protocol Trace buffer. The beginning position is at the first data byte in the buffer. The end is just after the last data byte.
<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern unsigned char</td>
<td>l2pp_trbuff</td>
<td></td>
<td>First data byte in the Layer 2 Protocol Trace buffer. Address of this variable—segment number plus offset—will indicate the physical location of the first data byte, two bytes from the beginning of the buffer. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern unsigned long</td>
<td>l2pp_trbuff_end</td>
<td></td>
<td>First byte in the two-byte trailer of the Layer 2 Protocol Trace buffer—i.e., after the last data byte. Address of this variable—segment number plus offset—will indicate the physical location of the end of the data area, hexadecimal FFFE bytes from the beginning of the buffer. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern unsigned char</td>
<td>l3pp_trbuff</td>
<td></td>
<td>First data byte in the Layer 3 Protocol Trace buffer. Address of this variable—segment number plus offset—will indicate the physical location of the first data byte, two bytes from the beginning of the buffer. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern unsigned long</td>
<td>l3pp_trbuff_end</td>
<td></td>
<td>First byte in the two-byte trailer of the Layer 3 Protocol Trace buffer—i.e., after the last data byte. Address of this variable—segment number plus offset—will indicate the physical location of the end of the data area, hexadecimal FFFE bytes from the beginning of the buffer. Line Setup configured for emulate or monitor mode.</td>
</tr>
</tbody>
</table>
(B) Structures

The structure variables in Table 61-9 contain the high and low bytes of a beginning and ending offset and wrap-count in the Layer 2 and Layer 3 Protocol Trace buffers. Create a logical beginning (or ending) offset within a buffer by combining the two offset-variables relating to a beginning (or ending) position into a single, two-byte offset. Add the resulting offset to the address of \textit{l3_trbuff} to identify the physical address of a logical location.

The example below uses \texttt{#define} preprocessor directives for determining beginning and ending offsets in the Layer 3 Protocol Trace buffer. When \texttt{get\_l3pp\_value\_end} is encountered in a program, for example, each of the two “end” offset-variables is cast into a \texttt{long} and, if necessary, shifted left to its appropriate position in an offset. Then the two variables are added together.

\begin{verbatim}
#define get_l3pp_value_begin 
((unsigned long) (l3pp_trbuff_ctl.begin_of_lhi) << 8) + 
((unsigned long) (l3pp_trbuff_ctl.begin_of_llo))

#define get_l3pp_value_end 
((unsigned long) (l3pp_trbuff_ctl.end_of_lhi) << 8) + 
((unsigned long) (l3pp_trbuff_ctl.end_of_llo))
\end{verbatim}

When the ending offset, in this example, is added to the address of \textit{l3_trbuff}, the result is the address of the logical end in the buffer:

\begin{verbatim}
unsigned long end_address;
end_address = &l3_trbuff + get_l3pp_value_end;
\end{verbatim}

You may also use the offsets and wrap counts to determine how much data is currently in the buffer. Include the wrap count in the high two bytes of a four-byte offset. Then subtract the beginning offset from the ending offset.

\begin{verbatim}
#define get_l3pp_value_begin 
((unsigned long) (l3pp_trbuff_ctl.begin_wrap_hi) << 24) + 
((unsigned long) (l3pp_trbuff_ctl.begin_wrap_lo) << 16) + 
((unsigned long) (l3pp_trbuff_ctl.begin_of_lhi) << 8) + 
((unsigned long) (l3pp_trbuff_ctl.begin_of_llo))

#define get_l3pp_value_end 
((unsigned long) (l3pp_trbuff_ctl.end_wrap_hi) << 24) + 
((unsigned long) (l3pp_trbuff_ctl.end_wrap_lo) << 16) + 
((unsigned long) (l3pp_trbuff_ctl.end_of_lhi) << 8) + 
((unsigned long) (l3pp_trbuff_ctl.end_of_llo))
\end{verbatim}

\begin{verbatim}
unsigned long end, begin, count;
end = get_l3pp_value_end;
begin = get_l3pp_value_begin;
count = end - begin;
\end{verbatim}
### Table 61-9
Protocol Trace Buffer Structures

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Structure Name:</strong></td>
<td>Ipp_trbuff_ctl</td>
<td></td>
<td>Declared as type struct. The variables contained in this structure monitor logical location in a Protocol Trace buffer. Reference structure variables as follows: Ipp_trbuff_ctl.begin_off_hi.</td>
</tr>
<tr>
<td>unsigned char</td>
<td>begin_off_hi</td>
<td>0-ffl0-255</td>
<td>High byte of a 2-byte offset from the physical beginning of the Protocol Trace buffer to a logical beginning in the buffer. Range of the two-byte offset is 2 through hexadecimal FFFE.</td>
</tr>
<tr>
<td>unsigned char</td>
<td>begin_off_lo</td>
<td>0-ffl0-255</td>
<td>Low byte of a 2-byte offset from the physical beginning of the Protocol Trace buffer to a logical beginning in the buffer. Range of the two-byte offset is 2 through hexadecimal FFFE.</td>
</tr>
<tr>
<td>unsigned char</td>
<td>begin_wrap_hi</td>
<td>0-ffl0-255</td>
<td>High byte of a 2-byte count of the number of times a logical beginning has wrapped through the Protocol Trace buffer.</td>
</tr>
<tr>
<td>unsigned char</td>
<td>begin_wrap_lo</td>
<td>0-ffl0-255</td>
<td>Low byte of a 2-byte count of the number of times a logical beginning has wrapped through the Protocol Trace buffer. It will have a value of zero only once. Once the count reaches hexadecimal FFFF, it will wrap to one.</td>
</tr>
<tr>
<td>unsigned char</td>
<td>end_off_hi</td>
<td>0-ffl0-255</td>
<td>High byte of a 2-byte offset from the physical beginning of the Protocol Trace buffer to a logical end in the buffer. Range of the two-byte offset is 2 through hexadecimal FFFE.</td>
</tr>
<tr>
<td>unsigned char</td>
<td>end_off_lo</td>
<td>0-ffl0-255</td>
<td>Low byte of a 2-byte offset from the physical beginning of the Protocol Trace buffer to a logical end in the buffer. Range of the two-byte offset is 2 through hexadecimal FFFE.</td>
</tr>
<tr>
<td>unsigned char</td>
<td>end_wrap_hi</td>
<td>0-ffl0-255</td>
<td>High byte of a 2-byte count of the number of times a logical end has wrapped through the Protocol Trace buffer.</td>
</tr>
<tr>
<td>unsigned char</td>
<td>end_wrap_lo</td>
<td>0-ffl0-255</td>
<td>Low byte of a 2-byte count of the number of times a logical end has wrapped through the Protocol Trace buffer. It will have a value of zero only once. Once the count reaches hexadecimal FFFF, it will wrap to one.</td>
</tr>
<tr>
<td><strong>Structure Name:</strong></td>
<td>12pp_trbuff_ctl</td>
<td></td>
<td>An instance of the Ipp_trbuff_ctl structure, declared as type extern struct Ipp_trbuff_ctl. The variables contained in this structure monitor logical location in the Layer 2 Protocol Trace buffer. Has the same structure as Ipp_trbuff_ctl. Reference structure variables as follows: 12pp_trbuff_ctl.begin_off_hi.</td>
</tr>
</tbody>
</table>
Table 61-9 (continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Structure Name:</td>
<td>i3pp_trbuff_ctl</td>
<td>An instance of the lpp_trbuff_ctl structure, declared as type extern struct lpp_trbuff_ctl. The variables contained in this structure monitor logical location in the Layer 3 Protocol Trace buffer. Has the same structure as lpp_trbuff_ctl. Reference structure variables as follows: i3pp_trbuff_ctl.begin_off_h.</td>
</tr>
</tbody>
</table>

(C) Routines

There are no routines associated exclusively with Protocol Traces.
62 Counters, Timers, and Accumulators

62.1 Counters

The translator declares the following structure for counters that are entered as softkey tokens on the Protocol Spreadsheet:

```c
struct counter_struct
{
    unsigned long current;
    unsigned long last;
    unsigned long maximum;
    unsigned long minimum;
    unsigned short sample_count;
    unsigned long total_high;
    unsigned short total_low_low;
    unsigned short total_low_high;
    unsigned short out_of_range;
    unsigned short changed;
    unsigned long prev;
    unsigned long old;
};
struct counter_struct counter_name={0,0,0,-0ull};
```

The first eight counter variables in the structure are used to calculate statistical values whenever the counter is sampled. See Table 62-1. Three of the variables—`counter_name.current`, `counter_name.prev`, and `counter_name.old`—come into play each time the counter is incremented, decremented, or set to a particular value.

Counters are internal program variables and counter interrupts are strictly program-generated signals, so the C programmer is free to ignore this structure and maintain counts and statistics in a different way. Please note, however, that the 68010 CPU expects this counter structure when it polls the 80286 periodically for statistical values to display in columns on the tabular and graphic stats screens.

(A) Current, Previous, and Old Values

When a counter is incremented, decremented, or set to a specific value on the Protocol Spreadsheet, the program does not signal a `counter_name_change` interrupt automatically. First it verifies that the new value of the counter really is a change from the previous value. See Table 62-2. For this comparison, the program needs to maintain two variables, `counter_name.current` and `counter_name.prev`.

62-1
<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structure Name: counter_struct</td>
<td></td>
<td>Structure of a counter. Declared as type struct. Declared automatically if a program counter is used. Program counters assigned to structure as follows: struct counter_struct counter_name. Reference a structure variable as follows: counter_name.current.</td>
</tr>
<tr>
<td>unsigned long</td>
<td>current</td>
<td>This value of the counter is acted on directly by program actions.</td>
</tr>
<tr>
<td>unsigned long</td>
<td>last</td>
<td>Last sampled value; displayed on the tabular statistics screen.</td>
</tr>
<tr>
<td>unsigned long</td>
<td>maximum</td>
<td>Maximum value of all samples; displayed on the tabular statistics screen.</td>
</tr>
<tr>
<td>unsigned long</td>
<td>minimum</td>
<td>Minimum value of all samples; displayed on the tabular statistics screen.</td>
</tr>
<tr>
<td>unsigned short</td>
<td>sample_count</td>
<td>Number of samples.</td>
</tr>
<tr>
<td>unsigned long</td>
<td>total_high</td>
<td>High four bytes of an eight-byte counter total.</td>
</tr>
<tr>
<td>unsigned short</td>
<td>total_low_low</td>
<td>Low two bytes of an eight-byte counter total. This two-byte variable counts to 65,535.</td>
</tr>
<tr>
<td>unsigned short</td>
<td>total_low_high</td>
<td>Bytes 3 and 4 of an eight-byte counter total.</td>
</tr>
<tr>
<td>unsigned short</td>
<td>out_of_range</td>
<td>Number is out of range, either incremented beyond the range or decremented below 0; should not be factored into averages. For future use.</td>
</tr>
<tr>
<td>unsigned short</td>
<td>changed</td>
<td>When converting a counter action to C, the translator compares prev with current to determine whether counter has changed. Then prev is updated to current and counter_name_change is signaled.</td>
</tr>
<tr>
<td>unsigned long</td>
<td>prev</td>
<td>When converting a counter condition to C, the translator compares old with current to determine whether true condition is new (transitional). After program logic has examined counter, old is updated to prev.</td>
</tr>
<tr>
<td>unsigned long</td>
<td>old</td>
<td></td>
</tr>
</tbody>
</table>

Here, for example, is the C translation of the simple action COUNTER example SET 5.

```c
counter_example.current = 5;
if (counter_example.prev != counter_example.current)
{
    counter_example.old = counter_example.prev;
    counter_example.prev = counter_example.current;
    signal (counter_example_change);
}
```
It is clear from the translation that the variable counter_example.prev is used to limit the number of counter_example_change interrupts to those cases where the current value of the counter really has changed.

What is counter_name.old used for? We will preface the answer by citing the behavior of the counter in the following spreadsheet example.

STATE: threshold_condition
    CONDITIONS: KEYBOARD " "
    ACTIONS: COUNTER spacebar INC
    CONDITIONS: COUNTER spacebar GE 7
    ACTIONS: ALARM

Each time you press the space bar while this program is running, the counter will increment, but no matter how many times you press the space bar the alarm will only sound once. It will sound on the seventh keystroke, the first time the counter is greater than or equal to 7. If the program had a decrement or set action that lowered the counter to less than 7, the alarm would sound again when the counter reached the 7 threshold.

The translator accomplishes this threshold condition by coding the waitfor clause as follows:

```
counter_spacebar_change && (! (counter_spacebar.old >= 7))) && (counter_spacebar.current >= 7);
```

Since counter_spacebar.prev was used (and then updated to “current”) in the if statement that sent the counter_spacebar_change interrupt, the “old” value is required in the waitfor condition to insure a “transitional” or “threshold” counter condition.
(B) Sampling a Counter

Here is the translator's version of a counter sample action:

```c
counter_name.last = counter_name.current;
if (counter_name.current > counter_name.maximum)
    { counter_name.maximum = counter_name.current; }
if (counter_name.current < counter_name.minimum)
    { counter_name.minimum = counter_name.current; }
counter_name.sample_count++;
{
    unsigned long temp;
    temp = (counter_name.current & 0x0000ffff) + counter_name.total_low_low;
    counter_name.total_low_low = temp;
    temp = (counter_name.current >> 16) + counter_name.total_low_high + (temp >> 16);
    counter_name.total_low_high = temp;
    counter_name.total_high += temp >> 16;
}
counter_name.current = 0;
```

In order to establish an average value for all samples, a grand total for current values at the time of each sampling must be maintained. Since an ordinary INTERVIEW current counter is 32 bits, the counter that maintains the grand total of current counts must be larger (64 bits). There is no data type this large in C, and so the "total" counter is distributed among three variables and the somewhat complicated coding involving the `temp` variable is required to add the current counter to this composite counter.

(C) Updating the Statistics Screen

The CPM polls the MPM continuously to see if data is available to be output to the printer or the plasma display. This data includes character data, trace data, prompts, and values to be posted to the statistics screens.

In order to know where on the statistics screens the values for the particular counters (and timers and accumulators) should be placed, the 68010 CPU on the CPM needs some help from the program (that is, from the MPM). This help is in the form of a "stat message" that the translator (or the programmer) codes once at the beginning of the program. The stat message is a structure that the MPM sends to the CPM. See Table 62-3. The stat message says, in effect, "Here is the address of a counter structure. When you access this structure during the running of the program in order to pull out the current, last, maximum, minimum, total, and sample-count values, display those values on the row of the tabular stats screen where the user has typed 'spacebar'" (for example).
### Table 62-3

#### Counter, Timer, and Accumulator Structures

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Structure Name:</strong> stat_msg</td>
<td></td>
<td></td>
<td>Structure of a stat message. A stat message is sent once for each named counter, timer, or accumulator. Declared as type struct. Declared automatically if a softkey-entered COUNTER is used as a condition, or if softkey-entered COUNTER, TIMER, or ACCUMUL action is taken. Program stat messages assigned to structure as follows: struct stat_msg name. You must assign values to the elements of the structure. Reference a structure variable as follows: name.type.</td>
</tr>
<tr>
<td>unsigned short</td>
<td>op_type</td>
<td>0a00/2560</td>
<td>Register statistics objects from the MPM to the CPM. Other values and meanings for future use.</td>
</tr>
<tr>
<td>unsigned short</td>
<td>type</td>
<td>0100/256</td>
<td>0200/512</td>
</tr>
<tr>
<td>unsigned long</td>
<td>object_name</td>
<td></td>
<td>The MPM (80286) address of a counter, timer, or accumulator name, converted to CPM (68010) format. To get an object_name address, enter: name.object_name = get_68k_phys_addr(&quot;name_of_counter&quot;);</td>
</tr>
<tr>
<td>unsigned long</td>
<td>object_address</td>
<td></td>
<td>The MPM (80286) address of a counter, timer, or accumulator structure, converted to CPM (68010) format. To get a structure address for a counter, enter: name.object_address = get_68k_phys_addr(&amp;counter_name_of_counter);</td>
</tr>
</tbody>
</table>

Here is a C program that causes the current value of a counter named “key” to increment on the tabular-statistics screen each time an ASCII-keyboard key is struck.

```c
{
    struct
    {
        unsigned short op_type;
        unsigned short type;
        unsigned long object_name;
        unsigned long object_address;
    } stat_msg;
    extern unsigned long get_68k_phys_addr();
}
```
struct counter_struc
{
    unsigned long current;
    unsigned long last;
    unsigned long maximum;
    unsigned long minimum;
    unsigned short sample_count;
    unsigned long total_high;
    unsigned short total_low_high;
    unsigned short total_low;
    unsigned short out_of_range;
    unsigned short changed;
    unsigned long prev;
    unsigned long old;
};
struct counter_structure counter_key;
extern fast_event keyboard_new_key;

STATE: update_stat_screen
{
    stat_msg.op_type = 2560;
    stat_msg.type = 256;
    stat_msg.object_name = get_68kPhys_addr("key");
    stat_msg.object_address = get_68kPhys_addr(&counter_key);
    send_stat_message(&stat_msg);
    waitfor
    {
        keyboard_new_key:
        {
            counter_key.current++;
        }
    }
}

The variable stat_msg.object_name is a pointer to the name of the counter that the user has entered on the protocol spreadsheet. The program gives this name to the CPM, and expects the CPM to locate the name among the names that the user has entered on the tabular or graphic statistics menu. The delivery to the CPM of a pointer to the stats-menu name and a pointer to the counter structure is the purpose of the stat message. The message allows the CPM to correlate a line on the statistics results screen with an actual program counter (or timer or accumulator).
NOTE TO C PROGRAMMERS: When the translator creates a counter variable it adds the prefix `counter_` to the spreadsheet name, but the programmer who is working primarily in C and is not making use of spreadsheet counters can name the counter any way he wishes, with or without the prefix. Similarly, the string that is communicated to the CPM in `stat_msg.object_name` ("key" in the example above) must agree with the name on the stats menu, but it need not bear any resemblance to the name of the counter structure.

NOTE ALSO: In most of the examples in this manual, we have not bothered to declare routines since it is not necessary. In the absence of a declaration, the compiler assumes that the routine is external and that it returns an integer. In nearly all cases, this assumption works. `get_68k_phys_addr()` returns a `long`, however, and must be declared.
62.2 Timers

The translator declares the following structure for timers that are entered as softkey tokens on the Protocol Spreadsheet:

```c
struct timer_struct
{
    unsigned long current;
    unsigned long last;
    unsigned long maximum;
    unsigned long minimum;
    unsigned short sample_count;
    unsigned long total_high;
    unsigned short total_low_low;
    unsigned short total_low_high;
    unsigned long start_tick_value;
    unsigned short running;
    unsigned short changed;
};
```

There are no timer conditions in the software (since timeouts provide the time-triggging function), and therefore all of the variables in the structure serve as data for the CPM when it updates the stats screens. See Table 62-4. A stat message must be sent so the CPM can correlate a line on the statistics results screen with the correct program timer. The stat message is documented in the previous section on counters. The timer stat message is different only in respect that the `stat_msg.type` element should be set to 512 instead of 256.

Timer restart, continue, and stop actions are explained in this section. The clear action is simply a matter of changing the elements in the structure to zero (except for `timer_name.minimum`, which becomes the one's complement of zero).

(A) Time Ticks

Time ticks are timed increments of either of two hardware counters in the INTERVIEW. The programmer can select which of the two timing mechanisms to use for a given timer.

One tick-counter is on the FEB card and is used to time-stamp incoming data and EIA leads. The intervals between ticks is determined on the FEB Setup menu. Ticks can be enabled/disabled on the same menu. The current value of this counter is available in a variable called `ll_tick_count`. See Table 62-5. The current value always reflects the number of ticks since the program entered Run mode. The number of ticks may or may not equate to the amount of time in Run mode, since ticks are also encoded in playback data and the playback rate is subject to “local conditions” such as playback speed and idle suppression.

FEB time ticks are the most precise timing mechanism in that they have a resolution to 10 microseconds. They also represent the most durable method of timekeeping, since they preserve the original data timings even during playback.
### Table 62-4
**Timer Structures**

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Structure Name:</strong> timer_struct</td>
<td></td>
<td></td>
<td>Structure of a timer. Declared as type <code>struct</code>. Declared automatically if a program timer is used. Program timers assigned to structure as follows: <code>struct timer_struct timer_name</code>. Reference a structure variable as follows: <code>timer_name.current</code>.</td>
</tr>
<tr>
<td>unsigned long</td>
<td>current</td>
<td></td>
<td>Current value of timer, not updated while timer is running. Values are in microseconds rounded to tick-unit on FEB Setup screen.</td>
</tr>
<tr>
<td>unsigned long</td>
<td>last</td>
<td></td>
<td>Value of last sample; displayed on the tabular statistics screen.</td>
</tr>
<tr>
<td>unsigned long</td>
<td>maximum</td>
<td></td>
<td>Maximum value of all samples; displayed on the tabular statistics screen.</td>
</tr>
<tr>
<td>unsigned long</td>
<td>minimum</td>
<td></td>
<td>Minimum value of all samples; displayed on the tabular statistics screen. Should be initialized as <code>0ul</code>.</td>
</tr>
<tr>
<td>unsigned short</td>
<td>sample_count</td>
<td></td>
<td>Number of samples.</td>
</tr>
<tr>
<td>unsigned long</td>
<td>total_high</td>
<td></td>
<td>High four bytes of an eight-byte timer total.</td>
</tr>
<tr>
<td>unsigned short</td>
<td>total_low_low</td>
<td></td>
<td>Low two bytes of an eight-byte timer total.</td>
</tr>
<tr>
<td>unsigned short</td>
<td>total_low_high</td>
<td></td>
<td>Bytes 3 and 4 of an eight-byte timer total.</td>
</tr>
<tr>
<td>unsigned long</td>
<td>start_tick_value</td>
<td></td>
<td>Tick-count in microseconds when timer was started, restarted, or continued. For line-related conditions at Layer 1, this value is stored in <code>11tick_count</code>; for non-line conditions, use <code>get_wall_time_286_ticks</code> routine.</td>
</tr>
<tr>
<td>unsigned short</td>
<td>running</td>
<td>0</td>
<td>Stopped. This variable is polled and a zero stops the timer from incrementing and sets the current value to <code>timer_name.current</code> (understood as microseconds).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-0</td>
<td>Running. All 1's in this variable causes the timer to increment, showing a value that equals (wall-time ticks - <code>timer_name.start_tick_value</code>) + <code>timer_name.current</code>.</td>
</tr>
<tr>
<td>unsigned short</td>
<td>changed</td>
<td>-0</td>
<td>For future use.</td>
</tr>
</tbody>
</table>
Table 62-5
Timer Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern unsigned long</td>
<td>11_tick_count</td>
<td>This variable counts ticks from the start of Run mode.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tick=sec, msec, etc., depending on FEB setup.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Subtract early value from later value to create a timer.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACTIONS:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{ displayf(&quot;%ld msecs &quot;, (11_tick_count - \</td>
</tr>
<tr>
<td></td>
<td></td>
<td>timer_name.start_tick_value));}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Add to start_of_run_time to determine more precise current time</td>
</tr>
<tr>
<td></td>
<td></td>
<td>for time-stamping events.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td></td>
<td>start_of_run_date</td>
<td>Date when Run mode entered.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Byte 1 (low byte) indicates day; byte 2 stores month; and byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 and 4 indicates year. May be used to time-stamp events.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>See also start_of_run_time.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td></td>
<td>start_of_run_time</td>
<td>Time when Run mode entered.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Byte 1 (low byte) indicates seconds; byte 2 stores minutes; and byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 indicates hours. May be used to time-stamp events.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>See also start_of_run_date and 11_tick_count.</td>
</tr>
</tbody>
</table>

† In the example below, the displayf (or tracef) routine uses timer variables to time-stamp good BCCs on the DCE side. (Similar programming could determine the current date.) The tick unit selected on the FEB Setup menu is seconds. Adjust the program as needed for other tick units.

```c
extern unsigned long start_of_run_date,
start_of_run_time, 11_tick_count;
unsigned short seconds, hours, minutes, tick_mins, tick_secs, tick_hours;
#define SECS(run_time) ((unsigned short)(run_time & 0xff))
#define MINS(run_time) (((unsigned short)(run_time >> 8)) & 0xff)

STATE: time
CONDITIONS: DCE GOOD_BCC
ACTIONS:
{ 
tick_secs = 11_tick_count % 60;
tick_mins = (11_tick_count + SECS(start_of_run_time)) / 60;
tick_hours = (tick_mins + MINS(start_of_run_time)) / 60;
displayf("Time: %d:%d:%d
",
(unsigned short)((start_of_run_time >> 16) & 0xff) + tick_hours)%24,
(MINS(start_of_run_time) + tick_mins)%60,
(SECS(start_of_run_time) + tick_secs)%60);
} 
```
The other tick-counter is on the MPM and is referred to as the wall-time clock. This clock ticks once per millisecond and drives the timers displayed on the statistics results screens—at least while they are incrementing. At the moment a timer stops incrementing, the programmer can reach in and replace the incremented value with a timer value based the FEB tick-counter instead.

The current value of this wall-time tick-counter is available to the program via the `get_wall_time_286_ticks` routine. The current value always reflects both the number of ticks and the actual elapsed time ("wall time") since the program entered Run mode.

(B) Running

While it increments on the stats screen, a timer always is driven by wall-time ticks. To start a current timer incrementing, first you must have sent a stat message to correlate the timer structure with a timer line on the stats screen. At that point the simple statement `timer_name.running = -0` will start the timer. The value of the timer at any given time while it is running will be the MPM (wall-time) ticks minus the `timer_name.start_tick_value` plus any `timer_name.current` value.

To stop a timer, change `timer_name.running` to zero. The current column of the timer will immediately display the value of `timer_name.current` (zero, unless you have done something in your program to calculate the current value of the timer). The stats display will interpret `timer_name.current` as a value in microseconds and convert it to the unit selected for that timer line.

(C) Restart

The translator has two different versions of the timer restart action, depending on what condition precipitated the action. The first version is used if the condition was data-related (or EIA-related) and time ticks are enabled on the FEB Setup menu. Here is this data-timer version:

```c
unsigned long temp;
convert_tick_count (11_tick_count, &temp);
timer_name.current = 0;
timer_name.start_tick_value = temp;
timer_name.running = -0;
```

The `convert_tick_count` routine converts `11_tick_count` into microseconds and stores the result in `temp`. The value of `temp` is assigned immediately to `timer_name.start_tick_value`. When the 68010 sees that `timer_name.running` equals the one's complement of zero, it subtracts the start-tick value from the `11`-tick count and displays the difference in the current column of the timer line. Since the start-tick value was derived a moment before from the `11`-tick count, the difference will be zero. The current column on the stats screen should begin a timer at zero following a restart.
A slightly different version of the program is used if the condition was nondata-related or if time ticks are disabled in the FEB. The `convert_tick_count` routine is not used and the following routine is used in its place:

```c
get_wall_time_286_ticks (&temp);
```

This routine returns the current value of the wall-time tick-counter, in milliseconds zero-padded to microseconds. It stores the value in `temp` and the program proceeds as above.

(D) Continue

The timer-continue action is very similar to the restart. There are just two differences. One, the action is enclosed in an if statement that verifies that `timer_name.running` equals zero—that the timer actually is stopped, in other words; and two, `timer_name.current` is not set to zero, but retains the value it received the last time the timer stopped.

(E) Stop

Here is one of the two versions of a timer stop action:

```c
if (timer_name.running != 0)
{
    unsigned long temp;
    convert_tick_count (I1_tick_count, &temp);
    timer_name.current += temp - timer_name.start_tick_value;
    timer_name.running = 0;
}
```

In this translation, the start-tick value is subtracted from the current tick count, and any pending current value (held over if the timer was continued) is added in. The result is a new `timer_name.current` value. This value is posted to the stats screen as soon as the 68010 sees `timer_name.running = 0`.

The other version of the stop action uses `get_wall_time_286_ticks` instead of `convert_tick_count`.

(F) Sample Action

The code that produces the sample action is identical to the code that sampled a counter. See Section 62.1(B). The `timer_name.sample_count` variable's not equaling zero causes minimum, maximum, and average values to be displayed.

62.3 Accumulators

Shown below is the structure of an accumulator as the translator declares it (and as the 68010 accesses it to update the statistics screens). Also refer to Table 62-6. Note that there is no current value, since an accumulator neither counts nor times. There are no “previous” and “old” values, because in its spreadsheet implementation an accumulator never is tested in a Conditions block.
struct accumulator_struct
{
    unsigned long last;
    unsigned long maximum;
    unsigned long minimum;
    unsigned short sample_count;
    unsigned long total_high;
    unsigned short total_low_low;
    unsigned short total_low_high;
    unsigned short changed;
};

Here is the translator's version of an accumulate action when the object of the accumulation (selected by the user) was the maximum sampled value of a counter named framechar.

    accumulator_name.last = accumulator_framechar.maximum;
    if (accumulator_name.last > accumulator_name.maximum)
    {
        accumulator_name.maximum = accumulator_name.last;
    }
    if (accumulator_name.last < accumulator_name.minimum)
    {
        accumulator_name.minimum = accumulator_name.last;
    }

    accumulator_name.sample_count++;
    {
        unsigned long temp;
        temp = (accumulator_name.last & 0x0000ffff) + accumulator_name.total_low_low;
        accumulator_name.total_low_low = temp;
        temp = (accumulator_name.last >> 16) + accumulator_name.total_low_high + (temp >> 16);
        accumulator_name.total_low_high = temp;
        accumulator_name.total_high += temp >> 16;
    }
    accumulator_name.changed = -0;

A stat message must be sent so the CPM can correlate a line on the statistics results screen with the correct accumulator. The stat message is documented in the previous section on counters. The accumulator stat message is different only in respect that the stat_msg.type element should be set to 0 instead of 256.

The accumulator_name.sample_count variable's not equaling zero causes minimum, maximum, and average values to be displayed.
### Table 62-6
Accumulator Structures

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structure Name: accumulator_struct</td>
<td>Structure of an accumulator. Declared as type struct. Declared automatically by program when the user softkey-enters an ACCUMULATE action. Specific accumulator assigned to structure as follows: struct accumulator_struct accumulator_name. Reference a structure variable as follows: accumulator_name.last.</td>
<td></td>
</tr>
<tr>
<td>unsigned long</td>
<td>last</td>
<td>Value of last sample; displayed on the tabular statistics screen.</td>
</tr>
<tr>
<td>unsigned long</td>
<td>maximum</td>
<td>Maximum value of all samples; displayed on the tabular statistics screen.</td>
</tr>
<tr>
<td>unsigned long</td>
<td>minimum</td>
<td>Minimum value of all samples; displayed on the tabular statistics screen. Should be initialized as -0ul.</td>
</tr>
<tr>
<td>unsigned short</td>
<td>sample_count</td>
<td>Number of samples.</td>
</tr>
<tr>
<td>unsigned long</td>
<td>total_high</td>
<td>High four bytes of an eight-byte accumulator total.</td>
</tr>
<tr>
<td>unsigned short</td>
<td>total_low_low</td>
<td>Low two bytes of an eight-byte accumulator total.</td>
</tr>
<tr>
<td>unsigned short</td>
<td>total_low_high</td>
<td>Bytes 3 and 4 of an eight-byte accumulator total.</td>
</tr>
<tr>
<td>unsigned short</td>
<td>changed</td>
<td>For future use.</td>
</tr>
</tbody>
</table>

### 62.4 Routines

**get_68k_phys_addr**

**Synopsis**

```c
extern unsigned long get_68k_phys_addr(variable_ptr);
unsigned char * variable_ptr;
```

**Description**

This routine converts the address of a specified variable in the 80286 processors (MPM boards) to 68010 (CPM) format. This routine must be declared.

**Inputs**

The only parameter is the address to be converted.
Returns

The get_68k_phys_addr routine returns the converted address.

Example

See send_stat_message routine.

send_stat_message

Synopsis

extern void send_stat_message(struct_stat_msgytr);
struct stat_msg
{
    unsigned short op_type;
    unsigned short type;
    unsigned long object_name;
    unsigned long object_address;
};
struct stat_msg * struct_stat_msg_ptr;

Description

The send_stat_message routine sends the stat message structure to the 68010 CPU (CPM board). The current use of this routine sends the addresses of program counters, timers, and accumulators in the 80286 processors (MPM boards) to the CPM board where the tabular and graphic statistics displays are located.

The routine is called only one time in a program for each named counter, timer, or accumulator. Entering COUNTER as a condition or action (or TIMER or ACCUMUL as actions) via softkey on the Protocol Spreadsheet automatically declares the counter named and sends the stat message.

Inputs

The only parameter is a pointer to the structure of the stat message. For an explanation of the elements of the stat message, see Table 62-3.

Example

You plan on incrementing a counter named "dte_info" when a DTE Info frame is detected.

{struct
    {
        unsigned short op_type;
        unsigned short type;
        unsigned long object_name;
        unsigned long object_address;
    } stat_msg;

62-15
struct counter_structure
{
    unsigned long current;
    unsigned long last;
    unsigned long maximum;
    unsigned long minimum;
    unsigned long sample_count;
    unsigned long total_high;
    unsigned short total_low_low;
    unsigned short total_low_high;
    unsigned short out_of_range;
    unsigned short changed;
    unsigned long prev;
    unsigned long old;
};

struct counter_structure counter_dte_info = {0, 0, 0, -0ul};
extern unsigned long get_68kyhys_addr();

LAYER: 2
STATE: send_stat_message
    CONDITIONS: ENTER_STATE
    ACTIONS:
    {
        stat_msg.op_type = 2560;
        stat_msg.type = 256;
        stat_msg.object_name = get_68k_phys_addr("dte_info");
        stat_msg.object_address = get_68k_phys_addr(&counter_dte_info);
        send_stat_message(&stat_msg);
    }
    NEXT_STATE: count_info
STATE: count_info
    CONDITIONS: DTE INFO
    ACTIONS:
    {
        counter_dte_info.current++;
    }

get_wall_time_ticks

Synopsis

extern void get_wall_time_ticks(ticks_68k_format_ptr);
unsigned long * ticks_68k_format_ptr;

Description

The get_wall_time_ticks routine gets the number of wall-time ticks (in CPM storage format) from the time \([\text{time}]\) was hit. The wall clock gives millisecond resolution rounded to microseconds.

Inputs

The only input is a pointer to the location where the returned time-tick value will be stored.
Example

```c
{
    unsigned long ticks;
}
```

**LAYER: 2**

**STATE:** get_ticks

**CONDITIONS:** KEYBOARD

**ACTIONS:**

```c
{
    get_wall_time_ticks(&ticks);
}
```

**get_wall_time_286_ticks**

**Synopsis**

```c
extern void get_wall_time_286_ticks(unsigned long *ticks_286_format_ptr);
```

**Description**

The *get_wall_time_286_ticks* routine gets the number of wall-time ticks (in MPM storage format) from the time was hit. The wall clock gives millisecond readings rounded to microseconds. Use this routine prior to setting the *start_tick_value* in a timer action when *Time Ticks: OFF* has been selected on the Front-End Buffer Setup screen. Also use this routine to derive the *start_tick_value* if the condition is not line-related, e.g., KEYBOARD, even when time ticks are enabled on the FEB Setup menu.

**Inputs**

The only input is a pointer to the location where the returned time-tick value will be stored.

**Example**

```c
{
    unsigned long ticks_286;
}
```

**LAYER: 3**

**STATE:** getTicks

**CONDITIONS:** KEYBOARD

**ACTIONS:**

```c
{
    get_wall_time_286_ticks(&ticks_286);
    displayf("%lu", ticks_286);
}
```
convert_tick_count

Synopsis

extern void convert_tick_count(mpm_format_ticks, converted_ticks_ptr);
unsigned long mpm_format_ticks;
unsigned long * converted_ticks_ptr;

Description

The convert_tick_count routine converts a designated tick count into microseconds.

Use this routine to derive the start_tick_value for a timer action if ticks are enabled on the FEB Setup menu and the condition is line-related, e.g., RCV INFO.

Inputs

The first parameter is a designated tick count as long as it is in MPM storage format. It may be any of the layer tick counts. The unit of the ll_tick_count (and other layers' tick counts) value is determined on the Front End Buffer menu.

The second parameter is a pointer to the location where the returned tick count converted to microseconds will be stored.

Example

{ extern unsigned long ll_tick_count;
  unsigned long converted_ticks;
}  

LAYER: 1 
STATE: convert_ticks 
  CONDITIONS: RECEIVE GOOD_BCC 
  ACTIONS: 
  { 
    convert_tick_count(ll_tick_count, &converted_ticks);
    displayf("%lu", converted_ticks);
  }
Figure 63-1 Primitive Data Unit and sample Pointer-List Buffer being passed down the layers.
The most convenient tools for handling protocol headers while data is moving down and up the layers in the INTERVIEW are the spreadsheet SEND and GIVE_DATA actions in the various protocol packages. For instances when a protocol package is not loaded, such as when you are developing a new protocol or simply using a protocol that is not yet an option on the Layer Setup screen, OSI structures, variables, and routines in C become essential tools also.

63.1 Structures

The programmer may access the information in primitive data units conveniently by using a C structure as a multibyte pointer that is superimposed on data in the PDU’s. Before using a structure-pointer, it is necessary to understand the contents of IL buffers and primitive data units. All structures referenced may be found in Table 63-1.

(A) Interlayer Message Buffers

There are a maximum sixteen IL buffers in use at a given time. These buffers may be one of two kinds: data-character or pointer-list. In buffers being passed up the layers, data-character buffers (Figure 63-2) are always used. In buffers going down the layers, pointer-list buffers (Figure 63-1) are primarily used. The difference is that pointer-list buffers contain list-nodes which provide information about the location of data (or “lists”) inserted or referenced in the buffer, while data-character buffers do not.

1. **Header.** Each IL buffer contains a header that stores useful information such as the status of the maintain bits that prevent the buffer from being returned to the general pool; the position of the buffered data in the INTERVIEW’s display buffer; and the tick count (time) when the data was buffered from the line. (See il_buffer structure.)

2. **Service Data Unit.** The IL buffer also contains the data itself. This data component, the service data unit (or “SDU”), is added to as the buffer is passed down the layers, and subtracted from as a buffer travels up the layers. A data-character IL buffer includes all the data that was present when the data was first buffered, and the contents of this buffer do not change as the buffer is passed up the layers. What changes is the service data unit, derived from the data-start offset in the PDU.
The first part of the SDU in a pointer-list buffer is a *list-header node* (structure `il_list_header`) which contains information about the location of the first and last text nodes. As a buffer is passed down from Layer 3 to Layer 2 in X.25 (see Figure 63-1), a new text node containing a Layer 3 protocol header is inserted in buffer. Since the Layer 3 data will precede user data, the list node for the protocol information is referenced ahead of any other list nodes, changing the first-node reference in the list header. (If text is appended to the end of existing data, the list node referenced as last will change.)

The SDU in a pointer-list buffer also includes *list nodes* (structure `il_list_node`) which give a pointer to data, the length of the data pointed to, and the offset from the start of the buffer to the next list node.

Finally, the service data unit in all buffers includes *data*, whether copied into the buffer (usually protocol information) or located in memory outside of the buffer (usually user data).

![Diagram of PDU and data-character buffer](image)

*Figure 63-2* Primitive Data Unit and sample Data-Character Buffer being passed up the layers.
(B) Primitive Data Units

Like interlayer message buffers, PDU's have a format that is dependent on which direction the primitive is being passed. Refer again to Figure 63-1 and Figure 63-2.

1. *IL buffer number.* The buffer number to be passed with the primitive is always stored in the primitive. This buffer number is actually an 80286-processor segment number.

2. *Data-start offset.* The offset to the beginning of the service data unit for a given layer is different for the two types of buffers. In a pointer-list buffer going down the layers, the data-start offset will indicate the offset from the beginning of the buffer to the list-header node. This offset will vary if different linked lists have been started at different layers. Each list will have its own list header. In a data-character buffer going up the layers, the data-start offset will change from layer to layer. For example, a buffer containing X.25 data that is being passed from Layer 2 to Layer 3 will have an offset at Layer 3 two bytes beyond the offset at Layer 2.

3. *Data length.* The size of the SDU in a data-character buffer also varies from layer to layer. In the example just given, the SDU will be smaller by two bytes at Layer 3 than it was at Layer 2. In pointer-list buffers, the length of all data is unknown at any given layer.

(C) Accessing Information in Structures

There are two stages that are preliminary to accessing the information in these structures. The first step is to convert the 80286-processor segment number into a 32-bit address. The second stage is to place a pointer, in the shape of an IL buffer structure, at that address. Let's use an IL buffer as an example.

1. *Converting a segment number.* The IL-buffer segment number is returned any time you access one of the external, protocol-independent *il_buffer* variables listed in Table 63-1. These variables have names like *m_lo_dl_il_buff* and *up_n_il_buff*.

To make a pointer to an IL buffer, (1) shift the 80286 segment number to the left sixteen bits, since a full address in the 80286 is 32 bits long; (2) cast it as a *long*, so that the segment number is in the high 16 bits and the offset to a buffer for that segment is zero (the low 16 bits); and (3) cast it as a pointer. The following expression will take care of all three requirements:

```c
(void *) ((long) m_lo_dl_il_buff <<16);
```

Now you have a pointer to the first memory location of the most recent monitor-mode IL buffer passed up from Layer 2 to Layer 3. An upward-moving IL buffer was illustrated in Figure 63-2. The precise structure of both the IL buffer is given in the following declaration.
{
  struct il_buffer
  {
    unsigned short lock;
    unsigned short maintain_bits;
    unsigned short buffer_size;
    unsigned short transmit_tag;
    unsigned short receive_tag;
    unsigned long char_buff_frame_start;
    unsigned long char_buff_frame_end;
    unsigned short tick_count_high;
    unsigned short tick_count_mid;
    unsigned short tick_count_low;
    unsigned short available_space_offset;
    unsigned short bytes_remaining;
    unsigned long bcc_indicator;
    unsigned char data [4064];
  }
}

2. Create a structure-pointer at a given address. First, declare the structure of il_buffer, as indicated above. Then declare il_buffer_pointer as a structure-pointer, as follows:

  struct il_buffer * il_buffer_pointer;

Converting the segment number and assigning it to il_buffer_pointer may be accomplished with this one statement:

  il_buffer_pointer = (void *) ((long) m_io_d1 il_buff <<16);

Now a structure has been created around the most recent upward-moving IL buffer at Layer 3. This means that rather than moving a pointer around in the IL buffer, you can access elements in the buffer directly. The tick_count_low variable, for example, would be called il_buffer_pointer->tick_count_low. (The -> operator is used in place of the dot operator in structure-pointers.)

The first element of the data string would be called il_buffer_pointer->data[0]. Here is a program that displays on the prompt line the fifth data element, the packet-type byte, in every IL buffer that is monitored at Layer 3.
extern event m_lo_dl_prmtv;
extern volatile unsigned short m_lo_dl_il_buff;
struct il_buffer
{
    unsigned short lock;
    unsigned short maintain_bits;
    unsigned short buffer_size;
    unsigned short transmit_tag;
    unsigned short receive_tag;
    unsigned long char_buff_frame_start;
    unsigned long char_buff_frame_end;
    unsigned short tick_count_high;
    unsigned short tick_count_mid;
    unsigned short tick_count_low;
    unsigned short available_space_offset;
    unsigned short bytes_remaining;
    unsigned long bcc_indicator;
    unsigned char data [4064];
};
struct il_buffer * il_buffer_pointer;
}

LAYER: 3
STATE: monitor_il_buffers
CONDITIONS:
{
    m_lo_dl_prmtv
}
ACTIONS:
{
    il_buffer_pointer = (void *) ((long) m_lo_dl_il_buff <<16);
pos_cursor (0,0);
displayf ("%02x ", il_buffer_pointer->data[4]);
}

If you run this program, be sure to load in the Layer 2 and Layer 3 personality packages for X.25. These packages will take care of delivery of the monitor primitives to Layer 3.
Table 63-1
OSI Structures

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structure Name</td>
<td>pdu</td>
<td>Structure of an OSI primitive data unit (PDU). Declared as type struct. Use this structure as follows. Declare the entire structure. Make a pointer to a PDU by shifting m_lo_dl_pdu_seg (or up_n_pdu_seg) 16 bits to the left. Then convert this pointer to a pointer to a PDU structure: struct pdu * pdu_pointer. pdu_pointer = (void *)(long)m_lo_dl_pdu_seg &lt;&lt; 16. Reference a structure-pointer variable as follows: pdu_pointer-&gt;primitive_code.</td>
<td></td>
</tr>
<tr>
<td>unsigned char</td>
<td>primitive_code</td>
<td></td>
<td>Codes for OSI variables are listed in Table 63-2 through Table 63-8. For Layer 3 primitive codes, for example, refer to Table 63-4. The value of this variable is also stored in external variable m_lo_dl_prmvt_code (or up_n_prmv_code).</td>
</tr>
<tr>
<td>unsigned char</td>
<td>path</td>
<td>0-8</td>
<td>Path number, both directions. The value of this variable is also stored in external variable m_lo_dl_prmvt_path (or up_n_prmvt_path).</td>
</tr>
<tr>
<td>unsigned long</td>
<td>parameter</td>
<td></td>
<td>For future use. At present, under user control.</td>
</tr>
<tr>
<td>unsigned short</td>
<td>relay_baton</td>
<td></td>
<td>Maintain bit passed with an interlayer-message buffer, both directions. Zero in this variable identifies maintain bit.</td>
</tr>
<tr>
<td>unsigned short</td>
<td>il_buffer_number</td>
<td></td>
<td>Segment number of the interlayer-message buffer, both directions. The value of this variable is also stored in external variable m_lo_dl_il_buff (or up_n_il_buff).</td>
</tr>
<tr>
<td>unsigned char</td>
<td>buffer_contents</td>
<td>0</td>
<td>Contains data-character buffer type. Must be used for buffer being passed up.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Contains pointer-list buffer type. May be used for buffers being passed up, but is currently used primarily for buffers being passed down.</td>
</tr>
<tr>
<td>unsigned short</td>
<td>data_start_offset</td>
<td></td>
<td>Offset from the beginning of the buffer to the header node in the SDU of an interlayer-message buffer in an OSI primitive being sent down from a layer above, in a primitive being sent up from a layer below, it is the offset to the SDU. Varies according to the layer at which the buffer is located. For example, in a buffer passed up to Layer 3 from Layer 2, the offset would be to the beginning of the Layer 3 header, bypassing Layer 2 header information. The value of this variable is also stored in external variable m_lo_dl_sdu_offset (or up_n_sdu).</td>
</tr>
<tr>
<td>unsigned short</td>
<td>data_length</td>
<td></td>
<td>Length of the service data unit, including headers and user data. Only for primitives sent up from layer below. Varies with the layer where the buffer is located. For example, at Layer 3, length would exclude Layer 2 header (or trailer) information. The value of this variable is also stored in external variable m_lo_dl_sdu_size.</td>
</tr>
</tbody>
</table>
**Table 63-1 (continued)**

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Structure Name: il_buffer</td>
<td></td>
<td>Structure of an interlayer-message buffer, both directions. Declared as type <code>struct</code>. Use this structure as follows. Declare the entire structure. Make a pointer to an <code>il_buffer</code> by shifting <code>m_lo_dclllbuff</code> (or <code>up_n ll_buff</code>) 16 bits to the left: <code>il_buffer_pointer = (void *)((long)(lo_dclllbuff &lt;&lt; 16));</code> Then convert this pointer to a pointer to an <code>il_buffer</code> structure: <code>struct il_buffer * il_buffer_pointer</code>. Reference a structure-pointer variable as follows: <code>il_buffer_pointer-&gt;tick_count_low</code>.</td>
</tr>
<tr>
<td></td>
<td>lock</td>
<td>0</td>
<td>Internal variable which prevents structure from being updated by more than one program at the same time.</td>
</tr>
<tr>
<td></td>
<td>maintain_bits</td>
<td></td>
<td>Two-byte variable which provides the status of the maintain bits. A bit with a value of 1 is in use.</td>
</tr>
<tr>
<td></td>
<td>buffer_size</td>
<td>1000/4096</td>
<td>Currently, the only value. Bits 1–3 define bcc indication: no bcc, good bcc, bad bcc, abort, half bad bcc (DDCMP)</td>
</tr>
<tr>
<td></td>
<td>transmit_tag</td>
<td>0</td>
<td>Bits 1–3 define bcc indication: no bcc, good bcc, bad bcc, abort, half bad bcc (DDCMP)</td>
</tr>
<tr>
<td></td>
<td>receive_tag</td>
<td>0</td>
<td>Bits 4–8 for future use. Bits 4 identifies side of the line: td, rd</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Bit 4 identifies side of the line: td, rd</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Bit 5—message buffer overflow: frame fits in buffer, frame too large for the buffer</td>
</tr>
<tr>
<td></td>
<td>char_buff_frame_start</td>
<td></td>
<td>Location in the character buffer of the start of the buffered data.</td>
</tr>
<tr>
<td></td>
<td>char_buff_frame_end</td>
<td>1</td>
<td>Location in the character buffer of the end of the buffered data.</td>
</tr>
</tbody>
</table>

*(il_buffer structure continued on next page)*
### Table 63-1 (continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned short</td>
<td>tick_count_high</td>
<td></td>
<td>Value of internal variable that counts the number of times ( I_1 _\text{tick} _\text{count} ) has reached its maximum value. Together, the three ( \text{il_buffer} ) tick-count variables preserve at each layer the original time when the end of the data (BCC) was clocked into the buffer.</td>
</tr>
<tr>
<td>unsigned short</td>
<td>tick_count_mid</td>
<td></td>
<td>16 high-order bits of 32-bit ( I_1 _\text{tick} _\text{count} ).</td>
</tr>
<tr>
<td>unsigned short</td>
<td>tick_count_low</td>
<td></td>
<td>16 low-order bits of 32-bit ( I_1 _\text{tick} _\text{count} ).</td>
</tr>
<tr>
<td>unsigned short</td>
<td>available_space_offset</td>
<td></td>
<td>Offset to the next available space in the interlayer-message buffer.</td>
</tr>
<tr>
<td>unsigned short</td>
<td>bytes_remaining</td>
<td>0</td>
<td>Available number of bytes remaining in the buffer.</td>
</tr>
<tr>
<td>unsigned long</td>
<td>bcc_indicator</td>
<td>reserved</td>
<td>Contains all data including each layer's header information, as well as the first of two block check characters. Does not vary from layer to layer.</td>
</tr>
<tr>
<td>unsigned char</td>
<td>data [4064]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Structure Name:** \( \text{il\_list\_header} \)

Structure of the header node in an interlayer-message buffer. Only for primitives sent down from the layer above. Declared as type `struct`. Use this structure as follows.

Declare the entire structure. Make a pointer to an \( \text{il\_list\_header} \) by shifting \( \text{up\_n\_il\_buff} \) (or \( \text{m\_lo\_dl\_il\_buff} \)) 16 bits to the left and adding the `data_start_offset` from the PDU structure (also stored as external variable \( \text{up\_n\_sdu} \) or \( \text{m\_lo\_dl\_sdu\_offset} \)):

\[
\text{il\_list\_header\_pointer} = \text{(void *)}((\text{long})\text{up\_n\_il\_buff} << 16) + \text{up\_n\_sdu}).
\]

Then convert this pointer into a pointer to an \( \text{il\_list\_header} \) structure:

\[
\text{struct} \text{il\_list\_header* il\_list\_header\_pointer}.
\]

Reference a structure-pointer variable as follows:

\[
\text{il\_list\_header\_pointer->last\_node\_offset}.
\]

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned short</td>
<td>first_node_offset</td>
<td></td>
<td>Offset from the beginning of the buffer to the first text node in the buffer. Varies according to the layer at which the buffer is located. At Layer 2, the offset would be to different starting node than at Layer 3.</td>
</tr>
<tr>
<td>unsigned short</td>
<td>last_node_offset</td>
<td></td>
<td>Offset to the location of the last text node in the buffer, from the beginning of the buffer.</td>
</tr>
<tr>
<td>unsigned long</td>
<td>reserved</td>
<td></td>
<td>reserved</td>
</tr>
<tr>
<td>Type</td>
<td>Variable</td>
<td>Value (hex/decimal)</td>
<td>Meaning</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------</td>
<td>--------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Structure Name:</td>
<td>il_list_node</td>
<td>Structure of text nodes in an interlayer-message buffer. Only for primitives sent down from the layer above. Declared as type struct. Use this structure as follows. Declare the entire structure. Make a pointer to an il_list_node by shifting up_n_il_buff (or m_lo_dj/ il_buff) 16 bits to the left and adding the first_node_offset (or last_node_offset) from the il_list_header structure: il_list_node_pointer = (void*)((long)up_n_il_buff « 16) + il_list_header_pointer-&gt;first_node_offset. Point to the next node as follows: next_node_pointer = (il_list_node_pointer + il_list_node_pointer-&gt;next_node_offset).</td>
<td></td>
</tr>
<tr>
<td>unsigned char *</td>
<td>data_pointer</td>
<td>Pointer to the data in a text node.</td>
<td></td>
</tr>
<tr>
<td>unsigned short</td>
<td>data_length</td>
<td>Length of the data in a text node.</td>
<td></td>
</tr>
<tr>
<td>unsigned short</td>
<td>next_node_offset</td>
<td>Offset to the location of the next text node in the buffer, from the beginning of the buffer. Generally, there is a text node for each layer’s header information and one for the user data. A buffer that started at Layer 3 would have two text nodes, one for Layer 3 header information and one for user data (if any). At Layer 2, the buffer would acquire an additional text node.</td>
<td></td>
</tr>
</tbody>
</table>
63.2 Variables

OSI variables are layer-specific. The information stored in the OSI variables may be obtained by using the structure-pointer to IL buffers and primitives. But rather than requiring the user to repeat this process at each layer as a buffer moves through the layers, monitor and emulate variables have been made available at Layers 2–7 to store layer-specific, as well as general, information: the interlayer-buffer number, the offset to the service data unit, the path number, the size of the SDU, the segment number of the PDU, etc. There are also event variables which indicate that a primitive has been received at a given layer. Table 63-2 through Table 63-8 give the current OSI variables and their meanings.

The exchange of connect primitives shown primarily in Figure 30-4 is demonstrated in Figure 63-3 using C variables and routines. The SEND actions insert data in a buffer and send the buffer in a DATA REQ primitive. See Section 63.3 for an explanation of the _insert_il_buff_list_cnt and send primitive routines. The conditions use event variables to detect primitives and non-event variables to identify specific primitive types.

Figure 63-3 Layer 3 uses connect primitives to be sure that the Layer 2 entity below has established a link.
Table 63-2
Layer 1 OSI Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern volatile unsigned char</td>
<td>ph_prmtv_type</td>
<td>20/32</td>
<td>ph activate req</td>
</tr>
<tr>
<td></td>
<td></td>
<td>21/33</td>
<td>ph activate ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>22/34</td>
<td>ph activate resp</td>
</tr>
<tr>
<td></td>
<td></td>
<td>23/35</td>
<td>ph activate conf</td>
</tr>
<tr>
<td></td>
<td></td>
<td>24/36</td>
<td>ph data req</td>
</tr>
<tr>
<td></td>
<td></td>
<td>25/37</td>
<td>ph data ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2a/42</td>
<td>ph reset req</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2b/43</td>
<td>ph reset ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2c/44</td>
<td>ph reset resp</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2d/45</td>
<td>ph reset conf</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2e/46</td>
<td>ph deactivate req</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2f/47</td>
<td>ph deactivate ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>30/48</td>
<td>ph debug req</td>
</tr>
<tr>
<td></td>
<td></td>
<td>31/49</td>
<td>ph debug ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>33/51</td>
<td>ph error report ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>34/52</td>
<td>ph xmit req</td>
</tr>
<tr>
<td></td>
<td></td>
<td>35/53</td>
<td>ph set idle req</td>
</tr>
<tr>
<td></td>
<td></td>
<td>38/56</td>
<td>ph mgt facility req</td>
</tr>
<tr>
<td></td>
<td></td>
<td>39/57</td>
<td>ph mgt facility ind</td>
</tr>
</tbody>
</table>

OSI primitive code for primitives moving between Layers 1 and 2. Line Setup configured for emulate mode only.
### Table 63-3
Layer 2 OSI Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern event</td>
<td>lo_ph_prmtv</td>
<td>True when an OSI primitive is received at Layer 2 from Layer 1. Line Setup configured for emulate mode only.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>m_lo_ph_prmtv</td>
<td>True when an OSI primitive is received at Layer 2 from Layer 1. Line Setup configured for emulate or monitor mode.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>up_dl_prmtv</td>
<td>True when an OSI primitive is received at Layer 2 from Layer 3. Line Setup configured for emulate mode only.</td>
<td></td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>lo_ph_pdu_seg</td>
<td>OSI primitive data unit (PDU) IAPX-286 segment number received at Layer 2 from Layer 1. This segment number can be converted to a pointer by shifting it left 16 bits. Line Setup configured for emulate mode only.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>m_lo_ph_pdu_seg</td>
<td>OSI primitive data unit (PDU) IAPX-286 segment number received at Layer 2 from Layer 1. This segment number can be converted to a pointer by shifting it left 16 bits. Line Setup configured for emulate or monitor mode.</td>
<td></td>
</tr>
</tbody>
</table>
| extern volatile const unsigned char | lo_ph_prmtv_code | 21/33 ph activate ind  
23/35 ph activate conf  
25/37 ph data ind  
2b/43 ph reset ind  
2d/45 ph reset conf  
2f/47 ph deactivate ind  
31/49 ph debug ind  
33/51 ph error report ind  
35/57 ph mgt facility ind | OSI primitive code received at Layer 2 in a PDU from Layer 1. Line Setup configured for emulate mode only. |
|                            | m_lo_ph_prmtv_code | 24/36 td ph data ind  
27/38 rd ph data ind | OSI primitive code received at Layer 2 in a PDU from Layer 1. Line Setup configured for emulate or monitor mode. |
### Table 63-3 (continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern volatile const unsigned char</td>
<td>lo_ph_prmtv_path</td>
<td>0-8</td>
<td>Path number received at Layer 2 in a PDU from Layer 1. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_lo_ph_prmtv_path</td>
<td>0-8</td>
<td>Path number received at Layer 2 in a PDU from Layer 1. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>lo_ph_il_buff</td>
<td></td>
<td>Interlayer-buffer number (an IAPX-286 segment number) received at Layer 2 in a PDU from Layer 1. This segment number can be converted to a pointer by shifting it left 16 bits. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>m_lo_ph_il_buff</td>
<td></td>
<td>Interlayer-buffer number (an IAPX-286 segment number) received at Layer 2 in a PDU from Layer 1. This segment number can be converted to a pointer by shifting it left 16 bits. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>lo_ph_sdu</td>
<td></td>
<td>In OSI primitive received at Layer 2 from Layer 1, the offset to where the service data unit begins. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>m_lo_ph_sdu_offset</td>
<td></td>
<td>In OSI primitive received at Layer 2 from Layer 1, the offset to where the service data unit begins. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>m_lo_ph_sdu_size</td>
<td></td>
<td>Size of the service data unit in an interlayer-message buffer, displayed as SIZE on the Layer 2 trace screen. Received at Layer 2 from Layer 1. Same as data_length in a PDU. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>up_dli_pdu_seg</td>
<td></td>
<td>OSI primitive data unit (PDU) IAPX-286 segment number received at Layer 2 from Layer 3. This segment number can be converted to a pointer by shifting it left 16 bits. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>Type</td>
<td>Variable</td>
<td>Value (hex/decimal)</td>
<td>Meaning</td>
</tr>
<tr>
<td>------</td>
<td>----------</td>
<td>---------------------</td>
<td>---------</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>up_dl_prmtv_code</td>
<td>40/64</td>
<td>dl conn req</td>
</tr>
<tr>
<td></td>
<td></td>
<td>42/66</td>
<td>dl conn resp</td>
</tr>
<tr>
<td></td>
<td></td>
<td>44/68</td>
<td>dl data req</td>
</tr>
<tr>
<td></td>
<td></td>
<td>48/72</td>
<td>dl expd data req</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4a/74</td>
<td>dl reset req</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4c/76</td>
<td>dl reset resp</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4e/78</td>
<td>dl disconn req</td>
</tr>
<tr>
<td></td>
<td></td>
<td>50/80</td>
<td>dl debug req</td>
</tr>
<tr>
<td></td>
<td></td>
<td>52/82</td>
<td>dl unit data req</td>
</tr>
<tr>
<td></td>
<td></td>
<td>58/88</td>
<td>dl mgt facility req</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>OSI primitive code received at Layer 2 in a PDU from Layer 3. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>up_dl_prmtv_path</td>
<td>0-8</td>
<td>Path number received at Layer 2 in a PDU from Layer 3. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>up_dl_il_buff</td>
<td></td>
<td>Interlayer-buffer number (an IAPX-256 segment number) received at Layer 2 in a PDU from Layer 3. This segment number can be converted to a pointer by shifting it left 16 bits. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>up_dl_sdu</td>
<td></td>
<td>Offset to the start (header node) of the service data unit in an interlayer-message buffer. Received at Layer 2 from Layer 3. Same as data_start_offset in a PDU. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern unsigned long</td>
<td>l2_tick_count</td>
<td></td>
<td>32-bit l1_tick_count stored in header of most recent IL buffer passed up to Layer 2. Preserves at each layer the original time when the end of the data (BCC) was clocked into the buffer. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>Type</td>
<td>Variable</td>
<td>Value (hex/decimal)</td>
<td>Meaning</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------------------</td>
<td>--------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>extern event</td>
<td>lo_di_prmtv</td>
<td></td>
<td>True when an OSI primitive is received at Layer 3 from Layer 2. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern event</td>
<td>m_lo_di_prmtv</td>
<td></td>
<td>True when an OSI primitive is received at Layer 3 from Layer 2. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern event</td>
<td>up_n_prmtv</td>
<td></td>
<td>True when an OSI primitive is received at Layer 3 from Layer 4. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>lo_di_pdu_seg</td>
<td></td>
<td>OSI primitive data unit (PDU) IAPX-286 segment number received at Layer 3 from Layer 2. This segment number can be converted to a pointer by shifting it left 16 bits. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>m_lo_di_pdu_seg</td>
<td></td>
<td>OSI primitive data unit (PDU) IAPX-286 segment number received at Layer 3 from Layer 2. This segment number can be converted to a pointer by shifting it left 16 bits. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>lo_di_prmtv_code</td>
<td>41/85, 43/67, 45/69, 49/73, 4b/75, 4d/77, 4f/79, 51/81, 53/83, 55/85, 59/89</td>
<td>dl conn ind, dl conn conf, dl data ind, dl expd data ind, dl reset ind, dl reset conf, dl disconnect ind, dl debug ind, dl unit data ind, dl error report ind, dl mgt facility ind</td>
</tr>
</tbody>
</table>
### Table 63-4 (continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_lo_di_prmtv_code</td>
<td>44/68</td>
<td>td dl data ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>45/69</td>
<td>rd dl data ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>48/72</td>
<td>td dl expd data ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>49/73</td>
<td>rd dl expd data ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>54/84</td>
<td>td dl unit data ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>55/85</td>
<td>rd dl unit data ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>OSI primitive code received at Layer 3 in a PDU from Layer 2. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>lo_di_prmtv_path</td>
<td>0-8</td>
<td>Path number received at Layer 3 in a PDU from Layer 2. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_lo_di_prmtv_path</td>
<td>0-8</td>
<td>Path number received at Layer 3 in a PDU from Layer 2. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>lo_di_il_buff</td>
<td></td>
<td>Interlayer-buffer number (an IAPX-286 segment number) received at Layer 3 in a PDU from Layer 2. This segment number can be converted to a pointer by shifting it left 16 bits. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>m_lo_di_il_buff</td>
<td></td>
<td>Interlayer-buffer number (an IAPX-286 segment number) received at Layer 3 in a PDU from Layer 2. This segment number can be converted to a pointer by shifting it left 16 bits. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>lo_di_sdu</td>
<td></td>
<td>In OSI primitive received at Layer 3 from Layer 2, the offset to where the service data unit begins. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>m_lo_di_sdu_offset</td>
<td></td>
<td>In OSI primitive received at Layer 3 from Layer 2, the offset to where the service data unit begins. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>m_lo_di_sdu_size</td>
<td></td>
<td>Size of the service data unit in an interlayer-message buffer, displayed as SIZE on the Layer 3 trace screen. Received at Layer 3 from Layer 2. Same as <code>data_length</code> in a PDU. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>Type</td>
<td>Variable</td>
<td>Value (hex/decimal)</td>
<td>Meaning</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------</td>
<td>--------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>extern volatile short</td>
<td>up_n_pdu_seg</td>
<td></td>
<td>OSI primitive data unit (PDU) IAPX-286 segment number received at Layer 3 from Layer 4. This segment number can be converted to a pointer by shifting it left 16 bits. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>up_n_prmvt_code</td>
<td>60/96</td>
<td>n conn req</td>
</tr>
<tr>
<td></td>
<td></td>
<td>62/98</td>
<td>n conn resp</td>
</tr>
<tr>
<td></td>
<td></td>
<td>64/100</td>
<td>n data req</td>
</tr>
<tr>
<td></td>
<td></td>
<td>66/102</td>
<td>n data ack req</td>
</tr>
<tr>
<td></td>
<td></td>
<td>68/104</td>
<td>n expd data req</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6a/106</td>
<td>n reset req</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6c/108</td>
<td>n reset resp</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6e/110</td>
<td>n disconn req</td>
</tr>
<tr>
<td></td>
<td></td>
<td>70/112</td>
<td>n debug req</td>
</tr>
<tr>
<td></td>
<td></td>
<td>72/114</td>
<td>n unit data req</td>
</tr>
<tr>
<td></td>
<td></td>
<td>74/116</td>
<td>n qual data req</td>
</tr>
<tr>
<td></td>
<td></td>
<td>76/118</td>
<td>n qual data ack req</td>
</tr>
<tr>
<td></td>
<td></td>
<td>78/120</td>
<td>n mgt facility req</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>up_n_prmvt_path</td>
<td>0-8</td>
<td>Path number received at Layer 3 in a PDU from Layer 4. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>up_n_il_buff</td>
<td></td>
<td>Interlayer-buffer number (an IAPX-286 segment number) received at Layer 3 in a PDU from Layer 4. This segment number can be converted to a pointer by shifting it left 16 bits. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>up_n_sdu</td>
<td></td>
<td>Offset to the start (header node) of the service data unit in an interlayer-message buffer. Received at Layer 3 from Layer 4. Same as data_start_offset in a PDU. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern unsigned long</td>
<td>i3_tick_count</td>
<td></td>
<td>32-bit i1_tick_count stored in header of most recent IL buffer passed up to Layer 3. Preserves at each layer the original time when the end of the data (BCC) was clocked into the buffer. Line Setup configured for emulate or monitor mode.</td>
</tr>
</tbody>
</table>
Table 63-5
Layer 4 OSI Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern event</td>
<td>lo_n_prmtv</td>
<td></td>
<td>True when an OSI primitive is received at Layer 4 from Layer 3. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern event</td>
<td>m_lo_n_prmtv</td>
<td></td>
<td>True when an OSI primitive is received at Layer 4 from Layer 3. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern event</td>
<td>up_t_prmtv</td>
<td></td>
<td>True when an OSI primitive is received at Layer 4 from Layer 5. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>lo_n_pdu_seg</td>
<td></td>
<td>OSI primitive data unit (PDU) IAPX-286 segment number received at Layer 4 from Layer 3. This segment number can be converted to a pointer by shifting it left 16 bits. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>m_lo_n_pdu_seg</td>
<td></td>
<td>OSI primitive data unit (PDU) IAPX-286 segment number received at Layer 4 from Layer 3. This segment number can be converted to a pointer by shifting it left 16 bits. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>lo_n_prmtv_code</td>
<td>61/97 63/99 65/101 67/103 69/105 6b/107 6d/109 6f/111 71/113 73/115 75/117 77/119 79/121 7a/122</td>
<td>n conn n conn conf n data n data ack ind n expd data ind n reset ind n reset conf n disconn ind n debug ind n unit data ind n qual data ind n qual data ack ind n mgt facility ind n error report ind</td>
</tr>
</tbody>
</table>

OSI primitive code received at Layer 4 in a PDU from Layer 3. Line Setup configured for emulate mode only.
Table 63-5 (continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_lo_n_prmtv_code</td>
<td>64/100</td>
<td>td n data ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>65/101</td>
<td>rd n data ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>68/102</td>
<td>td n expd data ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>69/103</td>
<td>rd n expd data ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>74/116</td>
<td>td n unit data ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>75/117</td>
<td>rd n unit data ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>OSI primitive code received at Layer 4 in a PDU from Layer 3. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>lo_n_prmtv_path</td>
<td>0-8</td>
<td>Path number received at Layer 4 in a PDU from Layer 3. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_lo_n_prmtv_path</td>
<td>0-8</td>
<td>Path number received at Layer 4 in a PDU from Layer 3. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>lo_n_il_buff</td>
<td></td>
<td>Interlayer-buffer number (an IAPX-286 segment number) received at Layer 4 in a PDU from Layer 3. This segment number can be converted to a pointer by shifting it left 16 bits. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>m_lo_n_il_buff</td>
<td></td>
<td>Interlayer-buffer number (an IAPX-286 segment number) received at Layer 4 in a PDU from Layer 3. This segment number can be converted to a pointer by shifting it left 16 bits. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>lo_n_sdu</td>
<td></td>
<td>In OSI primitive received at Layer 4 from Layer 3, the offset to where the service data unit begins. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>m_lo_n_sdu_offset</td>
<td></td>
<td>In OSI primitive received at Layer 4 from Layer 3, the offset to where the service data unit begins. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>m_lo_n_sdu_size</td>
<td></td>
<td>Size of the service data unit in an Interlayer-message buffer. Received at Layer 4 from Layer 3. Same as data_length in a PDU. Line Setup configured for emulate or monitor mode.</td>
</tr>
</tbody>
</table>

63-21
### Table 63-5 (continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern volatile unsigned short</td>
<td>up_t_pdu_seg</td>
<td></td>
<td>OSI primitive data unit (PDU) IAPX-286 segment number received at Layer 4 from Layer 5. This segment number can be converted to a pointer by shifting it left 16 bits. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>up_t_prmtv_code</td>
<td>80/128</td>
<td>t conn req</td>
</tr>
<tr>
<td></td>
<td></td>
<td>82/130</td>
<td>t conn resp</td>
</tr>
<tr>
<td></td>
<td></td>
<td>84/132</td>
<td>t data req</td>
</tr>
<tr>
<td></td>
<td></td>
<td>88/136</td>
<td>t expd data req</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8e/142</td>
<td>t disconn req</td>
</tr>
<tr>
<td></td>
<td></td>
<td>90/144</td>
<td>t debug req</td>
</tr>
<tr>
<td></td>
<td></td>
<td>92/146</td>
<td>t unit data req</td>
</tr>
<tr>
<td></td>
<td></td>
<td>98/152</td>
<td>t mgt facility req</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>up_t_prmtv_path</td>
<td>0-8</td>
<td>Path number received at Layer 4 in a PDU from Layer 5. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>up_t_ll_buff</td>
<td></td>
<td>Interlayer-buffer number (an IAPX-286 segment number) received at Layer 4 in a PDU from Layer 5. This segment number can be converted to a pointer by shifting it left 16 bits. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>up_t_sdu</td>
<td></td>
<td>Offset to the start (header node) of the service data unit in an interlayer-message buffer. Received at Layer 4 from Layer 5. Same as data_start_offset in a PDU. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern unsigned long</td>
<td>l4_tick_count</td>
<td></td>
<td>32-bit l1_tick_count stored in header of most recent IL buffer passed up to Layer 4. Preserves at each layer the original time when the end of the data (BCC) was clocked into the buffer. Line Setup configured for emulate or monitor mode.</td>
</tr>
</tbody>
</table>
### Table 63-6
Layer 5 OSI Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern event</td>
<td>lo_t_prmtv</td>
<td></td>
<td>True when an OSI primitive is received at Layer 5 from Layer 4. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern event</td>
<td>m_lo_t_prmtv</td>
<td></td>
<td>True when an OSI primitive is received at Layer 5 from Layer 4. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern event</td>
<td>up_s_prmtv</td>
<td></td>
<td>True when an OSI primitive is received at Layer 5 from Layer 6. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>lo_t_pdu_seg</td>
<td></td>
<td>OSI primitive data unit (PDU) IAPX-286 segment number received at Layer 5 from Layer 4. This segment number can be converted to a pointer by shifting it left 16 bits. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>m_lo_t_pdu_seg</td>
<td></td>
<td>OSI primitive data unit (PDU) IAPX-286 segment number received at Layer 5 from Layer 4. This segment number can be converted to a pointer by shifting it left 16 bits. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>lo_t_prmtv_code</td>
<td>81/129</td>
<td>t conn ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>83/131</td>
<td>t conn conf</td>
</tr>
<tr>
<td></td>
<td></td>
<td>85/133</td>
<td>t data ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>89/137</td>
<td>t expd data ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8f/143</td>
<td>t disconn ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>91/145</td>
<td>t debug ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>93/147</td>
<td>t unit data ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>95/149</td>
<td>t error report ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>99/153</td>
<td>t mgt facility ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>OSI primitive code received at Layer 5 in a PDU from Layer 4. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_lo_t_prmtv_code</td>
<td>84/132</td>
<td>td t data ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>85/133</td>
<td>rd t data ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>88/136</td>
<td>td t expd data ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>89/137</td>
<td>rd t expd data ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>94/148</td>
<td>td t unit data ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>95/149</td>
<td>rd t unit data ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>OSI primitive code received at Layer 5 in a PDU from Layer 4. Line Setup configured for emulate or monitor mode.</td>
</tr>
</tbody>
</table>
### Table 63-6 (continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern volatile const unsigned char</td>
<td>lo_t_prmtv_path</td>
<td>0-8</td>
<td>Path number received at Layer 5 in a PDU from Layer 4. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_lo_t_prmtv_path</td>
<td>0-8</td>
<td>Path number received at Layer 5 in a PDU from Layer 4. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>lo_t_il_buff</td>
<td></td>
<td>Interlayer-buffer number (an IAPX-286 segment number) received at Layer 5 in a PDU from Layer 4. This segment number can be converted to a pointer by shifting it left 16 bits. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>m_lo_t_il_buff</td>
<td></td>
<td>Interlayer-buffer number (an IAPX-286 segment number) received at Layer 5 in a PDU from Layer 4. This segment number can be converted to a pointer by shifting it left 16 bits. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>lo_t_sdu</td>
<td></td>
<td>In OSI primitive received at Layer 5 from Layer 4, the offset to where the service data unit begins. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>m_lo_t_sdu_offset</td>
<td></td>
<td>In OSI primitive received at Layer 5 from Layer 4, the offset to where the service data unit begins. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>m_lo_t_sdu_size</td>
<td></td>
<td>Size of the service data unit in an Interlayer-message buffer. Received at Layer 5 from Layer 4. Same as <em>data_length</em> in a PDU. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>up_s_pdu_seg</td>
<td></td>
<td>OSI primitive data unit (PDU) IAPX-286 segment number received at Layer 5 from Layer 6. This segment number can be converted to a pointer by shifting it left 16 bits. Line Setup configured for emulate mode only.</td>
</tr>
</tbody>
</table>

63-24
<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern volatile const unsigned char</td>
<td>up_s_prmtv_code</td>
<td>a0/160</td>
<td>s conn req</td>
</tr>
<tr>
<td></td>
<td></td>
<td>a2/162</td>
<td>s conn resp</td>
</tr>
<tr>
<td></td>
<td></td>
<td>a4/164</td>
<td>s data req</td>
</tr>
<tr>
<td></td>
<td></td>
<td>a8/168</td>
<td>s expd data req</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ac/172</td>
<td>s release req</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ae/174</td>
<td>s release resp</td>
</tr>
<tr>
<td></td>
<td></td>
<td>b0/176</td>
<td>s debug req</td>
</tr>
<tr>
<td></td>
<td></td>
<td>b2/178</td>
<td>s unit data req</td>
</tr>
<tr>
<td></td>
<td></td>
<td>b8/184</td>
<td>s mgt facility req</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>OSI primitive code received at Layer 5 in a PDU from Layer 6. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>up_s_prmtv_path</td>
<td>0-8</td>
<td>Path number received at Layer 5 in a PDU from Layer 6. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>up_s_il_buff</td>
<td></td>
<td>Interlayer-buffer number (an IAPX-286 segment number) received at Layer 5 in a PDU from Layer 6. This segment number can be converted to a pointer by shifting it left 16 bits. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>up_s_sdu</td>
<td></td>
<td>Offset to the start (header node) of the service data unit in an interlayer-message buffer. Received at Layer 5 from Layer 6. Same as data_start_offset in a PDU. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern unsigned long</td>
<td>l5_tick_count</td>
<td></td>
<td>32-bit l1_tick_count stored in header of most recent IL buffer passed up to Layer 5. Preserves at each layer the original time when the end of the data (BCC) was clocked into the buffer. Line Setup configured for emulate or monitor mode.</td>
</tr>
</tbody>
</table>
### Table 63-7
Layer 6 OSI Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern event</td>
<td>lo_s_prmtv</td>
<td></td>
<td>True when an OSI primitive is received at Layer 6 from Layer 5. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern event</td>
<td>m_lo_s_prmtv</td>
<td></td>
<td>True when an OSI primitive is received at Layer 6 from Layer 5. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern event</td>
<td>up_p_prmtv</td>
<td></td>
<td>True when an OSI primitive is received at Layer 6 from Layer 7. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>lo_s_pdu_seg</td>
<td></td>
<td>OSI primitive data unit (PDU) IAPX-286 segment number received at Layer 6 from Layer 5. This segment number can be converted to a pointer by shifting it left 16 bits. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>m_lo_s_pdu_seg</td>
<td></td>
<td>OSI primitive data unit (PDU) IAPX-286 segment number received at Layer 6 from Layer 5. This segment number can be converted to a pointer by shifting it left 16 bits. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>lo_s_prmtv_code</td>
<td>a1/161</td>
<td>s conn ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>a3/163</td>
<td>s conn conf</td>
</tr>
<tr>
<td></td>
<td></td>
<td>a5/165</td>
<td>s data ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>a9/169</td>
<td>s expd data ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ad/173</td>
<td>s release ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>af/175</td>
<td>s release conf</td>
</tr>
<tr>
<td></td>
<td></td>
<td>b1/177</td>
<td>s debug ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>b3/179</td>
<td>s unit data ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>b5/181</td>
<td>s error report ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>b9/185</td>
<td>s mgt facility ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>OSI primitive code received at Layer 6 in a PDU from Layer 5. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_lo_s_prmtv_code</td>
<td>a4/164</td>
<td>td s data ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>a5/165</td>
<td>rd s data ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>a8/168</td>
<td>td s expd data ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>a9/169</td>
<td>rd s expd data ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>b4/180</td>
<td>td s unit data ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>b5/181</td>
<td>rd s unit data ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>OSI primitive code received at Layer 6 in a PDU from Layer 5. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>Type</td>
<td>Variable</td>
<td>Value (hex/decimal)</td>
<td>Meaning</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>-----------------------</td>
<td>--------------------</td>
<td>---------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>lo_s_prmtv_path</td>
<td>0-8</td>
<td>Path number received at Layer 6 in a PDU from Layer 5. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_lo_s_prmtv_path</td>
<td>0-8</td>
<td>Path number received at Layer 6 in a PDU from Layer 5. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>lo_s_il_buff</td>
<td></td>
<td>Interlayer-buffer number (an IAPX-286 segment number) received at Layer 6 in a PDU from Layer 5. This segment number can be converted to a pointer by shifting it left 16 bits. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>m_lo_s_il_buff</td>
<td></td>
<td>Interlayer-buffer number (an IAPX-286 segment number) received at Layer 6 in a PDU from Layer 5. This segment number can be converted to a pointer by shifting it left 16 bits. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>lo_s_sdu</td>
<td></td>
<td>In OSI primitive received at Layer 6 from Layer 5, the offset to where the service data unit begins. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>m_lo_s_sdu_offset</td>
<td></td>
<td>In OSI primitive received at Layer 6 from Layer 5, the offset to where the service data unit begins. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>m_lo_s_sdu_size</td>
<td></td>
<td>Size of the service data unit in an interlayer-message buffer. Received at Layer 6 from Layer 5. Same as data_length in a PDU. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>up_p_pdu_seg</td>
<td></td>
<td>OSI primitive data unit (PDU) IAPX-286 segment number received at Layer 6 from Layer 7. This segment number can be converted to a pointer by shifting it left 16 bits. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>Type</td>
<td>Variable</td>
<td>Value (hex/decimal)</td>
<td>Meaning</td>
</tr>
<tr>
<td>------</td>
<td>-------------------</td>
<td>---------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>up_p_prmtv_code</td>
<td>c0/192</td>
<td>p conn req</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c2/194</td>
<td>p conn resp</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c4/196</td>
<td>p data req</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c8/200</td>
<td>p expd data req</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cc/204</td>
<td>p release req</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ce/206</td>
<td>p release resp</td>
</tr>
<tr>
<td></td>
<td></td>
<td>d0/208</td>
<td>p debug req</td>
</tr>
<tr>
<td></td>
<td></td>
<td>d2/210</td>
<td>p unit data req</td>
</tr>
<tr>
<td></td>
<td></td>
<td>d8/216</td>
<td>p mgt facility req</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>OSI primitive code received at Layer 6 from Layer 7 in a PDU.</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Line Setup configured for emulate mode only.</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Path number received at Layer 6 from Layer 7 in a PDU.</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Line Setup configured for emulate mode only.</strong></td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>up_p_prmtv_path</td>
<td>0-8</td>
<td><strong>Path number received at Layer 6 from Layer 7 in a PDU.</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Line Setup configured for emulate mode only.</strong></td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>up_p_il_buff</td>
<td></td>
<td><strong>Interlayer-buffer number (an IAPX-286 segment number) received at Layer 6 from Layer 7 in a PDU. This segment number can be converted to a pointer by shifting it left 16 bits.</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Line Setup configured for emulate mode only.</strong></td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>up_p_sdu</td>
<td></td>
<td><strong>Offset to the start (header node) of the service data unit in an Interlayer-message buffer.</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Received at Layer 6 from Layer 7.</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Same as data_start_offset in a PDU.</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Line Setup configured for emulate mode only.</strong></td>
</tr>
<tr>
<td>extern unsigned long</td>
<td>l6_tick_count</td>
<td></td>
<td><strong>32-bit l1_tick_count stored in header of most recent IL buffer passed up to Layer 6.</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Preserves at each layer the original time when the end of the data (BCC) was clocked into the buffer.</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Line Setup configured for emulate or monitor mode.</strong></td>
</tr>
</tbody>
</table>
### Table 63-8
Layer 7 OSI Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern event</td>
<td>lo_p_prmtv</td>
<td></td>
<td>True when an OSI primitive is received at Layer 7 from Layer 6. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern event</td>
<td>m_lo_p_prmtv</td>
<td></td>
<td>True when an OSI primitive is received at Layer 7 from Layer 6. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>lo_p_pdu_seg</td>
<td></td>
<td>OSI primitive data unit (PDU) IAPX-286 segment number received at Layer 7 from Layer 6. This segment number can be converted to a pointer by shifting it left 16 bits. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>m_lo_p_pdu_seg</td>
<td></td>
<td>OSI primitive data unit (PDU) IAPX-286 segment number received at Layer 7 from Layer 6. This segment number can be converted to a pointer by shifting it left 16 bits. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>lo_p_prmtv_code</td>
<td>c1/193</td>
<td>p conn ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c3/195</td>
<td>p conn conf</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c5/197</td>
<td>p data ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c9/201</td>
<td>p expd data ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cd/205</td>
<td>p release ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cf/207</td>
<td>p release conf</td>
</tr>
<tr>
<td></td>
<td></td>
<td>d1/209</td>
<td>p debug ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>d3/211</td>
<td>p unit data ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>d5/213</td>
<td>p error report ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>d9/217</td>
<td>p mgf facility ind</td>
</tr>
<tr>
<td></td>
<td>m_lo_p_prmtv_code</td>
<td>c4/196</td>
<td>td p data ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c5/197</td>
<td>rd p data ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c8/200</td>
<td>td p expd data ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c9/201</td>
<td>rd p expd data ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>d4/212</td>
<td>td p unit data ind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>d5/213</td>
<td>rd p unit data ind</td>
</tr>
<tr>
<td></td>
<td>lo_p_prmtv_path</td>
<td>0-8</td>
<td>Path number received at Layer 7 in a PDU from Layer 6. Line Setup configured for emulate mode only.</td>
</tr>
</tbody>
</table>
Table 63-8 (continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_lo_p_prmtv_path</td>
<td>0-8</td>
<td>Path number received at Layer 7 in a PDU from Layer 6. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>lo_p_ll_buff</td>
<td></td>
<td>Interlayer-buffer number (an IAPX-286 segment number) received at Layer 7 in a PDU from Layer 6. This segment number can be converted to a pointer by shifting it left 16 bits. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>m_lo_p_ll_buff</td>
<td></td>
<td>Interlayer-buffer number (an IAPX-286 segment number) received at Layer 7 in a PDU from Layer 6. This segment number can be converted to a pointer by shifting it left 16 bits. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>lo_p_sdu</td>
<td></td>
<td>In OSI primitive received at Layer 7 from Layer 6, the offset to where the service data unit begins. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>m_lo_p_sdu_offset</td>
<td></td>
<td>In OSI primitive received at Layer 7 from Layer 6, the offset to where the service data unit begins. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>m_lo_p_sdu_size</td>
<td></td>
<td>Size of the service data unit in an interlayer-message buffer. Received at Layer 7 from Layer 6. Same as data_length in a PDU. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern unsigned long</td>
<td>l7_tick_count</td>
<td></td>
<td>32-bit l_t_tick_count stored in header of most recent IL buffer passed up to Layer 7. Preserves at each layer the original time when the end of the data (BCC) was clocked into the buffer. Line Setup configured for emulate or monitor mode.</td>
</tr>
</tbody>
</table>

63.3 Routines

OSI routines available at each layer make sending primitives to a layer above or below possible (see Figure 63-3). The routine name and its arguments provide the same information as the softkey selections on the Protocol Spreadsheet. (In the early phases of compiling the program, the C translator uses the routines to convert the spreadsheet softkey-token primitives into C.) All routines are protocol-independent.

63–30
(A) Layer-Independent OSI Routines

The following interlayer buffer service routines operate at any layer, regardless of protocol (or in the absence of a protocol package).

_get_il_msg_buff

Synopsis
extern void _get_il_msg_buff(buffer_number_ptr, maintain_bit_ptr);
unsigned short * buffer_number_ptr;
unsigned short * maintain_bit_ptr;

Description

The _get_il_msg_buff routine gets a free interlayer message buffer from the pool and returns the buffer number to the caller for use in subsequent calls to other interlayer buffer services. It also returns a maintain bit for use in the freeing operation.

Inputs

The first parameter is a pointer to the location where the buffer number is to be stored. The buffer number that is returned is actually an iAPX-286 segment number which can be converted to a pointer by shifting it 16 bits to the left. If there is no free buffer available, the routine will wait for one to become available.

The second parameter is a pointer to the location where the maintain bit will be stored. Since it must be used in the freeing operation, the maintain bit value should not be modified. The zero bit in this variable indicates your maintain bit.

Example

The variables in which the returned buffer number and maintain bit will be stored must be declared. When calling the routine, reference the addresses of these variables.

{
    unsigned short il_buffer_number;
    unsigned short relay_baton;
}

LAYER: 4
STATE: get_a_buffer
    CONDITIONS: KEYBOARD
    ACTIONS:
    {
        _get_il_msg_buff(&il_buffer_number, &relay_baton);
    }

The routine will get a buffer number and store it in variable il_buffer_number. It will also return a maintain bit and store it in variable relay_baton.
_start_il_buff_list

Synopsis

extern void _start_il_buff_list(il_buffer_number, start_offset_ptr);
unsigned short il_buffer_number;
unsigned short * start_offset_ptr;

Description

The _start_il_buff_list routine starts a linked list of text inside an interlayer message buffer. The list is made up of a header node and text nodes. The header node contains offsets to the first and last text nodes. Each text node contains a pointer to the actual text, the length of the text, and the offset to the next text node. This routine actually creates the header node inside the interlayer message buffer and initializes the first and last text node offsets to zero, indicating an empty list. It will return the offset to the list header node for use in subsequent list service calls.

Inputs

The first parameter is the interlayer message buffer number that will contain the list.

The second parameter is a pointer to the location where the offset to the list header will be stored. The returned offset will be zero if there is insufficient room in the buffer for the header node and one text node. Otherwise, it is the offset from the beginning of the message buffer to the start of the header node.

To convert the offset into a pointer, shift the buffer number 16 bits to the left and add the offset:

(void *)(((long)il_buffer_number << 16) + data_start_offset);

Example

Get a buffer and start a linked list. The variable in which the returned offset will be stored must be declared. When calling the routine, reference the address of this variable.

```c
{
    unsigned short il_buffer_number;
    unsigned short relay_baton;
    unsigned short data_start_offset;
}
```
The routine will get the offset to the header node and store it in variable data_start_offset.

\_dup\_il\_buff\_list\_start

Synopsis

extern unsigned short \_dup\_il\_buff\_list\_start(il_buffer\_number, start\_offset, 
new\_start\_offset\_ptr);
unsigned short il_buffer\_number;
unsigned short start\_offset;
unsigned short \* new\_start\_offset\_ptr;

Description

This routine duplicates the header node of a pointer list. In order for a layer to retain the ability to resend a buffer—that is, to reference again the same list header with the same first-node offset—it must keep its own linked list safe from data inserted at a layer below. The \_dup\_il\_buff\_list\_start routine allows the lower layer to start its own list.

If the lower layer will insert data into the buffer, it need duplicate only the list header ("list_start"), not the entire list. If the layer will append data to the end of the buffer, it must duplicate the complete linked list via the \_dup\_il\_buff\_list routine.

Inputs

The first parameter is the interlayer message buffer number in which the header node will be duplicated.

The second parameter is the offset to the header node to be duplicated.

The third parameter is a pointer to the location where the offset to the new header node will be stored.

Returns

This routine returns zero if there is not enough room in the buffer for the duplicated header node and at least one list node.
Example

Duplicate the header node of a buffer passed down from Layer 3.

```c
extern volatile unsigned short up_dl_il_buff;
extern volatile unsigned short up_dl_sdu;
unsigned short l2_data_start_offset;
}
```

```c
LAYER: 3
STATE: message
CONDITIONS: KEYBOARD
ACTIONS: DL_DATA REQ
```

```c
LAYER: 2
STATE: duplicate_header
CONDITIONS: DL_DATA REQ
ACTIONS:
{
    _dup_il_buff_list_start(up_dl_il_buff, up_dl_sdu, &l2_data_start_offset);
/* See _insert_il_buff_list_cnt routine on how information is inserted in the buffer. */
}
```

```c
_dup_il_buff_list
```

Synopsis

```c
extern unsigned short _dup_il_buff_list(il_buffer_number, start_offset, new_start_offset_ptr);
unsigned short il_buffer_number;
unsigned short start_offset;
unsigned short * new_start_offset_ptr;
```

Description

This routine duplicates an entire pointer list. In order for a layer to be able to retain the ability to resend a buffer—that is, to reference again the same list header with the same first- and last-node offsets—it must keep its own linked list safe from data inserted and appended at a layer below. The `_dup_il_buff_list` routine allows the lower layer to have its own list.

If the lower layer will append data to the buffer, it should duplicate the entire linked list. If the layer will only insert data into the buffer, it need only duplicate the header node via the `_dup_il_buff_list_start` routine.

Inputs

The first parameter is the interlayer message buffer number in which the list will be duplicated.

The second parameter is the offset to the header node of the list to be duplicated.
The third parameter is a pointer to the location where the offset to the header node for the new list will be stored.

**Returns**

This routine returns zero if the duplication is successful. If there is not enough room in the buffer to duplicate the list, one is returned.

**Example**

Duplicate the entire pointer list of a buffer passed down from Layer 3.

```c
{ extern volatile unsigned short up_dl_il_buff;
  extern volatile unsigned short up_dl_sdu;
  unsigned short l2_data_start_offset;
}

LAYER: 3
STATE: message
  CONDITIONS: KEYBOARD *
  ACTIONS: DL_DATA_REQ "0 5 66 (FOX)"

LAYER: 2
STATE: duplicate_list
  CONDITIONS: DL_DATA_REQ
  ACTIONS:
  {
    _dup_il_list(up_dl_il_buff, up_dl_sdu, &l2_data_start_offset);
  /* See _append_il_list_cnt routine on how information is appended to the buffer. */
  }

_open_space_in_il_buff

**Synopsis**

`extern void _open_space_in_il_buff(il_buffer_number, length, space_offset_ptr);`

`unsigned short il_buffer_number;`

`unsigned short length;`

`unsigned short * space_offset_ptr;`

**Description**

The _open_space_in_il_buff routine opens up the requested amount of space in the specified interlayer message buffer. It returns an offset from the beginning of the buffer to the start of the open space.

**Inputs**

The first parameter is the interlayer message buffer number in which space is to be made.
The second parameter is the amount of space (number of bytes) requested.

The third parameter is a pointer to the location where the returned offset will be stored. The returned offset will be zero if there is insufficient room in the buffer.

To convert the offset into a pointer, shift the buffer number 16 bits to the left and add the offset:

\[(\text{void } *)(\text{(long)} \text{il_buffer_number} \ll 16) + \text{available_space_offset};\]

**Example**

Always open space in the buffer if you are going to copy data (usually header information) into the buffer. If you are not going to copy data into the buffer, but reference its location in memory outside the buffer (usually user data), you do not need to open space.

The variable in which the returned offset will be stored must be declared. When calling the routine, reference the address of this variable. The length may be entered as a numeric value, in which case a length variable need not be declared.

For example, a buffer at Layer 3 will have three X.25-header bytes inserted. The call for space to hold the header would look like this:

```c
{  
    unsigned short il_buffer_number;
    unsigned short relay_baton;
    unsigned short data_start_offset;
    unsigned short available_space_offset;
}

STATE: get_space
CONDITIONS: KEYBOARD " "
ACTIONS:
{
    _get_il_msg_buff(&il_buffer_number, &relay_baton);
    _start_il_buff_list(il_buffer_number, &data_start_offset);
    _open_space_in_il_buff(il_buffer_number, 3, &available_space_offset);
    /* See _insert_il_buff_list_cnt routine on how information is inserted in the buffer. */
}
```

The routine will get the offset to the next available space in the buffer and store it in variable `available_space_offset`.

Once space has been opened, the buffer-number and available-space variables can be converted into an open-space pointer. With this pointer, data can be copied into the space. The pointer can then be referenced in an `insert_il_buff_list_cnt` routine, so that the opened space becomes threaded onto the linked list in the IL buffer. See the programming example under `insert_il_buff_list_cnt`.
_free_IL_msg_buff

Synopsis

extern void _free_il_msg_buff(unsigned short il_buffer_number, unsigned short relay_baton);
unsigned short il_buffer_number;
unsigned short relay_baton;

Description

The _free_IL_msg_buff routine returns an interlayer message buffer to the pool of free buffers. Before actually returning the buffer to the pool, this routine verifies that all maintain bits have been reset, assuring that all users have freed this buffer.

Inputs

The first parameter is the interlayer--buffer number to be freed.

The second parameter is the maintain bit associated with the buffer user to be freed.

Example

See _set_maint_buff_bit routine.

_set_maint_buff_bit

Synopsis

extern void _set_maint_buff_bit(unsigned short il_buffer_number, unsigned short new_bit); unsigned short il_buffer_number;
unsigned short new_bit;

Description

The _set_maint_buff_bit routine sets a new maintain bit for a given interlayer message buffer. It returns that bit to the caller to be used in the freeing operation.

The maintain bit allocated in the _free_IL_msg_buff routine should be considered valid only for the layer at which it was obtained. Once you pass a buffer, the maintain bit will hold the buffer at the next layer only until action on it has been processed. (In Spreadsheet terms, the buffer will be held until the ACTIONS block has been processed in response to the first CONDITIONS block identifying the buffer. In any other CONDITIONS block referring to the buffer, the buffer will not be found unless an additional maintain bit was set.) The maintain bit
eventually will be freed automatically whether or not any action is taken on it at the next layer. To hold a buffer at a particular layer, or to continue passing the buffer (in either direction), a new maintain bit must be set. The same maintain bit cannot be used continuously, since it will be freed after the first process on it (an ACTION to send, for example).

If you wish to keep a buffer available for your use while also sending it to another layer, set two maintain bits. One will be used to pass the buffer; the other will "maintain" the buffer for other processes. The latter will have to be freed via the _free_il_msg_buff routine.

Inputs
The first parameter is the interlayer-buffer number in which the new bit will be set.

The second parameter is a pointer to the location where the returned maintain bit will be stored. There are sixteen maintain bits reserved for each interlayer buffer. Each bit is identified by a two-byte variable with a single zero. The first maintain bit allocated is the least significant, so the value returned is hexadecimal FFFE (binary 11111111 11111110). The last maintain bit allocated is 7FFF (01111111 11111111). If all the maintain bits are already in use, FFFF will be returned.

The maintain bit value should not be modified. It must be used in the freeing operation to make sure the buffer is returned to the free buffer pool.

Example
The variable in which the returned maintain bit will be stored must be declared. When calling the routine, reference the address of this variable. For example, you receive a buffer at Layer 2 from Layer 3 (up_dl_il_buff) and insert information into it. Before passing the buffer to Layer 1, set two maintain bits. The one stored in variable maintain_bit will hold the buffer for the purpose of repeated resends of the frame, if necessary, and will have to be freed via the _free_il_msg_buff routine. When you pass the buffer down, use the bit in variable l2_relay_baton. When you resend the frame, set a new resend_baton bit and pass that down, still holding maintain_bit in reserve for subsequent resends.

```
{ 
    unsigned short l2_relay_baton;
    unsigned short resend_baton;
    unsigned short maintain_bit;
    extern volatile unsigned short up_dl_il_buff;
    extern volatile unsigned short up_dl_sdu;
    unsigned short l2_data_start_offset;
    unsigned short available_space_offset;
    static unsigned char l2_data[2] = {0x01, 0x00};
    int i;
    unsigned char * ptr_l2;
```
```c
#define make_ptr(number, offset) (((long)((number) << 16) + offset))

LAYER: 3
STATE: send_fox_message
CONDITIONS: KEYBOARD * *
ACTIONS: DL_DATA REQ "%X%8(FOX)"

LAYER: 2
STATE: send_a_buffer
CONDITIONS: DL_DATA REQ
ACTIONS:

LAYER: 1
STATE: resend_buffer
CONDITIONS: RECEIVE STRING "%8(XXXX1001)"
ACTIONS:

/* See Section 59, Monitor/Transmit Line Data, for an explanation of the il_transmit routine. */

CONDITIONS: RECEIVE STRING "%8(XXXX0001)"
ACTIONS:

/* See _free_il_msg_buff for an explanation of this routine. */

_insert_il_buff_list_cnt

Synopsis

extern unsigned short _insert_il_buff_list_cnt(il_buffer_number, data_start_offset, text_ptr, text_length);
unsigned short il_buffer_number;
unsigned short data_start_offset;
unsigned char *text_ptr;
unsigned short text_length;
```
Description

The _insert_il_buff_list_cnt routine inserts a text node at the beginning of a linked list of text inside of an interlayer message buffer. It will set the text pointer and byte-count in the text node to the values specified.

Inputs

The first parameter is the interlayer-buffer number in which the linked list will be inserted.

The second parameter is the offset to the header node for the linked list, from the beginning of the buffer.

The third parameter is a pointer to a text.

The fourth parameter is the length of the text.

Returns

If the insert is successful, a value of 0 is returned; if it is not successful, a value of 1 is returned. If you want to check the returned value, do so at the time the routine is called, as in the following example at Layers 2 and 3.

Example

If text is to be copied into the buffer, a pointer to the text must be declared. If not, when calling the _insert_il_buff_list_cnt routine, reference the address of the text. The length of the text may be entered as an integer, in which case a length variable need not be declared.

Always open space in the buffer if you are going to copy data (usually header information) into the buffer. If you are not going to copy data into the buffer, but reference its location in memory outside the buffer (usually user data), you do not need to open space.

In the following spreadsheet example, an interlayer-buffer number is obtained at Layer 5, a header node is created in the buffer, and the address of a fox message text (located in memory outside of the buffer) is inserted into a text node in the buffer.

```c
{ 
    unsigned short ii_buffer_number;
    unsigned short relay_baton;
    unsigned short i4_relay_baton
    unsigned short i3_relay_baton;
    unsigned short i2_relay_baton;
    unsigned short data_start_offset;
    unsigned short i2_data_start_offset;
};
```
unsigned short available_space_offset;
static unsigned char data[1] = "FOX";
static unsigned char i3_data[3] = {0x10, 0x04, 0x00};
static unsigned char i2_data[2] = {0x01, 0x00};
int i;
int length;
extern volatile unsigned short up_t_il_buff;
extern volatile unsigned short up_n_il_buff;
extern volatile unsigned short up_dl_il_buff;
extern volatile unsigned short up_n_sdu;
extern volatile unsigned short up_dl_sdu;
extern volatile unsigned short up_t_sdu;
unsigned char *ptr_i3, *ptr_i2;

/* Whenever make_ptr is encountered, the first parameter will be shifted 16 bits to the left. This is done to include the first parameter with the first parameter. */

#define make_ptr(number, offset) (*((void *)((long)number << 16) + offset))

LAYER: 5
STATE: begin_message
CONDITIONS: KEYBOARD **
ACTIONS:
{
  _get_il_msg_buff(&iil_buffer_number, &relay_baton);
  _start_il_buff_list(iil_buffer_number, &data_start_offset);
}

/* Do not include the terminating null character in the length determination of a string. */
length = sizeof(data) - 1;

/* The address of data outside of the buffer is given for insertion. The data itself is not copied into the buffer. The buffer is then passed down to Layer 4 (see send_ptrmsg below for an explanation of this routine). */
_insert_il_buff_list_cnt(iil_buffer_number, data_start_offset, &data[0], length);
send_ptrmsg_below(iil_buffer_number, relay_baton, data_start_offset, 0, Ox84, 0);

At Layer 4 a new maint buffer is set to use in passing the buffer to Layer 3. Since no data is inserted, the same data_start_offset is used (in the form of the variable up_t_sdu). The buffer is then passed down to Layer 3 (see send_ptrmsg_below for an explanation of this routine).

LAYER: 4
STATE: pass
CONDITIONS: T_DATA_REQ
ACTIONS:
{
  _set_maint_buff_bit(up_t_il_buff, &i4_relay_baton);
  send_ptrmsg_below(up_t_il_buff, i4_relay_baton, up_t_sdu, 0, Ox64, 0);
}

At Layer 3, space is opened for an X.25 packet header. A pointer to the opened space is created and the data is inserted into the linked list passed down from Layer 4.
LAYER: 3
STATE: insert_and_send
CONDITIONS: N_DATA_REQ
ACTIONS:
{
_open_space_in_il_buff(up_n_il_buff, 3, &available_space_offset);
ptr_l3 = make_ptr(up_n_il_buff, available_space_offset);
for(i = 0; i < 3; i++)
{
    *ptr_l3 = l3_data[i];
    ptr_l3++;
}
/* The location of the data in the buffer is referenced in the insert routine, so the pointer must be moved back to the beginning of the opened space. The offset to the Layer 3 header node is given in the insert routine. If the insertion is not successful, an alarm will sound and a message will be displayed on the prompt line of the screen. */
ptr_l3 -= 3;
if(_insert_il_buff_list_cnt(up_n_il_buff, up_n_sdu, ptr_l3, 3) != 0)
{
sound_alarm();
display_prompt("Insert failed at Layer 3.");
}
/* A new maintain bit is set for passing the buffer. The buffer is then passed down to Layer 2 (see send_dl_prmtv_below for an explanation of this routine). */
_set_maint_buff_bit(up_n_il_buff, &13_relay_baton);
send_dl_prmtv_below(up_n_il_buff, 13_relay_baton, up_n_sdu, 0, 0x44, 0);
}

At Layer 2, a new linked list is started. The Layer 2 header could be inserted into the linked list passed down from Layer 3; but if Layer 3 wants to retain the ability to resend a buffer—that is, to reference again the same list header with the same first-node offset—it must keep its own linked list safe from data inserted at Layer 2.

LAYER: 2
STATE: insert_more
CONDITIONS: DL_DATA_REQ
ACTIONS:
{
/* The _dup_il_buff_list_start routine allows Layer 2 to start its own list. Part of this routine copies the Layer 3 header into the Layer 2 header node. */
_dup_il_buff_list_start(up_dl_il_buff, up_dl_sdu, &i2_data_start_offset);
/* Space is opened in the buffer. A pointer to this location is created and the data is copied into the buffer. */
_open_space_in_il_buff(up_dl_il_buff, 2, &available_space_offset);
ptr_l2 = make_ptr(up_dl_il_buff, available_space_offset);
for(i = 0; i < 2; i++)
{
    *ptr_l2 = l2_data[i];
    ptr_l2++;
}
/* The location of the data in the buffer is referenced in the insert routine, so the pointer must be moved back to the beginning of the opened space. The offset to the Layer 2 header node is given in the insert routine. If the insertion is not successful, an alarm will sound and a message will be displayed on the prompt line of the screen. */
ptr_l2 -=2;
if(_insert_il_buff_list_cnt(up_dl_il_buff, 12_data_start_offset, ptr_l2, 2) != 0)
{
    sound_alarm();
    pos_cursor(0, 30);
    displays("Insert failed at Layer 2.");
}
/* A new maintain bit is set for passing the buffer. The buffer is then passed down to Layer 1
(see send_ph_prmtv_below for an explanation of this routine). */
_set_maintBuff_bit(up_dl_il_buff, &12_relay_baton);
send_ph_prmtv_below(up_dl_il_buff, 12_relay_baton, 12_data_start_offset, 0, OxU, 0);
}

The following text will be sent out onto the line and displayed as line data:
%¨%¨%¨THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG 0123456789

_APPEND_IL_BUFF_LIST_CNT

Synopsis

extern unsigned short _append_il_buff_list_cnt(il_buffer_number, data_start_offset, text_ptr,
        text_length);
unsigned short il_buffer_number;
unsigned short data_start_offset;
unsigned char * text_ptr;
unsigned short text_length;

Description

The _append_il_buff_list_cnt routine appends a text node at the end of a linked
list of text inside of an interlayer message buffer. It will set the text pointer and
count in the text node to the information provided.

Inputs

See _insert_il_buff_list_cnt routine.

Returns

See _insert_il_buff_list_cnt routine.

Example

Two modifications to the program shown for the _insert_il_buff_list_cnt routine
are all that is required to make the program work for appending data. The
changes primarily involve Layer 2 in the example, so we will replicate only that
portion of the program below. Substitute _append_il_buff_list_cnt for every
occurrence _insert_il_buff_list_cnt. When data is to be appended in a buffer,
you should duplicate the entire linked list received from the layer above, not just
the header node. So also substitute _dup_il_buff_list for _dup_il_buff_list_start.
LAYER: 2
STATE: insert_more
CONDITIONS: DL_DATA_REQ
ACTIONS:
{
_dup_il_buff_list(up_dl_il_buff, up_dl_sdu, &12_data_start_offset);
_open_space_in_il_buff(up_dl_il_buff, 2, &available_space_offset);
ptr_l2 = make_ptr(up_dl_il_buff, available_space_offset);
for(i = 0; i < 2; i++)
{
  *ptr_l2 = 12_data[i];
  ptr_l2++;
}
ptr_l2 -=2;
if(_append_il_buff_list_cnt(up_dl_il_buff, 12_data_start_offset, ptr_l2, 2) != 0)
  {
    sound_alarm();
    pos_cursor(0,30);
    displays("Insert failed at Layer 2.");
  }
_set maint_buff_bit(up_dl_il_buff, &12 Relay_baton);
send phosphate_above(up_dl_il_buff, 12 Relay_baton, 12_data_start_offset, 0, 0x24, 0);
}

The following text will be sent out onto the line and displayed as line data:

THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG 0123456789°1.

(B) Layer 1 OSI Routines

OSI data primitives are handled automatically between Layers 1 and 2. In the "up" direction, line data is placed in an IL buffer and the associated data primitive is given automatically to Layer 2. In the "down" direction, data primitives are received at Layer 1 and put out automatically onto the line.

In the absence of line data, if you want to originate a buffer at Layer 1 and send it upward, use the following routine. In primitives being sent down the layers, Layer 1 will automatically send the primitive out onto the line.

send phosphate_to_above

Synopsis

extern void send phosphate_to_above(1l_buffer_number, relay_baton, data_start_offset, size, code, path);
unsigned short il_buffer_number;
unsigned short relay_baton;
unsigned short data_start_offset;
unsigned short size;
unsigned char code;
unsigned char path;

63-44
Description

The send_ph_to_above emulate routine passes a specified interlayer message buffer from Layer 1 to Layer 2 in an OSI primitive. Received line data is placed in an IL buffer and passed automatically to Layer 2. If you wish to get a buffer “manually” at Layer 1 and then pass it up, use this routine.

Inputs

The first parameter is the interlayer buffer number returned by the _get_il_msg_buff routine.

The second parameter is the returned maintain bit from the _get_il_msg_buff routine. As soon as Layer 2 processing on the buffer is completed, the bit is automatically freed.

The third parameter is the returned offset (from the call to _start_il_buff_list) to the Layer 1 service data unit in a buffer.

The fourth parameter is the length of the data in the buffer.

The fifth parameter is the code specifying the type of primitive in which the buffer will be sent. Refer to variable lo_ph_prmtv_code in Table 63-3 for the appropriate primitive code.

The sixth parameter is the path number along which the buffer will be sent.

Example

Get a buffer at Layer 1. Assuming X.25 protocol, insert data into the buffer and pass it up to Layer 2.

```c
{ unsigned short il_buffer_number;
  unsigned short relay_baton;
  unsigned short data_start_offset;
  unsigned short available_space_offset;
  int length;
  int i;
  static unsigned char data[] = {0x01, 0x00, 0x10, 0x04, 0x00, 0x02, 0x01, 0x01};
  unsigned char *ptr;
}

LAYER: 1
STATE: get_buffer
CONDITIONS: KEYBOARD * *
ACTIONS:
{ _get_il_msg_buff(&il_buffer_number, &relay_baton);
  _start_il_buff_list(il_buffer_number, &data_start_offset);
  length = sizeof(data);
  _open_space_in_il_buff(il_buffer_number, length, &available_space_offset);
  ptr = (void *)(((long)il_buffer_number << 16) + available_space_offset);
}`
for(i = 0; i < length; i++)
{
    *ptr = data[i];
    ptr++;
}
ptr-=length;
_insert_il_buff_list_cnt(il_buffer_number, data_start_offset, ptr, length);
send_ph_to_above(il_buffer_number, relay_baton, data_start_offset, length, 0x25, 0);
}

(C) Layer 2 OSI Routines

The following routines pass OSI primitives from Layer 2 to either Layer 3 or Layer 1.

**send_dl_prmtv_above**

**Synopsis**

```c
extern void send_dl_prmtv_above(il_buffer_number, i2_relay_baton, i2_data_start_offset, size, i2_code, path);
```

**Inputs**

The first parameter is the interlayer buffer number to be sent. For a buffer which has been received at Layer 2 from Layer 1, the variable lo_ph_il_buff may be used to identify the buffer number.

The second parameter is the returned maintain bit from a call to _set_maint_buff_bit. It is used only to pass a received buffer from Layer 2 to Layer 3. As soon as Layer 3 processing on the buffer is completed, the bit is automatically freed.

The third parameter is the offset to the Layer 2 service data unit in a received buffer. The variable lo_ph_sdu contains the offset to the service data unit when the buffer reached Layer 2. The offset must be incremented by the length of the Layer 2 header.
NOTE: In general, do not modify extern variables, such as lo_ph_sdu, which may be updated by other processes. Name another variable, assign it the same value, and then increment that variable. Or, after lo_ph_sdu has been named in the argument of the send routine, add the length of the Layer 2 header, as in the example below.

The fourth parameter is the length of the data in the buffer. Use the length indicated in the pdu structure—pdu.data_length. Then subtract the length of the Layer 2 header.

The fifth parameter is the code specifying the type of primitive in which the buffer will be sent. Refer to variable lo_dl_prmtv_code in Table 63-4 for the appropriate primitive code.

The sixth parameter is the path number along which the buffer will be sent. For a buffer which has been received at Layer 2 from Layer 1, the variable lo_ph_prmtv_path may be used to specify the path number.

Example

A buffer is received at Layer 2 from Layer 1. Assuming X.25 protocol, the data specific to Layer 2 (the frame header) begins at the SDU offset (lo_ph_sdu) and consists of two bytes. Before the buffer is passed up to Layer 3, the offset to the SDU and the size of the SDU will be adjusted by two bytes and a new maintain bit will be set.

```c
{struct pdu
{
    unsigned char primitive_code;
    unsigned char path;
    unsigned long parameter;
    unsigned short relay_baton;
    unsigned short il_buffer_number;
    unsigned char buffer_contents;
    unsigned short data_start_offset;
    unsigned short data_length;
};
struct pdu * pdu_ptr;
extern volatile unsigned short lo_ph_pdu_parm;
extern volatile const unsigned char lo_ph_prmtv_path;
extern volatile unsigned short lo_ph_il_buff;
extern volatile unsigned short lo_ph_sdu;
unsigned short l2_relay_baton;
}`
```
LAYER: 2
STATE: send_buffer_up
CONDITIONS: PH_DATA_IND
ACTIONS:
{
    pdu_ptr = (void *)&lo_ph_pdu_seg << 16);
    _set_maint_buf_bit(lo_ph_pdu, &l2_relay_baton);
    send_dl_prmtv_above(lo_ph_il_buff, l2_relay_baton, lo_ph_sdu + 2,
                        pdu_ptr->data_length - 2, 0x45, lo_ph_prmtv_path);
}

send_m_dl_prmtv_above

Synopsis

extern void send_m_dl_prmtv_above(ll_buffer_number, l2_relay_baton, l2_data_start_offset,
                                 size, l2_code, path);
unsigned short il_buffer_number;
unsigned short l2_relay_baton;
unsigned short l2_data_start_offset;
unsigned short size;
unsigned char l2_code;
unsigned char path;

Description

The send_m_dl_prmtv_above monitor routine passes a specified interlayer message buffer from Layer 2 to Layer 3 in an OSI monitor primitive.

Inputs

See send_dl_prmtv_above. Use the monitor variables m_lo_ph_il_buff, m_lo_ph_sdu_offset, and m_lo_ph_sdu_size as input. Refer to variable m_lo_dl_prmtv_code in Table 63-4 for the appropriate primitive code.

Example

Make the appropriate variable declarations. For a condition monitoring RD data primitives, the Layer 2 programming block should look like this:

LAYER: 2
STATE: send_buffer_up
CONDITIONS: PH_RD_DATA_IND
ACTIONS:
{
    _set_maint_buf_bit(m_lo_ph_il_buff, &l2_relay_baton);
    send_m_dl_prmtv_above(m_lo_ph_il_buff, l2_relay_baton, m_lo_ph_sdu_offset + 2,
                          m_lo_ph_sdu_size - 2, 0x45, m_lo_ph_prmtv_path);
}
send_ph_prmtv_below

Synopsis

extern void send_ph_prmtv_below(il_buffer_number, l2_relay_baton, l2_data_start_offset, size,
     l2_code, path);
unsigned short il_buffer_number;
unsigned short l2_relay_baton;
unsigned short l2_data_start_offset;
unsigned short size;
unsigned char l2_code;
unsigned char path;

Description

The send_ph_prmtv_below emulate routine passes a specified interlayer message buffer from Layer 2 to Layer 1 in an OSI primitive.

Inputs

The first parameter is the interlayer buffer number to be sent. For a buffer which has been received at Layer 2 from Layer 3, the variable up_dl_il_buff may be used to identify the buffer number. If the buffer originated at Layer 2, use the buffer-number variable named in the _get_il_msg_buff routine. (See _insert_il_buff_list_cnt routine example at Layer 5.)

The second parameter is the returned maintain bit from a call to _set_maint_buff_bit. It is used only to pass a received buffer from Layer 2 to Layer 1. As soon as Layer 1 processing on the buffer is completed, the bit is automatically freed. If the buffer originated at Layer 2, use the maintain bit variable named in the _get_il_msg_buff routine. (See _insert_il_buff_list_cnt routine example at Layer 5.)

The third parameter is the offset to the Layer 2 list header node in the buffer. For a buffer which has been received at Layer 2 from Layer 3, the variable up_dl_sdu may be used to indicate the offset.

The fourth parameter is the size of the data in the buffer. It will always be set to zero since the data length is unknown in a primitive being passed down the layers.

The fifth parameter is the code specifying the type of primitive in which the buffer will be sent. Refer to variable ph_prmtv_type in Table 63-2 for the appropriate primitive code.

The sixth parameter is the path number along which the buffer will be sent. For a buffer which has been received at Layer 2 from Layer 3, the variable up_dl_prmtv_path may be used to specify the path number.
Example

A buffer is received at Layer 2 from Layer 3. No text will be inserted at Layer 2. (For information on inserting text, see _insert_il_buff_list_cnt routine.) The buffer will be passed to Layer 1, requiring a new maintain bit to be set. If values are entered for the code and path, variables for code and path need not be declared.

{ extern volatile unsigned short up_dl_il_buff;
  extern volatile unsigned short up_dl_sdu;
  unsigned short l2_relay_baton;
}

LAYER: 2
STATE: pass_buffer_down
CONDITIONS: DL_DATA_REQ
ACTIONS:
{
  _set_maint_buff_bit(up_dl_il_buff, &l2_relay_baton);
  send_ph_prmtv_below(up_dl_il_buff, l2_relay_baton, up_dl_sdu, 0, 0x24, 0);
}

(D) Layer 3 OSI Routines

The following routines pass OSI primitives from Layer 3 to either Layer 4 or Layer 2.

send_n_prmtv_above

Synopsis

extern void send_n_prmtv_above(il_buffer_number, l3_relay_baton, l3_data_start_offset, size,
  l3_code, path);
unsigned short il_buffer_number;
unsigned short l3_relay_baton;
unsigned short l3_data_start_offset;
unsigned short size;
unsigned char l3_code;
unsigned char path;

Description

The send_n_prmtv_above emulate routine passes a specified interlayer message buffer from Layer 3 to Layer 4 in an OSI primitive.

Inputs

The first parameter is the interlayer buffer number to be sent. For a buffer which has been received at Layer 3 from Layer 2, the variable lo_dl_il_buff may be used to identify the buffer number.
The second parameter is the returned maintain bit from a call to 
_set_main_maint_bit. It is used only to pass a received buffer from Layer 3 to 
Layer 4. As soon as Layer 4 processing on the buffer is completed, the bit is 
automatically freed.

The third parameter is the offset to the Layer 3 service data unit in a received 
buffer. The variable lo_dl_sdu contains the offset to the service data unit when 
the buffer reached Layer 3. The offset must be incremented by the length of 
the Layer 3 header.

NOTE: In general, do not modify extern variables, such as 
lo_dl_sdu, which may be updated by other processes. Name 
another variable, assign it the same value, and then increment 
that variable. Or, after lo_dl_sdu has been named in the 
argument of the send routine, add the length of the Layer 3 
header, as in the example below.

The fourth parameter is the length of the data in the buffer. Use the length 
indicated in the pdu structure—pdu.data_length. Then subtract the length of the 
Layer 3 header.

The fifth parameter is the code specifying the type of primitive in which the 
buffer will be sent. Refer to variable lo_n_prmtv_code in Table 63-5 for the 
appropriate primitive code.

The sixth parameter is the path number along which the buffer will be sent. For 
a buffer which has been received at Layer 3 from Layer 2, the variable 
lo_dl_prmtv_path may be used to specify the path number.

Example

A buffer is received at Layer 3 from Layer 2. Assuming X.25 protocol, the 
header consists of three bytes. The offset to and size of the service data unit 
will be adjusted by three bytes, a new maintain bit will be set, and the buffer will 
be passed up to Layer 4.

{ 
  struct pdu 
  { 
    unsigned char primitive_code; 
    unsigned char path; 
    unsigned long parameter; 
    unsigned short relay_baton; 
    unsigned short il_buffer_number; 
    unsigned char buffer_contents; 
    unsigned short data_start_offset; 
    unsigned short data_length; 
  };

 63-51
struct pdu * pdru_ptr;
extern volatile unsigned short lo_di pdu_seg;
extern volatile const unsigned char lo_di prmvtv_path;
extern volatile unsigned short lo_di ilBuff;
extern volatile unsigned short lo_di sdu;
unsigned short 13_relay_baton;

LAYER: 3
STATE: send buffer up
CONDITIONS: DL_DATA IND
ACTIONS:
{
    pdu_ptr = (void *)(long)lo_di pdu seg << 16;
    _set maint_buff_bit(lo_di il_buff, &13_relay_baton);
    send n prmtv above(lo_di il_buff, 13_relay_baton, lo_di sdu + 3,
    pdu_ptr->data_length - 3, 0x65, lo_di prmvtv_path);
}

send m n prmtv_above

Synopsis

extern void send m n prmtv_above(int_buffer_number, 13_relay_baton, 13_data_start_offset,
size, 13_code, path);
unsigned short il_buffer_number;
unsigned short 13_relay_baton;
unsigned short 13_data_start_offset;
unsigned short size;
unsigned char 13_code;
unsigned char path;

Description

The send m n prmtv_above monitor routine passes a specified interlayer message buffer from Layer 3 to Layer 4 in an OSI monitor primitive.

Inputs

See send n prmtv_above. Use the monitor variables m_lo_di il_buff, m_lo_di sdu_offset, and m_lo_di sdu size as input. Refer to variable m_lo_n prmtv_code in Table 63-5 for the appropriate primitive code.

Example

Make the appropriate variable declarations. For a condition monitoring RD data primitives, the Layer 3 programming block should look like this:

LAYER: 3
STATE: send buffer up
CONDITIONS: DL_RD_DATA IND
ACTIONS:
{
    _set maint_buff_bit(m_lo_di il_buff, &13_relay_baton);
    send m n prmtv above(m_lo_di il_buff, 13_relay_baton, m_lo_di sdu_offset + 3,
    m_lo_di sdu size - 3, 0x65, m_lo_di prmvtv_path);
}
send_dl_prmtv_below

Synopsis

extern void send_dl_prmtv_below(il_buffer_number, i3 Relay_baton, i3 data_start_offset, size, i3_code, path);
unsigned short il_buffer_number;
unsigned short i3 Relay_baton;
unsigned short i3 data_start_offset;
unsigned short size;
unsigned char i3_code;
unsigned char path;

Description

The send_dl_prmtv_below emulate routine passes a specified interlayer message buffer from Layer 3 to Layer 2 in an OSI primitive.

Inputs

The first parameter is the interlayer buffer number to be sent. For a buffer which has been received at Layer 3 from Layer 4, the variable up_n_il_buff may be used to identify the buffer number. If the buffer originated at Layer 3, use the buffer-number variable named in the _get_il_msg_buff routine. (See _insert_il_buff_list_cnt routine example at Layer 5.)

The second parameter is the returned maintain bit from a call to _set_maint_buff_bit. It is used only to pass a received buffer from Layer 3 to Layer 2. As soon as Layer 2 processing on the buffer is completed, the bit is automatically freed. If the buffer originated at Layer 3, use the maintain bit variable named in the _get_il_msg_buff routine. (See _insert_il_buff_list_cnt routine example at Layer 5.)

The third parameter is the offset to the Layer 3 list header node in the buffer. For a buffer which has been received at Layer 3 from Layer 4, the variable up_n_sdu may be used to indicate the offset.

The fourth parameter is the size of the data in the buffer. It will always be set to zero since the data length is unknown in a primitive being passed down the layers.

The fifth parameter is the code specifying the type of primitive in which the buffer will be sent. Refer to variable up_dl_prmtv_code in Table 63-3 for the appropriate primitive code.

The sixth parameter is the path number along which the buffer will be sent. For a buffer which has been received at Layer 3 from Layer 4, the variable up_n_prmtv_path may be used to specify the path number.
Example

A buffer is received at Layer 3 from Layer 4. No text will be inserted at Layer 3. (For information on inserting text, see _insert_il_buff_list_cnt routine.) The buffer will be passed to Layer 2, requiring a new maintain bit to be set. If values are entered for the code and path, these variables need not be declared.

```c
extern volatile unsigned short up_n_il_buff;
extern volatile unsigned short up_n_sdu;
unsigned short l3 Relay_baton;
}
LAYER: 3
STATE: pass_buffer_down
CONDITIONS: N_DATA_REQ
ACTIONS:
{
  _set_main_buff_bit(up n il_buff, &l3 Relay_baton);
  send dl prmtv_below(up n il_buff, l3 Relay_baton, up n sdu, 0x44, 0);
}
```

(E) Layer 4 OSI Routines

The following routines pass OSI primitives from Layer 4 to either Layer 5 or Layer 3.

**send_t_prmtv_above**

**Synopsis**

```c
extern void send_t_prmtv_above(il_buffer_number, l4 Relay_baton, l4_data_start_offset, size,
  l4 code, path);
unsigned short il_buffer_number;
unsigned short l4 Relay_baton;
unsigned short l4_data_start_offset;
unsigned short size;
unsigned char l4 code;
unsigned char path;
```

**Description**

The *send_t_prmtv_above* emulate routine passes a specified interlayer message buffer from Layer 4 to Layer 5 in an OSI primitive.

**Inputs**

The first parameter is the interlayer buffer number to be sent. For a buffer which has been received at Layer 4 from Layer 3, the variable *lo n il_buff* may be used to identify the buffer number.
The second parameter is the returned maintain bit from a call to _set_maint_buff_bit. It is used only to pass a received buffer from Layer 4 to Layer 5. As soon as Layer 5 processing on the buffer is completed, the bit is automatically freed.

The third parameter is the offset to the Layer 4 service data unit in a received buffer. The variable lo_n_sdu contains the offset to the service data unit when the buffer reached Layer 4. The offset must be incremented by the length of the Layer 4 header, if any.

NOTE: In general, do not modify extern variables, such as lo_n_sdu, which may be updated by other processes. Name another variable, assign it the same value, and then increment that variable. Or, after lo_n_sdu has been named in the argument of the send routine, add the length of the Layer 4 header, if any.

The fourth parameter is the length of the data in the buffer. Use the length indicated in the pdu structure—pdu.data_length. Then subtract the length of the Layer 4 header, if any.

The fifth parameter is the code specifying the type of primitive in which the buffer will be sent. Refer to variable lo_t_prmtv_code in Table 63-6 for the appropriate primitive code.

The sixth parameter is the path number along which the buffer will be sent. For a buffer which has been received at Layer 4 from Layer 3, the variable lo_n_prmtv_path may be used to specify the path number.

Example
A buffer is received at Layer 4 from Layer 3. The offset to and size of the service data unit will be adjusted if needed, a new maintain bit will be set, and the buffer will be passed up to Layer 5.

```c
{
    struct pdu
    {
        unsigned char primitive_code;
        unsigned char path;
        unsigned long parameter;
        unsigned short relay_baton;
        unsigned short il_buffer_number;
        unsigned char buffer_contents;
        unsigned short data_start_offset;
        unsigned short data_length;
    };  
    struct pdu * pdu_ptr;
    extern volatile unsigned short lo_n_pdu_seg;
    extern volatile const unsigned char lo_n_prmtv_path;
    
    {
        
```
extern volatile unsigned short lo_n_il_buff;
extern volatile unsigned short lo_n_sdu;
unsigned short 14_relay_baton;
}

LAYER: 4
STATE: send_buffer_up
    CONDITIONS: N_DATA_IND
    ACTIONS:
    {
        pdu_ptr = (void *)((long)lo_n_pdu_seg << 16);
        _set_maint_buff_bit(lo_n_il_buff, &14_relay_baton);
        send_t_prmtv_above(lo_n_il_buff, 14_relay_baton, lo_n_sdu, pdu_ptr->data_length,
                          0x85, lo_n_prmtv_path);
    }

send_m_t_prmtv_above

Synopsis

extern void send_m_t_prmtv_above(14_buffer_number, 14_relay_baton, 14_data_start_offset,
                                  size, 14_code, path);
unsigned short 14_buffer_number;
unsigned short 14_relay_baton;
unsigned short 14_data_start_offset;
unsigned short size;
unsigned char 14_code;
unsigned char path;

Description

The send_m_t_prmtv_above monitor routine passes a specified interlayer message
buffer from Layer 4 to Layer 5 in an OSI monitor primitive.

Inputs

See send_t_prmtv_above. Use the monitor variables m_lo_n_il_buff,
m_lo_n_sdu_offset, and m_lo_n_sdu_size as input. Refer to variable
m_lo_t_prmtv_code in Table 63-6 for the appropriate primitive code.

Example

Make the appropriate variable declarations. For a condition monitoring RD data
primitives, the Layer 4 programming block should look like this:

LAYER: 4
STATE: send_buffer_up
    CONDITIONS: N_RD_DATA_IND
    ACTIONS:
    {
        _set_maint_buff_bit(m_lo_n_il_buff, &14_relay_baton);
        send_m_t_prmtv_above(m_lo_n_il_buff, 14_relay_baton, m_lo_n_sdu_offset,
                             m_lo_n_sdu_size, 0x85, m_lo_n_prmtv_path);
    }

63-56
send_n_prmtv_below

Synopsis

.extern void send_n_prmtv_below(il_buffer_number, il4_relay_baton, il4_data_start_offset, size,
   il4_code, path);
  unsigned short il_buffer_number;
  unsigned short il4_relay_baton;
  unsigned short il4_data_start_offset;
  unsigned short size;
  unsigned char il4_code;
  unsigned char path;

Description

The send_n_prmtv_below emulate routine passes a specified interlayer message buffer from Layer 4 to Layer 3 in an OSI primitive.

Inputs

The first parameter is the interlayer buffer number to be sent. For a buffer which has been received at Layer 4 from Layer 5, the variable up_t_il_buff may be used to identify the buffer number. If the buffer originated at Layer 4, use the buffer-number variable named in the _get_il_msg_buff routine. (See _insert_il_buff_list_cnt routine example at Layer 5.)

The second parameter is the returned maintain bit from a call to _set_maint_buff_bit. It is used only to pass a received buffer from Layer 4 to Layer 3. As soon as Layer 3 processing on the buffer is completed, the bit is automatically freed. If the buffer originated at Layer 4, use the maintain bit variable named in the _get_il_msg_buff routine. (See _insert_il_buff_list_cnt routine example at Layer 5.)

The third parameter is the offset to the Layer 4 list header node in the buffer. For a buffer which has been received at Layer 4 from Layer 5, the variable up_t_sdu may be used to indicate the offset.

The fourth parameter is the size of the data in the buffer. It will always be set to zero since the data length is unknown in a primitive being passed down the layers.

The fifth parameter is the code specifying the type of primitive in which the buffer will be sent. Refer to variable up_n_prmtv_code in Table 63-4 for the appropriate primitive code.

The sixth parameter is the path number along which the buffer will be sent. For a buffer which has been received at Layer 4 from Layer 5, the variable up_t_prmtv_path may be used to specify the path number.
Example

A buffer is received at Layer 4 from Layer 5. No text will be inserted at Layer 4. (For information on inserting text, see _insert_il_buff_list_cnt routine.) The buffer will be passed to Layer 3, requiring a new maintain bit to be set. If values are entered for the code and path, variables for code and path need not be declared.

```c
{ extern volatile unsigned short up_t_il_buff;
  extern volatile unsigned short up_t_sdu;
  unsigned short L4_relay_baton;
}

LAYER: 4
STATE: pass_buffer_down
CONDITIONS: T_DATA_REQ
ACTIONS:
{ _set_maint_bufLbU(up_t_il_buff, L4_relay_baton);
  send_n_prmtv_below(up_t_il_buff, L4_relay_baton, up_t_sdu, 0, Ox64, 0);
}
```

(F) Layer 5 OSI Routines

The following routines pass OSI primitives from Layer 5 to either Layer 6 or Layer 4.

send_s_prmtv_above

Synopsis

```c
extern void send_s_prmtv_above(il_buffer_number, L5_relay_baton, L5_data_start_offset, size, L5_code, path);
unsigned short il_buffer_number;
unsigned short L5_relay_baton;
unsigned short L5_data_start_offset;
unsigned short size;
unsigned char L5_code;
unsigned char path;
```

Description

The send_s_prmtv_above emulate routine passes a specified inter-layer message buffer from Layer 5 to Layer 6 in an OSI primitive.

Inputs

The first parameter is the inter-layer buffer number to be sent. For a buffer which has been received at Layer 5 from Layer 4, the variable lo_t_ilBuff may be used to identify the buffer number.
The second parameter is the returned maintain bit from a call to _set_maint_buff_bit. It is used only to pass a received buffer from Layer 5 to Layer 6. As soon as Layer 6 processing on the buffer is completed, the bit is automatically freed.

The third parameter is the offset to the Layer 5 service data unit in a received buffer. The variable lo_t_sdu contains the offset to the service data unit when the buffer reached Layer 5. The offset must be incremented by the length of the Layer 5 header, if any.

**NOTE:** In general, do not modify **extern** variables, such as lo_t_sdu, which may be updated by other processes. Name another variable, assign it the same value, and then increment that variable. Or, after lo_t_sdu has been named in the argument of the send routine, add the length of the Layer 5 header, if any.

The fourth parameter is the length of the data in the buffer. Use the length indicated in the **pdu** structure—**pdu.data_length**. Then subtract the length of the Layer 5 header, if any.

The fifth parameter is the code specifying the type of primitive in which the buffer will be sent. Refer to variable lo_s_prmtv_code in Table 63-7 for the appropriate primitive code.

The sixth parameter is the path number along which the buffer will be sent. For a buffer which has been received at Layer 5 from Layer 4, the variable lo_t_prmtv_path may be used to specify the path number.

**Example**

A buffer is received at Layer 5 from Layer 4. The offset to and size of the service data unit will be adjusted if needed, a new maintain bit will be set, and the buffer will be passed up to Layer 6.

```c
struct pdu
{
  unsigned char primitive_code;
  unsigned char path;
  unsigned long parameter;
  unsigned short relay_baton;
  unsigned short il_buffer_number;
  unsigned char buffer_contents;
  unsigned short data_start_offset;
  unsigned short data_length;
};
struct pdu * pdu_ptr;
extern volatile unsigned short lo_t_pdu_seg;
extern volatile const unsigned char lo_t_prmtv_path;
extern volatile unsigned short lo_t_il_buff;
extern volatile unsigned short lo_t_sdu;
unsigned short l5_relay_baton;
```
INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

LAYER: 5
STATE: send_buffer_up
CONDITIONS: T_DATA IND
ACTIONS:
{
    pdu_ptr = (void *)((long)lo_t_pdu_seg << 16);
    _set_main_t_buf(m_lo_t_il_buff, &15_relay_baton);
    send_s_prmtv_above(lo_t_il_buff, is_data, pdu_ptr->data_length, 0xa5, lo_t_prmtv_path);
}

send_m_s_prmtv_above

Synopsis

extern void send_m_s_prmtv_above(15_buffer_number, 15_relay_baton, 15_data_start_offset, size, 15_code, path);
unsigned short 15_buffer_number;
unsigned short 15_relay_baton;
unsigned short 15_data_start_offset;
unsigned short size;
unsigned char 15_code;
unsigned char path;

Description

The send_m_s_prmtv_above monitor routine passes a specified inter-layer message buffer from Layer 5 to Layer 6 in an OSI monitor primitive.

Inputs

See send_s_prmtv_above. Use the monitor m_lo_t_il_buff, m_lo_t_sdu_offset, and m_lo_t_sdu_size variables as input. Refer to variable m_lo_s_prmtv_code in Table 63-7 for the appropriate primitive code.

Example

Make the appropriate variable declarations. For a condition monitoring RD data primitives, the Layer 5 programming block should look like this:

LAYER: 5
STATE: send_buffer_up
CONDITIONS: T_RD_DATA IND
ACTIONS:
{
    _set_main_t_buf(m_lo_t_il_buff, &15_relay_baton);
    send_m_s_prmtv_above(m_lo_t_il_buff, 15_relay_baton, m_lo_t_sdu_offset, m_lo_t_sdu_size, 0xa5, m_lo_t_prmtv_path);
}
send_t_prmtv_below

Synopsis

extern void send_t_prmtv_below(il_buffer_number, IS_relay_baton, IS_data_start_offset, size,
   IS_code, path);
unsigned short il_buffer_number;
unsigned short IS_relay_baton;
unsigned short IS_data_start_offset;
unsigned short size;
unsigned char IS_code;
unsigned char path;

Description

The send_t_prmtv_below emulate routine passes a specified inter-layer message buffer from Layer 5 to Layer 4 in an OSI primitive.

Inputs

The first parameter is the inter-layer buffer number to be sent. For a buffer which has been received at Layer 5 from Layer 6, the variable up_s_il_buff may be used to identify the buffer number. If the buffer originated at Layer 5, use the buffer-number variable named in the _get_il_msg_buff routine. (See _insert_il_buff_list_cnt routine example at Layer 5.)

The second parameter is the returned maintain bit from a call to _set_maint_buff_bit. It is used only to pass a received buffer from Layer 5 to Layer 4. As soon as Layer 4 processing on the buffer is completed, the bit is automatically freed. If the buffer originated at Layer 5, use the maintain bit variable named in the _get_il_msg_buff routine. (See _insert_il_buff_list_cnt routine example at Layer 5.)

The third parameter is the offset to the Layer 5 list header node in the buffer. For a buffer which has been received at Layer 5 from Layer 6, the variable up_s_sdu may be used to indicate the offset.

The fourth parameter is the size of the data in the buffer. It will always be set to zero since the data length is unknown in a primitive being passed down the layers.

The fifth parameter is the code specifying the type of primitive in which the buffer will be sent. Refer to variable up_t_prmtv_code in Table 63-5 for the appropriate primitive code.

The sixth parameter is the path number along which the buffer will be sent. For a buffer which has been received at Layer 5 from Layer 6, the variable up_s_prmtv_path may be used to specify the path number.
Example

A buffer is received at Layer 5 from Layer 6. No text will be inserted at Layer 5. (For information on inserting text, see \_insert\_il\_buff\_list\_cnt routine.) The buffer will be passed to Layer 4, requiring a new maintain bit to be set. If values are entered for the code and path, variables for code and path need not be declared.

```c
{ extern volatile unsigned short up_s_il_buff;
  extern volatile unsigned short up_s_sdu;
  unsigned short IS_relay_baton;
}
```

**LAYER: 5**

**STATE:** pass_buffer_down

**CONDITIONS:** S\_DATA\_REQ

**ACTIONS:**

```c
{ _set_maint_buff_bit(up_s_il_buff, &IS_relay_baton);
  send_t_prmtv_below(up_s_il_buff, IS_relay_baton, up_s_sdu, 0, Ox84, 0);
}
```

(G) Layer 6 OSI Routines

The following routines pass OSI primitives from Layer 6 to either Layer 7 or Layer 5.

**send_p_prmtv_above**

**Synopsis**

```
extern void send_p_prmtv_above(il_buffer_number, 16_relay_baton, 16_data_start_offset, size,
  16_code, path);
```

**Description**

The `send_p_prmtv_above` emulate routine passes a specified interlayer message buffer from Layer 6 to Layer 7 in an OSI primitive.

**Inputs**

The first parameter is the interlayer buffer number to be sent. For a buffer which has been received at Layer 6 from Layer 5, the variable `lo_s_il_buff` may be used to identify the buffer number.
The second parameter is the returned maintain bit from a call to \_set\_maint\_buff\_bit. It is used only to pass a received buffer from Layer 6 to Layer 7. As soon as Layer 7 processing on the buffer is completed, the bit is automatically freed.

The third parameter is the offset to the Layer 6 service data unit in a received buffer. The variable lo_s_sdu contains the offset to the service data unit when the buffer reached Layer 6. The offset must be incremented by the length of the Layer 6 header, if any.

NOTE: In general, do not modify extern variables, such as lo_s_sdu, which may be updated by other processes. Name another variable, assign it the same value, and then increment that variable. Or, after lo_s_sdu has been named in the argument of the send routine, add the length of the Layer 6 header, if any.

The fourth parameter is the length of the data in the buffer. Use the length indicated in the pdu structure—pdu.data\_length. Then subtract the length of the Layer 6 header, if any.

The fifth parameter is the code specifying the type of primitive in which the buffer will be sent. Refer to variable lo_p_prmtv\_code in Table 63-8 for the appropriate primitive code.

The sixth parameter is the path number along which the buffer will be sent. For a buffer which has been received at Layer 6 from Layer 5, the variable lo_s_prmtv\_path may be used to specify the path number.

Example

A buffer is received at Layer 6 from Layer 5. The offset to and size of the service data unit will be adjusted if needed, a new maintain bit will be set, and the buffer will be passed up to Layer 7.

{  
struct pdu
  {
    unsigned char primitive_code;
    unsigned char path;
    unsigned long parameter;
    unsigned short relay_baton;
    unsigned short il_buffer_number;
    unsigned char buffer_contents;
    unsigned short data_start_offset;
    unsigned short data_length;
  };
struct pdu * pdu_ptr;
extern volatile unsigned short lo_s_pdu_seg;
extern volatile const unsigned char lo_s_prmtv_path;
extern volatile unsigned short lo_s_il_buff;
extern volatile unsigned short lo_s_sdu;
unsigned short 16_relay_baton;
}

LAYER: 6
STATE: send_buffer_up
CONDITIONS: S_DATA IND
ACTIONS:
{
    pdu_ptr = (void *)((long)lo_s_pdu_seg << 16);
    _set_maint_buff_bit(lo_s_il_buff, &16_relay_baton);
    send_p_prmtv_above(lo_s_il_buff, 16_relay_baton, lo_s_sdu, pdu_ptr->data_length, 0xc5, lo_s_prmtv_path);
}

send_m_p_prmtv_above

Synopsis

extern void send_m_p_prmtv_above(il_buffer_number, 16_relay_baton, 16_data_start_offset, size, 16_code, path);
unsigned short il_buffer_number;
unsigned short 16_relay_baton;
unsigned short 16_data_start_offset;
unsigned short size;
unsigned char 16_code;
unsigned char path;

Description

The send_m_p_prmtv_above monitor routine passes a specified interlayer message buffer from Layer 6 to Layer 7 in an OSI monitor primitive.

Inputs

See send_p_prmtv_above. Use the monitor variables m_lo_s_il_buff, m_lo_s_sdu_offset, and m_lo_s_sdu_size as input. Refer to variable m_lo_p_prmtv_code in Table 63-8 for the appropriate primitive code.

Example

Make the appropriate variable declarations. For a condition monitoring RD data primitives, the Layer 6 programming block should look like this:

LAYER: 6
STATE: send_buffer_up
CONDITIONS: S_RD_DATA IND
ACTIONS:
{
    _set_maint_buff_bit(m_lo_s_il_buff, &16_relay_baton);
    send_m_p_prmtv_above(m_lo_s_il_buff, 16_relay_baton, m_lo_s_sdu_offset, m_lo_s_sdu_size, 0xc5, m_lo_s_prmtv_path);
}
send_s_prmtv_below

Synopsis

extern void send_s_prmtv_below(il_buffer_number, i6_relay_baton, i6_data_start_offset, size,
   i6_code, path);
unsigned short il_buffer_number;
unsigned short i6_relay_baton;
unsigned short i6_data_start_offset;
unsigned short size;
unsigned char i6_code;
unsigned char path;

Description

The send_s_prmtv_below emulate routine passes a specified interlayer message buffer from Layer 6 to Layer 5 in an OSI primitive.

Inputs

The first parameter is the interlayer buffer number to be sent. For a buffer which has been received at Layer 6 from Layer 7, the variable up_p_il_buff may be used to identify the buffer number. If the buffer originated at Layer 6, use the buffer-number variable named in the _get_il_msg_buff routine. (See _insert_il_buff_list_cnt routine example at Layer 5.)

The second parameter is the returned maintain bit from a call to _set_maint_buff_bit. It is used only to pass a received buffer from Layer 6 to Layer 5. As soon as Layer 5 processing on the buffer is completed, the bit is automatically freed. If the buffer originated at Layer 6, use the maintain bit variable named in the _get_il_msg_buff routine. (See _insert_il_buff_list_cnt routine example at Layer 5.)

The third parameter is the offset to the Layer 6 list header node in the buffer. For a buffer which has been received at Layer 6 from Layer 7, the variable up_p_sdu may be used to indicate the offset.

The fourth parameter is the size of the data in the buffer. It will always be set to zero since the data length is unknown in a primitive being passed down the layers.

The fifth parameter is the code specifying the type of primitive in which the buffer will be sent. Refer to variable up_s_prmtv_code in Table 63-6 for the appropriate primitive code.

The sixth parameter is the path number along which the buffer will be sent. For a buffer which has been received at Layer 6 from Layer 7, the variable up_p_prmtv_path may be used to specify the path number.
Example

A buffer is received at Layer 6 from Layer 7. No text will be inserted at Layer 6. (For information on inserting text, see _insert_il_buff_list_cnt routine.) The buffer will be passed to Layer 5, requiring a new maintain bit to be set. If values are entered for the code and path, variables for code and path need not be declared.

```
{ 
  extern volatile unsigned short up_p_il_buff;
  extern volatile unsigned short up_p_sdu;
  unsigned short il6_relay_baton;
}
```

**LAYER: 6**

**STATE:** pass_buffer_down

**CONDITIONS:** P_DATA_REQ

**ACTIONS:**

```
  _set_maint_buff_bit(up_p_il_buff, &il6_relay_baton);
  send_p_prmtv_below(up_p_il_buff, il6_relay_baton, up_p_sdu, 0, 0xa4, 0);
```

(H) Layer 7 OSI Routines

**send_p_prmtv_below**

**Synopsis**

```
extern void send_p_prmtv_below(il_buffer_number, relay_baton, data_start_offset, size, code, path);
unsigned short il_buffer_number;
unsigned short relay_baton;
unsigned short data_start_offset;
unsigned short size;
unsigned char code;
unsigned char path;
```

**Description**

The `send_p_prmtv_below` emulate routine passes a specified interlayer message buffer from Layer 7 to Layer 6 in an OSI primitive.

**Inputs**

The first parameter is the interlayer buffer number to be sent. Use the buffer-number variable named in the _get_il_msg_buff routine. (See _insert_il_buff_list_cnt routine example at Layer 5.)

The second parameter is the returned maintain bit from the call to _get_il_msg_buff.
The third parameter is the returned offset (from a call to \_start il\_buff\_list) to the Layer 7 list header node in the buffer.

The fourth parameter is the size of the data in the buffer. It will always be set to zero since the data length is unknown in a primitive being passed down the layers.

The fifth parameter is the code specifying the type of primitive in which the buffer will be sent. Refer to variable up\_p\_prmtv\_code in Table 63-7 for the appropriate code.

The sixth parameter is the path number along which the buffer will be sent.

**Example**

A buffer is obtained at Layer 7. The buffer will be passed to Layer 6, without any data inserted. (For information on inserting text, see \_insert il\_buff\_list\_cnt routine.) If values are entered for the code and path, variables for code and path need not be declared.

```c
{  
  unsigned short il_buffer_number;  
  unsigned short data_start_offset;  
  unsigned short relay_baton;  
}
```

**Layer: 7**

**State:** pass\_buffer\_down

**Conditions:** Keyboard

**Actions:**

```c
{  
  _get il\_msg\_buff(&il_buffer_number, &relay_baton);  
  _start il\_buff\_list(il_buffer_number, &data_start_offset);  
  send p\_prmtv\_below(il_buffer_number, relay_baton, data_start_offset, 0, 0xc4, 0);  
}
```
64 Print

The PRINTER port is a serial interface through which the programmer may direct output from the INTERVIEW to a printer. The printer port is located at the rear of the INTERVIEW between the REMOTE RS-232 and AUXILIARY ports.

NOTE: Before directing output to the printer port, configure the Printer Setup menu as explained in Section 14.2.

Each spreadsheet PRINT action or call to one of the C print routines causes output to be added to a queue of unprinted text in the print buffer. If not doing so already, the print server also begins to poll the print buffer for text to print. As long as there is unprinted text in the buffer, the print server polls the buffer, removes text, and sends it to the printer port of the INTERVIEW. Use the _print_buffer structure to monitor the flow of text in and out of the print buffer.

Use any of the four C print routines explained in this section to add text to the print buffer. Three of them—printf, printf, and prints—are similar to the displayc, displayf, and displays routines which direct output to the Display Window. See Section 61.3(C). With the set_print_header routine, you determine the heading which will appear at the top of each printed page. One other routine, sprintf, writes output to a string. The string can then be referenced in subsequent calls to printf. (You may also use the string named in sprintf in calls to displayf, tracef, or fprintf.)

64.1 Structures

Refer to Table 64-1 for the structure of the print buffer. Compare _print_buffer.in with _print_buffer.out to determine whether or not the print buffer has emptied. When the values of these two variables are equal, the buffer is empty.

NOTE: Consider the variables in the _print_buffer structure read-only variables. In general, do not modify extern structures or variables which may be updated by other processes.

At times, processes may add transactions to the print buffer more quickly than the print server takes them out. If a process cannot add to the buffer without overwriting unprinted text, a buffer overrun occurs. When your INTERVIEW is configured for
data playback, you can minimize print-buffer overruns by periodically suspending playback and allowing the print server to empty the buffer. In judging how often to suspend playback, keep in mind the following points: 1) In general, the more conditions a program has that trigger print actions, the more frequently playback should be suspended. 2) When planning to print Run-mode buffers, remember that the faster the playback speed, the quicker the print buffer fills.

Table 64-1
Print Structures

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Structure Name:</strong> print_buffer</td>
<td></td>
<td></td>
<td>Structure of the print buffer. Declared as type struct.</td>
</tr>
<tr>
<td>unsigned short</td>
<td>in</td>
<td>a-207/10-8199</td>
<td>offset into the print buffer (from the physical beginning of the buffer) to the location where next transaction text will be added. Advances with each spreadsheet PRINT action or call to a C print routine. When in equals out, the print buffer is empty.</td>
</tr>
<tr>
<td>unsigned short</td>
<td>out</td>
<td>a-207/10-8199</td>
<td>offset into the print buffer (from the physical beginning of the buffer) to the last transaction text printed from the buffer. Advances each time text is actually sent out the printer port of the INTERVIEW. When out equals in, the print buffer is empty.</td>
</tr>
<tr>
<td>unsigned short</td>
<td>buffer_end</td>
<td>209/8201</td>
<td>offset to the physical end of the print buffer—i.e., to the end of the array named buffer (see below)</td>
</tr>
<tr>
<td>unsigned short</td>
<td>lock</td>
<td></td>
<td>when process is printing, locks out other processes from accessing the print buffer</td>
</tr>
<tr>
<td>char</td>
<td>polling</td>
<td>0</td>
<td>non-zero</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>print server is not polling</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>print server is polling print buffer for text to print</td>
</tr>
<tr>
<td>char</td>
<td>overrun</td>
<td>0</td>
<td>non-zero</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>print buffer is not in overrun state</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>print buffer is in overrun state—i.e., a process attempting to add text to the print buffer can't be sent because unprinted text in the buffer would be overwritten. Following message will appear on printout: “print buffer overrun has occurred.”</td>
</tr>
<tr>
<td>char</td>
<td>buffer [8192]</td>
<td></td>
<td>array of text transactions</td>
</tr>
</tbody>
</table>

**Structure Name:** _print_buffer

An instance of the print_buffer structure, declared as type extern struct print_buffer. Use the variables contained in this structure to monitor flow of text in and out of the print buffer. Reference structure variables as follows: _print_buffer.in.
The following example shows how you might use a \texttt{TIMEOUT} condition to check the print buffer periodically. Each time the timeout expires, the program determines whether or not the buffer is half full. If so, playback is suspended. If the buffer is only one-quarter full, playback is resumed. (Other conditions in the program, not illustrated here, would cause print actions to send output to the print buffer.)

\begin{verbatim}
#define PRINT_BUFFER_SZ 8192
#define STOP_POINT (PRINT_BUFFER_SZ/2)
#define START_POINT (PRINT_BUFFER_SZ/4)

LAYER: 1
{
 struct print_buffer
 {
  unsigned short in;
  unsigned short out;
  unsigned short buffer_end;
  unsigned short lock;
  char polling;
  char overrun;
  }
 extern struct print_buffer _print_buffer;
 int crnt_buffer_sz;
}

STATE: check_print_buffer
CONDITIONS: ENTER_STATE
ACTIONS: TIMEOUT RESTART ck_buffer 0.01
CONDITIONS: TIMEOUT ok_buffer
ACTIONS:
{  crnt_buffer_sz = ((_print_buffer.in + PRINT_BUFFER_SZ) - _print_buffer.out) % PRINT_BUFFER_SZ;
  if(crnt_buffer_sz > STOP_POINT)
    suspend_rnd_play();
  else if(crnt_buffer_sz < START_POINT)
    start_rnd_play();
}  
TIMEOUT RESTART ck_buffer 0.01
\end{verbatim}

\section{Variables}

There are no variables associated exclusively with print functions.
64.3 Routines

printf

Synopsis

extern int printf(format_ptr, ...);
const char * format_ptr;

Description

The printf routine writes output to the print buffer for printing, under control of the string pointed to by format_ptr that specifies how subsequent arguments are converted for output. If there are insufficient arguments for the format, the behavior is
undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated but otherwise ignored. The printf routine returns when the end of the format string is encountered.

**Inputs**

The format is composed of zero or more directives: ordinary characters (not %), which are copied unchanged to the output stream; and conversion specifications, each of which results in fetching zero or more subsequent arguments. Each conversion specification is introduced by the character %. After the %, the following appear in sequence:

- **Zero or more flags** that modify the meaning of the conversion specification. The flag characters and their meanings are:
  - The result of the conversion will be left-justified within the field.
  - The result of a signed conversion will always begin with a plus or minus sign.

  *space* If the first character of a signed conversion is not a sign, a space will be prepended to the result. If the *space* and + flags both appear, the *space* flag will be ignored.

  *#* The result is to be converted to an "alternate form." For *d*, *i*, *u*, *c*, and *s* conversions, the flag has no effect. For *o* conversion, it increases the precision to force the first digit of the result to be a zero. For *x* (or *X*) conversion, a nonzero result will have Ox (or Ox) prepended to it.

- An optional decimal integer specifying a minimum *field width*. If the converted value has fewer characters than the field width, it will be padded on the left (or right, if the left adjustment flag, described above, has been given) to the field width. The padding is with spaces unless the field width integer starts with a zero, in which case the padding is with zeros.

- An optional *precision* that gives the minimum number of digits to appear for the *d*, *i*, *o*, *u*, *x*, and *X* conversions, or the maximum number of characters to be written from an array in an *s* conversion. The precision takes the form of a period (.) followed by an optional decimal integer; if the integer is omitted, it is treated as zero. The amount of padding specified by the precision overrides that specified by the field width.

- An optional *h* specifying that a following *d*, *i*, *o*, *u*, *x*, or *X* conversion specifier applies to a *short int* or *unsigned short int* argument (the argument will have been promoted according to the integral promotions, and its value shall be converted to *short int* or *unsigned short int* before printing); or an optional *l* specifying that a following *d*, *i*, *o*, *u*, *x*, or *X* conversion specifier applies to a *long int* or *unsigned long int* argument. If an *h* or *l* appears with any other conversion specifier, it is ignored.
A character that specifies the type of conversion to be applied. (Special AR extensions have been added.) The conversion specifiers and their meanings are:

d, i, o, u, x, X

The int argument is converted to signed decimal (d or i), unsigned octal (o), unsigned decimal (u), or unsigned hexadecimal notation (x or X); the letters abcdef are used for x conversion and the letters ABCDEF for X conversion. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeros. The default precision is 1. The result of converting a zero value with a precision of zero is no characters.

c The int argument is converted to an unsigned char, and the resulting character is written.

s The argument shall be a pointer to a null-terminated array of 8-bit chars. Characters from the string are printed up to (but not including) the terminating null character: if the precision is specified, no more than that many characters are printed. The string may be an array into which output was written via the sprintf routine.

p The argument shall be a pointer to void. The value of the pointer is converted to a sequence of printable characters, in this format: 0000:0000. There are always exactly 4 digits to the right of the colon. The number of digits to the left of the colon is determined by the pointer's value and the precision specified. Use this conversion to print 80286 memory addresses. The segment number will appear to the left of the colon and the offset to the right.

% A % is written. No argument is converted.

\n Writes hexadecimal OD OAt the ASCII carriage-return and linefeed characters. No argument is converted.

If a conversion specification is invalid, the behavior is undefined.

If any argument is or points to an aggregate (except for an array of characters using %s conversion or any pointer using %p conversion), the behavior is undefined.

In no case does a nonexistent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is expanded to contain the conversion result.

Returns

The printf routine returns the number of characters output.

Example

To print a date and time in the form “Sunday, July 3, 10:02,” where weekday and month are pointers to strings:
LAYER: 1
{
  unsigned char date_time [100];
  unsigned char weekday [10];
  unsigned char month [10];
  unsigned short day;
  unsigned char hour;
  unsigned char min;
}

STATE: output_to_printer
CONDITIONS: KEYBOARD • •
ACTIONS:
{
  printf("%s, %s %d, %.2d:%.2d\n", weekday, month, day, hour, min);
}

sprintf

Synopsis

extern int sprintf(string_ptr, format_ptr);
unsigned char string [128];
const char * format_ptr;

Description

The sprintf routine is similar to the printf routine, except that sprintf writes output to a string, while printf writes output directly to the print buffer for printing. The sprintf routine is useful for writing formatted output to a display, printer, or file.

The output is under control of the string pointed to by format_ptr that specifies how subsequent arguments are converted for output. If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated but otherwise ignored. The sprintf routine returns when the end of the format string is encountered.

Inputs

The first parameter is a pointer to the array to which output will be written.

For the second parameter, see printf routine.

Returns

This routine returns the number of characters written into the array, not counting the added null terminating character.

Example

Refer again to the sample program for the displayf routine in Section 61.3(C). This time you also want to send the output to a printer. By using the sprintf routine, you only have to enter the format string once.
INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

LAYER: 1
{
    unsigned char date_time [100];
    unsigned char weekday [10];
    unsigned char month [10];
    unsigned short day;
    unsigned char hour;
    unsigned char min;
}

STATE: output_to_display_window_and_printer
    CONDITIONS: KEYBOARD * *
    ACTIONS:
    {
        sprintf(date_time, "%s, %s %d, %.2d: %.2d\n", weekday, month, day, hour, min);
        displayf("%s", date_time);
        printf("%s", date_time);
    }

set_print_header

Synopsis
extern int set_print_header(format_ptr);
const char * format_ptr;

Description
This routine writes output to the print buffer, to be printed after each form feed, under control of the string pointed to by format_ptr. Paging is done automatically by the INTERVIEW. The set_print_header routine returns when the end of the format string is encountered.

Inputs
The format is composed of zero or more ordinary characters. Octal or hexadecimal values also may be input, with octal preceded by \ and hex by \x. Pad each value to three integers with leading zeroes.

The status information shown above the prompt line on the display screens of the INTERVIEW can be sent to a printer with the following inputs:

#d date (mm/dd/yy)
#t time (hh:mm)
#p page (not shown on the display screens)
#b block number
## #

Returns
The set_print_header routine returns the length of the header (0-255), or a -1 if the header exceeds the buffer size.
Example

If you want the date, time, and page number to appear in the heading on each page sent to a printer, enter the following:

```
LAYER: 2
STATE: header
CONDITIONS: ENTER_STATE
ACTIONS:
{  
  set_print_header("#### #d #t
}
```

The printer output will look like this:

```
## 09/01/89 09:30
...
## 09/01/89 09:31
...
```

prints

Synopsis

```
extern void prints(string_ptr);
const char * string_ptr;
```

Description

The prints routine is similar to the displays routines, except that prints writes output to the print buffer for printing while displays writes output to the Display Window. The output is under control of the string pointed to by the argument. The prints routine returns when the end of the string is encountered. The softkey equivalent of this routine is the PRINT PROMPT action on the Protocol Spreadsheet. A PRINT PROMPT action automatically time-stamps the output. Although prints does not, you can create your own time or date stamp with set_print_header.

Inputs

The input is a pointer to a string composed of zero or more ordinary characters. The newline nonliteral sequence \"\n\" writes hex 0D 0A (ASCII \r\n) to the output string. Octal or hexadecimal values also may be included in the string, with octal preceded by \ and hex by \x. Pad each value to three integers with leading zeroes.
Example

The following entry

`prints("End of test.");`

produces the following output to a printer:

End of test.
65 Disk I/O

The disk I/O routines explained in this section allow disk files to be read from and written to during Run mode. "Streams" describes how most of the routines operate on a data stream rather than the actual file. Under "Routines," all the disk I/O routines are explained. These routines perform read and write functions as well as other file maintenance tasks in Run mode, such as creating directories, renaming files, and deleting files.

65.1 Streams

Most disk I/O routines are not executed on the actual disk file, but on a stream which includes a copy of the file's data. Opening a disk file for reading or writing associates a stream with the file. A stream may be input or output. Input streams are read-only. Output streams are write-only. In either case, the stream remains associated with a disk file until the file is closed.

You may have more than one stream associated with a given file. (A maximum of ten streams may be open at one time.) For example, to read from and write to an existing file, you must open the file twice, once to create an input stream and once to create an output stream.

(A) Stream Components

A stream contains everything needed to perform disk I/O functions on a file.

1. Buffer. A buffer containing a copy of the data in a disk file is part of the stream. When a disk file is opened for reading, sectors of the disk containing the file are copied to this buffer.

Sometimes a file's size may exceed the maximum size (512 bytes) of the buffer. In this instance, as much data from the file as will fit in the buffer is copied. As each character is read from the input stream, it is removed. (The ungetc routine may temporarily return a removed character to an input stream.) Each call to fread, fgetc, or fgets further empties the buffer, while leaving the contents of the disk file unchanged. When the buffer is empty, the next sector (or sectors) of the disk file is (are) automatically copied into the buffer.

Similarly, when a file is opened for writing, the empty buffer is filled as fwrite or other output routines are invoked. Characters written to the output stream are not transferred to the disk file until there is a call to fflush. fflush is automatic in fclose or when the stream buffer is full.
2. **File-position indicator.** The file-position indicator keeps track of progression through the disk file. For files opened in read mode, the indicator is initially located at the first character (character zero) in the file. As characters are read from the input stream, the indicator advances through the file.

For existing files opened in append mode, the indicator is positioned after the last character in the file. For newly created files or files opened in overwrite mode, it is located at the beginning of the file. Every time an output routine is executed, the file-position indicator is advanced by the number of characters successfully written to the stream.

3. **Buffer pointer.** The stream also contains a pointer into the associated buffer of a file. In input streams, it points to the next character to be read. In output streams, it points to the next empty byte.

4. **EOF indicator.** If the end-of-file (EOF) indicator is set in a input stream, it means that a read operation encountered the end of the file. The EOF indicator is cleared via calls to fopen, fseek, rewind, clearerr, or ungetc.

5. **Error indicator.** In input streams, this indicator gets set when an fread, fgetc, or fgets routine does not successfully execute. Attempting to execute these input routines (or ungetc) on an output stream sets the error indicator. In output streams, the error indicator gets set when the fflush, fwrite, fputc, fputs, or fprintf routine does not successfully execute, or when output routines try to execute on an input stream. A call to fopen, clearerr, or fseek, clears the error indicator in either input or output streams. A rewind operation on an input stream also clears the indicator.

(B) **Stream Pointer**

The fopen routine returns a pointer to the stream. Disk I/O routines which perform operations on a stream require the stream pointer as an argument. It has been named stream_ptr in the routines discussed below.

(C) **Locking Streams**

Each file stream is locked internally during operations on it. If the user program is executing different conditions on multiple processors and both actions require writing to the same file stream, internally the stdio library will allow the first task that requests to write to execute until completion and the second task will be locked out. All processes that are locked out are temporarily put to sleep and removed from the tasking queues for that CPU. When the first process completes its operations on the stream, the locked-out processes are woken up and may try to claim the lock. Deadlock or deadly embrace situations can never arise internally to the stdio library.
If two or more file streams are associated with a single file, processes on each stream may try to operate on the file concurrently. Internal locking does not apply in this situation, so use the locking routines.

65.2 Routines

Disk I/O routines fall into four categories. The first category includes routines valid for both input and output streams, including the two locking routines (not exclusive to disk I/O). The remaining groups are routines valid for input streams only, routines applicable to output streams only, and routines which handle other file maintenance functions.

The routines and their descriptions closely conform to the ANSI specification for the Programming Language C, as defined in the draft document published July 9, 1986. Discrepancies with the ANSI standard are noted. The document number is X3J11-86-098. Refer to pages 107-129.

Use the #include <stdio.h> pre-processor directive with all disk I/O routines. The stdio.h file contains type definitions and function prototypes, making declarations of the routines unnecessary.

When a filename is required as an argument, give the absolute pathname of the file, prefixed by the device name. Valid device names are FD1, FD2, or HRD. See Section 13.2(B) for a discussion of absolute pathnames. The disk filename is required as an argument for the fopen routine, which opens a file for reading or writing. From that point on, disk I/O routines relating to that file use the stream pointer, explained above, as input. File maintenance routines, such as rename or remove, use the filename as input.

NOTE: A single program can perform disk I/O functions as well as data playback or recording. Disk I/O, however, must be suspended while disk recording (or playback) proceeds, and vice versa. RAM recording, on the other hand, may occur simultaneously with disk I/O operations. Refer to the start_rcrd_play and suspend_rcrd_play routines in Section 69 for more information on the interaction between disk I/O and recording/playback.

(A) Input/Output-Stream Routines

Several disk I/O routines may be executed on either input or output streams. fopen opens an existing disk file for reading or writing, or creates a new file. In each case, a stream is associated with the file until there is a call to fclose. fclose or a specific call to fflush delivers any output written to a stream to the host environment where it will be written to the disk file.

NOTE: Always include a call to fclose in your program to make sure output is written to the file.
Test the end-of-file and error indicators with the `feof` and `ferror` routines, respectively. These same indicators may be cleared via the `clearerr` routine.

The `fseek` and `rewind` routines manipulate the file-position indicator and erase any memory of a character put into the stream via `ungetc`.

The `lock` and `unlock` routines prevent deadlock from occurring when processes on multiple streams try to operate concurrently on a single file.

**fopen**

**Synopsis**

```c
#include <stdio.h>
extern FILE * fopen(char * filename, char * mode);
const char * filename;
const char * mode;
```

**Description**

The `fopen` routine opens a file for access. Depending on the open mode, a file can be opened for reading (via an input stream) or for writing (via an output stream). For existing files, this routine also clears the end-of-file and error indicators.

**Inputs**

The first parameter is a pointer to the file to be opened, represented as the name of the file, placed inside double quotation marks. The filename must be the absolute pathname, prefixed by the device name (HRD, FD1, or FD2).

The second parameter is a pointer to a string (represented as a character inside double quotation marks) which identifies the type of open to be performed. Of the ANSI standard open modes, the following are supported:

- **r**
  - Open an existing file for reading only. The file-position indicator is located at the start (character zero) of the file.

- **w**
  - Create a file, or open an existing file, for writing only. For an existing file, truncate its length to zero and discard the contents.

- **a**
  - Create a file, or open an existing file, for writing only. For an existing file, retain the contents and locate the file-position indicator at the end of the file. Append new data to the end of existing data, unless a call to `fseek` or `rewind` has repositioned the file-position indicator. In this instance, overwrite existing data. (This implementation is different from the ANSI specification which appends new data to the end of existing data regardless of any previous calls to `fseek`.)
rb  Currently implemented the same as "r." Use "rb" for the fseek
    routine.

wb  Currently implemented the same as "w." Use "wb" for the fseek
    routine.

ab  Currently implemented the same as "a." Use "ab" for the fseek
    routine.

Returns

This routine returns a pointer to the stream, with a type definition FILE
(defined in the stdio.h file).

If the open fails (for example, the file does not exist), zero is returned.

Example

Open a file called "buff01" in the /usr directory on a disk in floppy drive 2.
Store the pointer to the stream in stream_ptr. Indicate whether or not the open
is successful on the prompt line.

{  
#include <stdio.h>
FILE * stream_ptr;
}

LAYER: 1
STATE: open_a_file
CONDITIONS: ENTER_STATE
ACTIONS: PROMPT "Press 0 to open file.
CONDITIONS: KEYBOARD "cO"
ACTIONS:
   
   if((stream_ptr = fopen("FD2/usr/buff01", "r")) == 0)
      display_prompt("Cannot open file.
   else
      display_prompt("File opened.

fclose

Synopsis

#include <stdio.h>
extern int fclose(stream_ptr);
FILE * stream_ptr;

Description

All opened files must be closed. If the disk file to be closed is an input file,
then any data remaining in the stream buffer is discarded. If the file is an
output file, any data written to the stream is written to the file. (In other words,
fclose automatically calls fflush.) The stream is freed from its association with
the disk file, and the disk file is closed.

65-5
Inputs
The only parameter is the stream pointer.

Returns
If the stream is successfully closed, zero is returned. If errors are detected, or if the stream is already closed, a non-zero value is returned.

Example
Close the file that was opened in the fopen example. Indicate whether or not the close is successful on the prompt line.

```c
#include <stdio.h>
FILE * stream_ptr;

LAYER: 1
STATE: open_and_close_a_file
CONDITIONS: ENTER_STATE
ACTIONS: PROMPT "Press 0 to open file."
CONDITIONS: KEYBOARD "00"
ACTIONS:
{
if((stream_ptr = fopen("FD2iusr\buf01", "r")) == 0)
    display_prompt("Cannot open file.");
else
    display_prompt("File opened.");
}
CONDITIONS: KEYBOARD "C"
ACTIONS:
{
if(fclose(stream_ptr) != 0)
    display_prompt("Either file is already closed, or close cannot be executed.");
else
    display_prompt("File closed.");
}
```

**fflush**

**Synopsis**

```c
#include <stdio.h>
extern int fflush(FILE * stream_ptr);
```

**Description**

If `stream_ptr` points to an output stream, the `fflush` routine causes any unwritten data for that stream to be delivered to the host environment where it will be written to the file. If `stream_ptr` points to an input stream, the `fflush` routine undoes the effect of any preceding `ungetc` operation on the stream.

**Inputs**

The only parameter is the stream pointer.
Returns
If a write error occurs, non-zero is returned and the error indicator is set.

Example
Assume the X.25 personality package has been loaded in at Layer 2. Whenever you receive a frame type "unknown," write the actual value of the control byte to an output file stream and to the disk file.

```
#include <stdio.h>
FILE * stream_ptr;
extern volatile const unsigned char rcvd_frame_ctrl_byte_1;
```

```
LAYER: 2
STATE: write_then_fflush
CONDITIONS: ENTER_STATE
ACTIONS:
{
    if ((stream_ptr = fopen("/usr/frame_unkwn", "a")) == 0)
        display_prompt("Cannot open file.");
    else
        display_prompt("File opened.
        pos_cursor(1,0);

CONDITIONS: RCV UNKNOWN
ACTIONS:
{
    if (printf(stream_ptr, " %02x\n ", rcvd_frame_ctrl_byte_1) < 0)
        display("Error in printing to stream.
    else
        display("Print to stream completed.
    if (fflush(stream_ptr) != 0)
        display_prompt("Write error.
    else
        display_prompt("Write to file completed. Press C to close file.

CONDITIONS: KEYBOARD "cC"
ACTIONS:
{
    if (fclose(stream_ptr) != 0)
        display_prompt("Either file is already closed, or close cannot be executed.
    else
        display_prompt("File closed.
```

feof

Synopsis
#include <stdio.h>
extern int feof(stream_ptr);
FILE * stream_ptr;

Description
This routine tests the end-of-file indicator for an associated stream.
Inputs
The only parameter is the stream pointer.

Returns
The `feof` routine returns a non-zero value if the end-of-file indicator is set for
the stream.

Example
Get a character from a file. If it is not at the end of the file, display it;
otherwise prompt with "End of file."

```c
#include <stdio.h>
FILE * stream_ptr;
int character;

LAYER: 1
STATE: test_for_eof
CONDITIONS: ENTER_STATE
  ACTIONS: PROMPT "Press 0 to open file."
  CONDITIONS: KEYBOARD "00"
  ACTIONS:
  {
    if((stream_ptr = fopen("FD2/usr/buff01", "rb")) == 0)
      display_prompt("Cannot open file.");
    else
      display_prompt("File opened.. Press G to get character.
pos_cursor(1,0);
  }
  CONDITIONS: KEYBOARD "gG"
  ACTIONS:
  {
    character = fgetc(stream_ptr);
    if(feof(stream_ptr) != 0)
      display_prompt("End of file. Press C to close file.
else
      displayf("%c", character);
  }
  CONDITIONS: KEYBOARD "cC"
  ACTIONS:
  {
    if(close(stream_ptr) != 0)
      display_prompt("Either file is already closed, or close cannot be executed.
else
      display_prompt("File closed.
  }

ferror
Synopsis
#include <stdio.h>
extern int ferror(stream_ptr);
FILE * stream_ptr;
```
Description

This routine tests the error indicator for a stream.

Inputs

The only parameter is the stream pointer.

Returns

The *ferror* routine returns a non-zero value if the error indicator is set for the stream.

Example

Read a file called "buff01" from the /usr directory on the disk in drive 2. If the number of elements read is less than the number designated to be read, determine whether an end-of-file was encountered or a read error occurred.

```c
#include <stdio.h>
FILE * stream_ptr;
char data [6091];
si32_t n;
}
LAYER: 1
STATE: read_a_file
CONDITIONS: ENTER_STATE
ACTIONS: PROMPT "Press 0 to open file."
CONDITIONS: KEYBOARD 00
ACTIONS:
{
  if((stream_ptr = fopen("FD2/usr(buff01", "r") == 0) 
      display_prompt("Cannot open file.");
  else
      display_prompt("File opened. Press R to read the file.");
}
CONDITIONS: KEYBOARD R
ACTIONS:
{
  n = fread(data, 1, 6091, stream_ptr);
  if(n != 6091)
  {
    if(ferror(stream_ptr) != 0) 
      display_prompt("Read error.");
    else if(feof(stream_ptr) != 0)
      display_prompt("End-of-file encountered.");
  }
  else
    display("\n%.6091s", data);
    display_prompt("Press C to close the file.");
}```
clearerr

Synopsis

#include <stdio.h>
extern void clearerr(stream_ptr);
FILE * stream_ptr;

Description

This routine clears the end-of-file and error indicators for a stream. When an error occurs, no further operations are allowed until the error indicators are explicitly cleared. (These indicators are also cleared by a fopen or rewind operation.)

Inputs

The only parameter is the stream pointer.

Example

If a write error occurs, clear the indicators.

{
#include <stdio.h>
FILE * stream_ptr;
int character;
}

LAYER: 1
STATE: clear_indicators
CONDITIONS: ENTER_STATE
ACTIONS: PROMPT *Press O to open file.*
CONDITIONS: KEYBOARD *00*
ACTIONS:
{
    if((stream_ptr = fopen("FD2/usb/buff01", "wb")) == 0)
        display_prompt("Cannot open file.");
    else
        display_prompt("File opened. Press P to write character.");
}
CONDITIONS: KEYBOARD "pP"

ACTIONS:
{
    character = fputc('h', stream_ptr);
    if(character == EOF)
    {
        display_prompt("Write error. All indicators will be cleared.
                    clearerr(stream_ptr);
    }
    else
        display_prompt("Write completed. Press C to close the file.
    }

CONDITIONS: KEYBOARD "cc"

ACTIONS:
{
    if(fclose(stream_ptr) != 0)
        display_prompt("Either file is already closed, or close cannot be executed.  
    else
        display_prompt("File closed.

fseek

Synopsis

#include <stdio.h>
extern int fseek(stream_ptr, bytes, reference);
FILE * stream_ptr;
long int bytes;
int reference;

Description

This routine manipulates the file-position indicator, according to the ANSI specification for binary files. Future read operations will be referenced from that point. fseek clears the end-of-file indicator and resets the ungetc variable.

NOTE: The ANSI specification for text files is not currently implemented. To ensure proper execution of fseek if future releases include the ANSI specification for text files, open files for fseek as binary ("rb," "wb," or "ab").

Inputs

The first parameter is the stream pointer.

The second parameter is the number of characters the file-position indicator should be moved from a specified position. A positive number advances the file-position indicator forward in the file; a negative number moves it backward.
The third parameter specifies the location of the file-position indicator. 
SEEK_SET will move the file-position indicator from the beginning of the file; 
SEEK_END will move the file-position indicator from the end-of-file; and 
SEEK_CUR will move the file-position indicator from its current position.

**Returns**

This routine returns non-zero for an improper request; otherwise it returns zero.

**Example**

Open a file and move the file-position indicator 4 characters from the beginning of the file. Each time the 9 key is pressed, move the indicator one character backward from its current position. After 4 executions, the indicator will be back at the beginning of the file.

```c
#include <stdio.h>
FILE *stream_ptr;
int character;

layer: 1

STATE: move_indicator
CONDITIONS: ENTER_STATE
ACTIONS: PROMPT "Press C to open file.
CONDITIONS: KEYBOARD "oo"
ACTIONS:
{
    if((stream_ptr = fopen("FD2/usr/buff01", "rb")) == 0)
        display_prompt("Cannot open file.
    else
        display_prompt("File opened. ");
        pos_cursor(0,14);
        if(fseek(stream_ptr, 4, SEEK_SET) != 0)
            displays("Improper fseek request.
        else
            displays("Fseek completed. Press S to seek new position.
    }
}
CONDITIONS: KEYBOARD "SS"
ACTIONS:
{
    if(fseek(stream_ptr, -1, SEEK_CUR) != 0)
        display_prompt("Improper fseek request. Press C to close file.
    else
        display_prompt("Fseek completed. Press C to close file.
```
CONDITIONS: KEYBOARD "cC"

ACTIONS:
{
    if(fclose(stream_ptr) != 0)
        display_prompt("Either file is already closed, or close cannot be executed.");
    else
        display_prompt("File closed.");
}

rewind

Synopsis

#include <stdio.h>
extern void rewind(stream_ptr);
FILE * stream_ptr;

Description

This routine returns the file-position indicator to the beginning of the file (i.e., it is equivalent to an fseek with the number of characters to move set as zero and the specified position SEEK_SET). The rewind operation also clears the end-of-file and error indicators and erases any memory of the character in a previous ungetc operation.

Inputs

The only parameter is the stream pointer.

Example

In this example, the first call to fgetc following the rewind operation will read the first character in the file.

{
    #include <stdio.h>
    FILE * stream_ptr;
    int character;
}

layer: 1
STATE: move_indicator
CONDITIONS: ENTER_STATE
ACTIONS: PROMPT "Press O to open file.
CONDITIONS: KEYBOARD "oO"
ACTIONS:
{
    if((stream_ptr = fopen("FD2/usr/buff01", "rb")) == 0)
        display_prompt("Cannot open file.");
    else
        display_prompt("File opened. Press S to fseek.");
}
CONDITIONS: KEYBOARD *ss*
ACTIONS:
{
    if (fseek(stream_ptr, 4, SEEK_SET) != 0)
        display_prompt("Improper fseek request.");
    else
        display_prompt("Fseek completed. Press spacebar to rewind.");
}
CONDITIONS: KEYBOARD *gG*
ACTIONS:
{
    rewind(stream_ptr);
    display_prompt("Press G to get a character.");
}
CONDITIONS: KEYBOARD *cc*
ACTIONS:
{
    if (fclose(stream_ptr) != 0)
        display_prompt("Either file is already closed, or close cannot be executed.");
    else
        display_prompt("File closed.");
}

lock

Synopsis

#include <stdio.h>
extern void lock(lock_variable_ptr);
int * lock_variable_ptr;

Description

The lock routine implements a lock using the integer variable pointed to by the routine parameter. If the lock variable is currently locked, the task goes to sleep. When an unlock on the same variable occurs (within an independent task), the task invoking the lock function will attempt to claim the lock. If successful, the task is executed; otherwise, it goes back to sleep until the next unlock.

NOTE: If locking is used at any place in the program, all related or possibly concurrent routines must also use the locking functions.
NOTE: The lock variable should always be defined as a global integer, never as local to a function. The lock variable should never be altered by the user program or deadlock can occur. Deadlock also results if the lock is invoked twice within the same task without an intervening unlock.

**Inputs**

The only parameter is a pointer to the lock variable.

**Example**

Two tasks concurrently write to their own file streams. The file streams are local to the routine `write_fox`, making them independent of each other even though both are referenced by `stream_ptr`. During the `fclose` operation (which automatically calls `fflush`), however, both tasks need to write to the same file. The locking routines ensure that the writes to the file occur sequentially, not concurrently.

```c
#include <stdio.h>
const char data [] = "Quick Brown Fox\n";
int key;
void write_fox()
{
    FILE * stream_ptr;
    size_t n;
    lock(&key);
    if((stream_ptr = fopen("../usr/buff01", "a")) == 0)
        display_prompt("Cannot open file.");
    else
        display_prompt("File opened.");
    n = fwrite(data, 1, sizeof(data)-1, stream_ptr);
    pos_cursor(1,0);
    if(n != (sizeof(data)-1))
        display("Write error.\n");
    else
        display("Write completed.\n");
    if(fclose(stream_ptr) != 0)
        display("Either file is already closed, or close cannot be executed.\n");
    else
        display("File closed.\n");
    unlock(&key);
}
```
TEST: b
STATE: write_only
CONDITIONS: ON_SIGNAL xyz
ACTIONS:
{
  write_fox();
}

unlock

Synopsis

#include <stdio.h>
extern void unlock(lock_variable_ptr);
int * lock_variable_ptr;

Description

The unlock routine implements the inverse of the lock routine using the same
integer variable. Sleeping tasks will be woken up to retry their attempt to claim
the lock. One will succeed, and the rest will go back to sleep. See also lock
routine.

Inputs

The only parameter is a pointer to the lock variable.

Example

See lock routine.

(B) Input-Stream Routines

The following routines are valid for input streams only. An attempt to apply
them to output streams results in a read error. The error indicator for the input
stream will be set.

Three routines read characters from the input stream. The fread and fgets
routines transfer a specified number of characters from the stream buffer into a
user-defined array. fgetc reads the next character from the input stream. The
ungetc routine temporarily forces a designated character back into the input
stream.

fread

Synopsis

#include <stdio.h>
extern size_t fread(data_ptr, size, number, stream_ptr);
void * data_ptr;
size_t size;
size_t number;
FILE * stream_ptr;
Description

This routine reads elements from the input-stream buffer and puts them into a user-defined buffer. The file-position indicator is advanced by the number of characters successfully read. The fread routine can read a file whose elements are more than eight bits each, 16-bit shorts or 32-bit longs, for example. The fgets routine is similar to fread. fgets, however, reads only 8-bit characters. The primary use of fread is to read the entire contents of a file, whereas the primary purpose of fgets is to read from a file one line at a time.

Inputs

The first parameter is a pointer to an array in which the incoming data should be placed.

The second parameter is the number of bytes in each element to be read. If the value of this parameter is zero, the contents of the array and the stream remain unchanged.

The third parameter is the number of elements to be read. If the value of this parameter is zero, the contents of the array and the stream remain unchanged.

The fourth parameter is the stream pointer.

Returns

The fread routine returns the total number of elements read. If the number of elements read is less than the number of elements designated to be read, an end-of-file has been encountered or a read error has occurred. Use thefeof and ferror routines to distinguish an end-of-file from a read error. If an error occurs, the location of the file-position indicator is indeterminate.

Example

Read in a file called "buff01" from the /usr directory on the disk in drive 2 and display it on the Program Trace screen. (See Section 61.4 for information on using trace buffers in C.) Determine the size of the array data from the file size indicated on the File Maintenance screen.

```c
#include <trace_buf.h>
#include <stdio.h>
FILE *stream_ptr;
char data [6091];
size_t n;
extern struct trace_buf prog_trbuf;
```
INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

LAYER: 1
STATE: read_a_file
CONDITIONS: ENTER_STATE
ACTIONS: PROMPT "Press O to open file.
CONDITIONS: KEYBOARD "oO"
ACTIONS:
{ if((stream_ptr = fopen("FD2/usr/buff01", "r")) == 0)
  display_prompt("Cannot open file.
else
  display_prompt("File opened. Press R to read the file.
}
CONDITIONS: KEYBOARD "rR"
ACTIONS:
{ n = fread(data, 1, 6091, stream_ptr);
if(n != 6091)
  display_prompt("Either a read error has occurred, or an EOF has been
encountered.");
else
  tracef(&prog_trbuf, "%.6091s", data);
  display_prompt("Press C to close the file.
}
CONDITIONS: KEYBOARD "cC"
ACTIONS:
{ if(close(stream_ptr) != 0)
  display_prompt("Either file is already closed, or close cannot be executed.");
else
  display_prompt("File closed.
}

fgets

Synopsis

#include <stdio.h>
extern char * fgets(string_ptr, max_number, stream_ptr);
char * string_ptr;
int max_number;
FILE * stream_ptr;

Description

This routine gets at the most one less than the specified number of characters
from an input stream and puts them in an array. If an EOF, newline, or null is
encountered in the stream, no more characters will be read, even if the specified
number of characters has not yet been read. The newline will be retained. A
terminating null character is written after the last character read into the array.
The file-position indicator is advanced by the number of characters successfully
read. The fgets routine is similar to fread. The fread routine can read a file
whose elements are more than eight bits each, 16-bit *shorts* or 32-bit *longs*, for example. *fgets*, however, reads only 8-bit characters. The primary use of *fgets* is to read from a file one line at a time.

**Inputs**

The first parameter is a pointer to the array into which the characters will be put.

The second parameter is the maximum number of characters (minus one) to be read.

The third parameter is the stream pointer.

**Returns**

If the routine is successful, a pointer to the array is returned. If end-of-file is encountered before any characters have been read into the array or if a read error occurs, a null pointer is returned. The contents of the array are indeterminate when a read error occurs.

**Example**

Five characters, at the most, from a disk file will be put into an array called *data* and displayed on the screen.

```c
#include <stdio.h>
FILE * stream_ptr;
char data [10];
```

```c
LAYER: 1
STATE: read_characters
CONDITONS: ENTER_STATE
ACTIONS: PROMPT "Press O to open file.
CONDITONS: KEYBOARD "oo"
ACTIONS:
{
   if((stream_ptr = fopen("FD2/usr/buff01", "r")) == 0)
      display_prompt("Cannot open file.
else
      display_prompt("File opened. Press G to get string.
}
CONDITONS: KEYBOARD "G"
ACTIONS:
{
   fgets(data, 6, stream_ptr);
   displayf("\n%.6s", data);
   display_prompt("Press C to close the file.
}
```
CONDITIONS: KEYBOARD *cC*

ACTIONS:
{
    if(fclose(stream_ptr) != 0)
        display_prompt("Either file is already closed, or close cannot be executed.");
    else
        display_prompt("File closed.");
}

fgetc

Synopsis

```c
#include <stdio.h>
extern int fgetc(stream_ptr);
FILE * stream_ptr;
```

Description

The `fgetc` routine gets the next character (if present) from the input stream. The character is an `unsigned char` cast to an `int` (stored in the least-significant byte of the `int`). The file-position indicator advances by one character.

Inputs

The only parameter is the stream pointer.

Returns

This routine returns the next character in the input stream. EOF is returned if an end-of-file is encountered or if a read error occurs. The `stdio.h` file defines the macro EOF as -1. Use the `feof` and `ferror` routines to determine the reason for a returned EOF.

Example

In the following example, open an input file for reading. Each time the `p` key is pressed, display the next character in the file.

```c
#include <stdio.h>
FILE * stream_ptr;
int character, end;
}
```

65-20
CONDITIONS: KEYBOARD "gG"
ACTIONS:
{
    character = fgetc(stream_ptr);
    if(character == EOF)
    {
        end = fseek(stream_ptr);
        if(end != 0)
            display_prompt("EOF encountered.");
        else
            display_prompt("Read error.");
    }
    else
        displayf("%c", character);
}

CONDITIONS: KEYBOARD "oC"
ACTIONS:
{
    if(fclose(stream_ptr) != 0)
        display_prompt("Either file is already closed, or close cannot be executed.");
    else
        display_prompt("File closed.");
}

ungetc

Synopsis
#include <stdio.h>
extern int ungetc(character, stream_ptr);
int character;
FILE * stream_ptr;

Description
This routine temporarily forces a specified character into a variable associated with the input stream, overwriting the previous ungetc variable. The routine does not affect the location of the file-position indicator. The next fgetc will read the ungetc variable, not the stream. An intervening fflush, fseek, or rewind erases memory of the character. If the ungetc function is called too many times on the same stream without an intervening read, fflush, fseek, or rewind operation on that stream, the operation may fail. Ungetc also clears the end-of-file indicator.

Inputs
The first parameter is the character to be put into the input stream.
The second parameter is the stream pointer.

Returns
This routine returns the specified character. If the operation fails, EOF is returned and the input stream remains unchanged. It will fail if the values of the specified character and the macro EOF are equal.
Example

Read a character from the stream. Press the U key when you want to return the last character read to the stream. The next call to fgetc will read the returned character.

```c
#include <stdio.h>
FILE * stream_ptr;
int character;
}
LAYER: 1
STATE: get_next_character
CONDITIONS: ENTER_STATE
CONDITIONS: KEYBOARD "G"
ACTIONS:
{  
  if((stream_ptr = fopen("FD2/usr/buff01", "r")) == 0)  
    display_prompt("Cannot open file.
  else
    display_prompt("File opened. Press G to get a character.
  
CONDITIONS: KEYBOARD "G"
ACTIONS:
{  
  character = fgetc(stream_ptr);
  if(character == EOF)
    display_prompt("End of file or read error.
  else
    {  
      pos_cursor(0,0);
      display("character = %c  Press U to return character to stream.", character);
    }
  
CONDITIONS: KEYBOARD "U"
ACTIONS:
{  
  if((ungetc(character, stream_ptr)) == EOF)
    display_prompt("Character not returned.
  else
    display_prompt("Character returned.
  }
CONDITIONS: KEYBOARD "G"
ACTIONS:
{  
  if(fclose(stream_ptr) != 0)
    display_prompt("Either file is already closed, or close cannot be executed.
  else
    display_prompt("File closed.
```

(C) Output–Stream Routines

The following routines are valid for output streams only. An attempt to apply them to input streams will result in a write error. The error indicator for the output stream will be set.

Four routines write to output streams. The fwrite and fputs routines transfer a specified number of characters from a user-defined array into the stream buffer. fpwrite writes a character to the next empty byte in an output-stream buffer. fprintf writes formatted output to an output stream similar to the way display writes output to the Display Window.

fwrite

Synopsis

#include <stdio.h>
extern size_t fwrite(output_ptr, size, number, stream_ptr);
const void * output_ptr;
size_t size;
size_t number;
FILE * stream_ptr;

Description

This routine writes elements from a user-defined array to the output-stream buffer. The file-position indicator is advanced by the number of characters successfully written.

Inputs

The first parameter is a pointer to an array from which the data should be taken. Declare it as const if it is read-only. In cases where the array will be written to, as in the example below, do not include const as part of the declaration.

The second parameter is the number of bytes in each element to be written.

The third parameter is the number of elements to be written.

The fourth parameter is the stream pointer.

Returns

The fwrite routine returns the total number of elements written. If the number of elements written is less than the number of elements designated to be written, a write error has occurred. If an error occurs, the location of the file-position indicator is indeterminate.
Example

Read the contents of a file, and write them to a new file.

```c
#include <stdio.h>
FILE * read_stream;
FILE * write_stream;
char output [6091];
size_t n;
}

LAYER: 1
STATE: write_to_a_file
CONDITIONS: ENTER_STATE
   ACTIONS: PROMPT "Press 0 to open files.
       CONDITIONS: KEYBOARD "oO"
   ACTIONS:
   { if((read_stream = fopen("FD2/usr/buff01", "r")) == 0)
       display_prompt("Cannot open buff01. ");
       pos_cursor(0, 21);
   } else
       display_prompt("Buff01 opened. ");
       pos_cursor(0, 16);
   }
   if((write_stream = fopen("FD2/usr/new_file", "w")) == 0)
       display("Cannot open new file.");
   else
       display("New file opened. Press R to read buff01.");
   }
CONDITIONS: KEYBOARD "R"
   ACTIONS:
   { n = fread(output, 1, 6091, read_stream);
     if(n != 6091)
         display_prompt("Either a read error has occurred, or an EOF has been encountered. ");
   } else
       display_prompt("Press W to write to new_file.");
   }
CONDITIONS: KEYBOARD "W"
   ACTIONS:
   { n = fwrite(output, 1, 6091, write_stream);
     if(n != 6091)
         display_prompt("Write error. Press C to close files.");
     else
         display_prompt("Write completed. Press C to close files.");
   }
CONDITIONS: KEYBOARD "cC"
ACTIONS:
{
    if(fclose(read_stream) != 0)
    {
        display_prompt("Either buff01 is already closed, or close cannot be executed.");
        pos_cursor(0,0);
    }
    else
    {
        display_prompt("Buff01 closed.");
        pos_cursor(0,16);
    }
    if(fclose(write_stream) != 0)
        displays("Either new_file is already closed, or close cannot be executed.");
    else
        displays("New_file closed.");
}

fputs

Synopsis

#include <stdio.h>
extern int fputs(string_ptr, stream_ptr);
const char * string_ptr;
FILE * stream_ptr;

Description

This routine writes a string of characters from an array, excluding the
terminating null character, to the output stream. The file-position indicator is
advanced by the number of characters successfully written.

Inputs

The first parameter is a pointer to the string to be written.
The second parameter is the stream pointer.

Returns

This routine returns zero if it is successful; it returns a non-zero value if a write
error occurs.

Example

Write a fox message at the end of existing data in a file.

{ #include <stdio.h> FILE * stream_ptr;
  char data [] = "(FOX)\n"; }
INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

LAYER: 1
STATE: write_a_string
CONDITIONS: ENTER_STATE
ACTIONS: PROMPT "Press 0 to open file.
CONDITIONS: KEYBOARD "00"
ACTIONS:
{ if((stream_ptr = fopen("FD2/usr/buff01", "a")) == 0)
   display_prompt("Cannot open file.
else
   display_prompt("File opened. Press P to write string.
}
CONDITIONS: KEYBOARD "pP"
ACTIONS:
{ if(fputc(data, stream_ptr) != 0)
   display_prompt("Write error. Press C to close file.
else
   display_prompt("Write completed. Press C to close file.
}
CONDITIONS: KEYBOARD "rC"
ACTIONS:
{ if(fclose(stream_ptr) != 0)
   display_prompt("Either file is already closed, or close cannot be executed.
else
   display_prompt("File closed.

fputc

Synopsis

#include <stdio.h>
extern int fputc(character, stream_ptr);
int character;
FILE * stream_ptr;

Description

This routine writes a given character (cast to an unsigned char) to an output stream. The file-position indicator advances one character.

Inputs

The first parameter is the character to be written to the output stream. It may be given as a hexadecimal, octal, or decimal constant; as an alphanumeric constant inside single quotes; or as a variable. A hexadecimal value must be preceded by the prefix 0x or 0X; an octal value must be preceded by the prefix 0. If no prefix appears before the input, the number is assumed to be decimal.

The second parameter is the stream pointer.

65-26
Returns

If the character is successfully written to the output stream, the routine returns that character. If a write error occurs, EOF is returned and the error indicator is set.

Example

Open the named file. If the file does not already exist, create it. If it does exist, truncate its length to zero, thereby deleting its contents. Put the character read from the input stream pointed to by read_stream into the output stream pointed to by write_stream. This example is similar to the one given for fwrite, except that in this case, each time the 0 key is pressed, only one character is copied, rather than the entire file.

```c
#include <stdio.h>
FILE * read_stream;
FILE * write_stream;
int character;

LAYER: 1
STATE: copy_a_character
CONDITIONS: ENTER_STATE
ACTIONS: PROMPT "Press 0 to open files.
   CONDITIONS: KEYBOARD "00"
ACTIONS:
   
   if((read_stream = fopen("FD2/usr/buff01", "rU)) == 0)
   
   display_prompt("Cannot open buff01. ");
   pos_cursor(0,21);
   
   else
   
   display_prompt("Buff01 opened. ");
   pos_cursor(0,16);
   
   if((write_stream = fopen("FD2/usr/buff01_copy", "w")) == 0)
   
   displays("Cannot open buff01_copy.
   else
   
   displays("Buff01_copy opened. Press P to copy a character.
   
   CONDITIONS: KEYBOARD "pP"
ACTIONS:
   
   character = fgetc(read_stream);
   if(character == EOF)
   
   if(feof(read_stream) != 0)
   
   display_prompt("EOF encountered. Press C to close files.
   else
   
   display_prompt("Read error. Press C to close files.
   
   
   65-27
```
else
    fputc(character, write_stream);
}

CONDITIONS: KEYBOARD *cC*

ACTIONS:
{
    if(fclose(read_stream) != 0)
    {
        display_prompt("Either buff01 is already closed, or close cannot be executed. ");
        pos_cursor(0,0);
    } else
    {
        display_prompt("Buff01 closed. ");
        pos_cursor(0,16);
    }
    if(fclose(write_stream) != 0)
        display("Either buff01_copy is already closed, or close cannot be executed. ");
    else
        display("Buff01_copy closed. ");
}

---

fprintf

Synopsis

#include <stdio.h>
extern int fprintf(stream_ptr, format_ptr, ...);
FILE * stream_ptr;
char * format_ptr;

Description

The fprintf routine is similar to the sprintf routine, except that fprintf writes output to an output stream, while sprintf writes output to an array. The output is under control of the string pointed to by format_ptr that specifies how subsequent arguments are converted for output. If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated but otherwise ignored. The fprintf routine returns when the end of the format string is encountered. (Sprintf is documented in Section 64.3.)

Inputs

The first parameter is the stream pointer.

The second parameter points to the format string composed of zero or more directives: ordinary characters (not %), which are copied unchanged to the output stream; and conversion specifications, each of which results in fetching zero or more subsequent arguments. Each conversion specification is introduced by the character %. After the %, the following appear in sequence:
Zero or more flags that modify the meaning of the conversion specification. The flag characters and their meanings are:

- The result of the conversion will be left-justified within the field.
+ The result of a signed conversion will always begin with a plus or minus sign.

space If the first character of a signed conversion is not a sign, a space will be prepended to the result. If the space and + flags both appear, the space flag will be ignored.

# The result is to be converted to an "alternate form." For d, i, c, and s conversions, the flag has no effect. For o conversion, it increases the precision to force the first digit of the result to be a zero. For x (or X) conversion, a nonzero result will have Ox (or OX) prepended to it.

- An optional decimal integer specifying a minimum field width. If the converted value has fewer characters than the field width, it will be padded on the left (or right, if the left adjustment flag, described above, has been given) to the field width. The padding is with spaces unless the field width integer starts with a zero, in which case the padding is with zeros.

- An optional precision that gives the minimum number of digits to appear for the d, i, o, u, x, and X conversions or the maximum number of characters to be written from an array in an s conversion. The precision takes the form of a period (.) followed by an optional decimal integer; if the integer is omitted, it is treated as zero. The amount of padding specified by the precision overrides that specified by the field width.

- An optional h specifying that a following d, i, o, u, x, or X conversion specifier applies to a short int or unsigned short int argument (the argument will have been promoted according to the integral promotions, and its value shall be converted to short int or unsigned short int before printing); or an optional l specifying that a following d, i, o, u, x, or X conversion specifier applies to a long int or unsigned long int argument. If an h or l appears with any other conversion specifier, it is ignored.

- A character that specifies the type of conversion to be applied. (Special AR extensions have been added.) The conversion specifiers and their meanings are:
The \textit{int} argument is converted to signed decimal (d or i), unsigned
octal (o), unsigned decimal (u), or unsigned hexadecimal notation (x
or X); the letters abcdef are used for x conversion and the letters
ABCDEF for X conversion. The precision specifies the minimum
number of digits to appear; if the value being converted can be
represented in fewer digits, it will be expanded with leading zeros.
The default precision is 1. The result of converting a zero value with
a precision of zero is no characters.

The \textit{int} argument is converted to an \textit{unsigned char}, and the resulting
character is written.

The argument shall be a pointer to a null-terminated array of 8-bit
\textit{chars}. Characters from the string are written up to (but not including)
the terminating null character: if the precision is specified, no more
than that many characters are written. The string may be an array
into which output was written via the \textit{printf} routine.

The argument shall be a pointer to void. The value of the pointer is
converted to a sequence of printable characters, in this format:
\texttt{0000:0000}. There are always exactly 4 digits to the right of the
colon. The number of digits to the left of the colon is determined by
the pointer's value and the precision specified. Use this conversion to
display 80286 memory addresses. The 16-bit segment number will
appear to the left of the colon and the 16-bit offset to the right.

A \texttt{\%} is written. No argument is converted.

\texttt{\textbackslash n} Writes hexadecimal \texttt{0A}, the ASCII linefeed character. No argument
is converted.

If a conversion specification is invalid, the behavior is undefined.

If any argument is or points to an aggregate (except for an array of characters
using \texttt{\%s} conversion or any pointer using \texttt{\%p} conversion), the behavior is
undefined.

In no case does a nonexistent or small field width cause truncation of a field; if
the result of a conversion is wider than the field width, the field is expanded to
contain the conversion result.

\textbf{Returns}

This routine returns the number of characters written, or a negative value if an
output error occurred.

\textbf{Example}

Assume the X.25 personality package has been loaded in at Layer 2. When an
unknown frame is received, copy the actual value of the control byte to an
output stream.


```c
#include <stdio.h>
FILE *stream_ptr;
extern volatile const unsigned char rcd_frame_cntrl_byte_l;
}
LAYER: 2
STATE: save_unknowns
CONDITIONS: ENTER_STATE
ACTIONS: PROMPT "Press O to open file.
CONDITIONS: ENTER_STATE
ACTIONS:
{
  if((stream_ptr = fopen("FD2\usn)frame_unkwn", "w")) == 0)
    display_prompt("Cannot open file.
    else
    display_prompt("File opened.

} CONDITIONS: RCV UNKNOWN
ACTIONS:
{
  if(printf(stream_ptr, "%02x\n", rcd_frame_cntrl_byte_l) < 0)
    display_prompt("Error in printing to stream.
    else
    display_prompt("Print to stream completed. Press C to close file.

} CONDITIONS: KEYBOARD "cC"
ACTIONS:
{
  if(fclose(stream_ptr) != 0)
    display_prompt("Either file is already closed, or close cannot be executed.
    else
    display_prompt("File closed.
```

(D) File Maintenance Routines

rename

Synopsis

```
#include <stdio.h>
extern int rename(oldfie_ptr, newfile_ptr);
const char * oldfile_ptr;
const char * newfile_ptr;
```

Description

This routine renames a specified file. A file can only be renamed if it resides on the active disk, indicated on the Current Directory line of the File Maintenance screen. Renaming an open file does not affect subsequent disk I/O operations on the stream. The stream is still associated with the same file, even though the filename has changed.
Inputs
The first parameter is a pointer to a string, the current name of the file. Give
the absolute pathname of the file, prefixed by the device name (HRD, FD1, or
FD2).

The second parameter is a pointer to a string, the new name to be given to the
file. Give the absolute pathname of the file, prefixed by the device name.

Returns
If the rename operation succeeds, zero is returned. If it fails, a non-zero value
is returned. If the renaming fails, the file will still be known by its original
name.

Example
Change the name of a file from old to backup. Prompt whether or not the
rename operation was successful.

```
#include <stdio.h>
```

```
LAYER: 1
STATE: rename
CONDITIONS: ENTER_STATE
ACTIONS: PROMPT "Press spacebar to rename file."
CONDITIONS: KEYBOARD ""
ACTIONS:
{ if(rename("FD1/usr/old", "FD1/usr/backup") != 0)
   display_prompt("Rename failed.
 else
   display_prompt("File has been renamed.
```
Returns

Zero is returned if the file is removed; non-zero if it is not (for example, the file does not exist in the specified location).

Example

Remove file oldfile from the /usr directory on the disk in floppy drive 1. Prompt whether or not the remove operation was successful.

```c
#include <stdio.h>

LAYER: 1
STATE: delete_a_file
CONDITIONS: ENTER_STATE
ACTIONS: PROMPT "Press D to delete file.
CONDITIONS: KEYBOARD "dD"
ACTIONS:
{
  if(remove("FD1/oldfile") != 0)
    display_prompt("File has not been deleted.
  else
    display_prompt("File deleted.

mkdir

Synopsis

#include <stdio.h>
extern int mkdir(directory_ptr);
char * directory_ptr;

Description

This routine creates a directory.

Inputs

The only parameter is a pointer to a string, i.e., the name of the directory to be created. The absolute pathname must be used, prefixed by the device name (FD1, FD2, or HRD).

Returns

If the directory is created, zero is returned; otherwise, a non-zero value is returned.

Example

Create a sub-directory called disk_i_o in the /usr directory on the disk in drive 2.

{  
#include <stdio.h>  
}  

LAYER: 1  
STATE: make_directory  
CONDITIONS: ENTER_STATE  
ACTIONS: PROMPT "Press M to make a directory."
          CONDITIONS: KEYBOARD "mM"
          ACTIONS: 
          {  
              if(mkdir("FD2/usridisk_i_a") != 0)
                  display_prompt("Directory not created.");  
              else  
                  display_prompt("Directory created.");  
          }

_set_file_type

Synopsis
#include <stdio.h>

extern int _set_file_type(pathname_ptr, type_buff_ptr);
char * pathname_ptr;
char * type_buff_ptr;

Description
This routine determines the type identification of a specified file on the File
Maintenance screen. If a file is created by a "w" or "a" open mode and a file
type is not specified with the _set_file_type routine, it will be designated as an
ASCII file.

Inputs
The first parameter is a pointer to a string, the name of the file. The filename
must be the absolute pathname, prefixed by the device name (HRD, FD1, or
FD2).

The second parameter is a pointer to a string, the file type. The type may be
any of the following (upper or lower case is acceptable):

SYS      System
DIR      Directory
PRGM     Program
SETUP    Setup
OBJ      Object code
LOBJ     Linkable-object
ASCII    ASCII
BITIM    Bit-image data
CHDAT    Character data

65–34
Returns

If the operation succeeds, the routine returns zero; otherwise, it returns a non-zero value.

Example

The following example is almost the same one used for `fwrite`: read the contents of a program file and write them to a new file. The difference is that `new_file` is set to be a program file. In the `fwrite` example, the type designation in the file directory would default to "ASCII." It would still load and run as a program file, however, since the file's contents, not its type label, determine which operations are valid.

```c
#include <stdio.h>
FILE * read_stream;
FILE * write_stream;
char output [6091];
size_t n;
}

LAYER: 1
STATE: write_to_a_file
CONDITIONS: ENTER_STATE
   ACTIONS: PROMPT *Press 0 to open files.
   CONDITIONS: KEYBOARD *00*
   ACTIONS:
   {
      if((read_stream = fopen("FD2/usr/buff01", "r")) == 0)
      {
         display_prompt("Cannot open buff01.");
         pos_cursor(0,21);
      }
      else
      {
         display_prompt("Buff01 opened.");
         pos_cursor(0,16);
      }
      if((write_stream = fopen("FD2/usr/new_file", "w")) == 0)
         display("Cannot open new_file.");
      else
         display("New_file opened. Press "sS" to set the file type.");
   }
   CONDITIONS: KEYBOARD "sS"
   ACTIONS:
   {
      if(_set_file_type("FD2/usr/new_file", "PRGM") != 0)
         display_prompt("File type not set. Press R to read buff01.");
      else
         display_prompt("File type set. Press R to read buff01.");
   }
```
 CONDITIONS: KEYBOARD "rR"
ACTIONS:
{  
  n = fread(output, 1, 6091, read_stream);
  if(n != 6091)
    display_prompt("Either a read error has occurred, or an EOF has been encountered.");
  else
    display_prompt("Press W to write to new file.");
}
CONDITIONS: KEYBOARD "wW"
ACTIONS:
{  
  n = fwrite(output, 1, 6091, write_stream);
  if(n != 6091)
    display_prompt("Write error. Press C to close files.");
  else
    display_prompt("Write completed. Press C to close files.");
}
CONDITIONS: KEYBOARD "cC"
ACTIONS:
{  
  if(fclose(read_stream) != 0)
    
    
    
    display_prompt("Either buff01 is already closed, or close cannot be executed.");
    pos_cursor(0,0);
  else
    
    
    display_prompt("Buff01 closed.");
    pos_cursor(0,16);
  
  if(fclose(write_stream) != 0)
    
    
    display("Either new_file is already closed, or close cannot be executed.");
  else
    
    
    display("New file closed.");
}

_get_file_type

Synopsis

#include <stdio.h>
extern int _get_file_type(pathname_ptr, type_buff_ptr);
char * pathname_ptr;
char * type_buff_ptr;

Description

This routine determines the type of a specified file.

Inputs

The first parameter is a pointer to a string, the name of the file. The filename must be the absolute pathname, prefixed by the device name (HRD, FD1, or FD2).
The second parameter is a pointer to an array in which the file type should be written. See _set_file_type for the different file types.

Returns

If the operation succeeds, the routine returns zero; otherwise, it returns a non-zero value.

Example

```c
#include <stdio.h>
FILE * stream_ptr;
char type [8];

LAYER: 1
STATE: find_type
  CONDITIONS: ENTER_STATE
  ACTIONS: PROMPT "Press G to get file type."
  CONDITIONS: KEYBOARD "gG"
  ACTIONS:
  {
    if(_get_file_type("FD2/ust/new_file", &type[0]) != 0)
      display_prompt("File type not found.");
    else
      printf("File type=%s.
", type);
  }
```
66 Status

The structures and variables referenced in this section provide information about the current status of the programmer's INTERVIEW. This information must be accessed via C coding on the Protocol Spreadsheet since these structures and variables have no softkey equivalents.

66.1 Unit Configuration

Two structures presented in Table 66-1 may be accessed by the user to identify current features of the INTERVIEW. unit_setup variables reflect current Line Setup menu and FEB tick-rate selections. unit_config variables contain information about the user's INTERVIEW hardware and software.

66.2 Current Display Mode

The variables display_screen_changed, crnt_display_screen, and prev_display_screen track movement via softkey from one display screen to another. These variables also indicate transitions between Run mode and Freeze mode. They are documented in Section 61.1.
### Table 66-1
**Status Structures**

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Structure Name:</strong> unit_setup</td>
<td></td>
<td></td>
<td>Structure containing Line Setup and FEB tick-rate selections. Declared as type extern struct. Reference member variables of the structure as follows: unit_setup.speed_dce.</td>
</tr>
<tr>
<td>unsigned long</td>
<td>speed_dce</td>
<td>a/10</td>
<td>10 usec</td>
</tr>
<tr>
<td></td>
<td></td>
<td>64/100</td>
<td>100 usec</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3e8/1000</td>
<td>1 msec</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2710/10000</td>
<td>10 msec</td>
</tr>
<tr>
<td></td>
<td></td>
<td>186a0/100000</td>
<td>1000 msec</td>
</tr>
<tr>
<td></td>
<td></td>
<td>f4240/1000000</td>
<td>1 sec</td>
</tr>
<tr>
<td>unsigned char</td>
<td>bit_order_polarity</td>
<td>0</td>
<td>normal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>normal-inverse</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>reverse-normal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>reverse-inverse</td>
</tr>
<tr>
<td>unsigned char</td>
<td>bits_per_byte</td>
<td>5-8</td>
<td>tick rate selected on FEB Setup</td>
</tr>
<tr>
<td>unsigned char</td>
<td>clocking_type</td>
<td>0</td>
<td>internal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>external</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>internal-split</td>
</tr>
<tr>
<td>unsigned char</td>
<td>data_source</td>
<td>0</td>
<td>disk</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>line</td>
</tr>
<tr>
<td>unsigned char</td>
<td>format</td>
<td>0</td>
<td>sync</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>bop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>async</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>isoc</td>
</tr>
<tr>
<td>unsigned char</td>
<td>mode</td>
<td>0</td>
<td>automonitor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>monitor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>emulate dce</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>emulate dte</td>
</tr>
<tr>
<td>unsigned char</td>
<td>parity</td>
<td>0</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>even</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>odd</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>mark</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>space</td>
</tr>
<tr>
<td>unsigned char</td>
<td>code_name [13]</td>
<td></td>
<td>ASCII, EBCDIC, etc.</td>
</tr>
</tbody>
</table>

66-2
Table 66-1 (continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structure Name: unit_config</td>
<td>Structure containing unit configuration. Declared as type extern struct. Reference member variables of the structure as follows: unit_config.floppy_exists_mask.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unsigned char</td>
<td>floppy_exists_mask</td>
<td>1</td>
<td>floppy1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>floppy2†</td>
</tr>
<tr>
<td>unsigned char</td>
<td>hard_disk</td>
<td>0</td>
<td>not present</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>present</td>
</tr>
<tr>
<td>unsigned char</td>
<td>test_board</td>
<td>0</td>
<td>not present</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>present</td>
</tr>
<tr>
<td>unsigned char</td>
<td>mux</td>
<td>0</td>
<td>not present</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>present</td>
</tr>
<tr>
<td>unsigned char</td>
<td>modem</td>
<td>0</td>
<td>not present</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>present</td>
</tr>
<tr>
<td>unsigned char</td>
<td>num_mpms</td>
<td>0-4</td>
<td>number of MPM boards present</td>
</tr>
<tr>
<td>struct mpm_info</td>
<td>mpm[4]</td>
<td></td>
<td>array of structures. Each element in the array is an instance of the structure mpm_info and corresponds to one of four MPM boards which may be present. Reference member variables of the structure elements in the array as follows: unit_config.mpm[0].present.</td>
</tr>
<tr>
<td>unsigned char</td>
<td>cpm_rev</td>
<td>reserved</td>
<td>reserved</td>
</tr>
<tr>
<td>unsigned char</td>
<td>gbm_rev</td>
<td>reserved</td>
<td>reserved</td>
</tr>
<tr>
<td>unsigned char</td>
<td>pcm_rev</td>
<td>reserved</td>
<td>reserved</td>
</tr>
<tr>
<td>unsigned char</td>
<td>modem_rev</td>
<td>reserved</td>
<td>reserved</td>
</tr>
<tr>
<td>unsigned char</td>
<td>mux_rev</td>
<td>reserved</td>
<td>reserved</td>
</tr>
<tr>
<td>unsigned char</td>
<td>tim_type</td>
<td>f0/240</td>
<td>RS-232</td>
</tr>
<tr>
<td></td>
<td></td>
<td>f1/241</td>
<td>X.21</td>
</tr>
<tr>
<td></td>
<td></td>
<td>f2/242</td>
<td>V.35</td>
</tr>
<tr>
<td></td>
<td></td>
<td>f3/243</td>
<td>RS-449</td>
</tr>
<tr>
<td></td>
<td></td>
<td>f4/244</td>
<td>expansion adaptor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>f5/245</td>
<td>RC-8245</td>
</tr>
<tr>
<td></td>
<td></td>
<td>f6-fb/246-251</td>
<td>reserved</td>
</tr>
<tr>
<td></td>
<td></td>
<td>fc/252</td>
<td>ISDN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>fd/253</td>
<td>G.703</td>
</tr>
<tr>
<td></td>
<td></td>
<td>fe/254</td>
<td>T1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ff/255</td>
<td>none</td>
</tr>
<tr>
<td>unsigned long</td>
<td>last_ram_cpm</td>
<td></td>
<td>the value of this variable plus one yields the CPM memory size (in bytes)</td>
</tr>
</tbody>
</table>

(unit_config continued on next page)

† If (unit_config.floppy_exists_mask & value) == value, the drive is present.
For example, if (unit_config.floppy_exists_mask & 2) == 2, floppy drive 2 is present.
### Table 66-1 (continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned long</td>
<td>self_test_errors</td>
<td>(mask)</td>
<td>self-test errors encountered during power-up‡</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>CPM DRAM error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>CPM 32-bit counter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>CPM System Timing Controller (9513a)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>CPM DMAC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10/16</td>
<td>MPM0 DRAM (tested from CPM-global bus)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20/32</td>
<td>MPM0 DRAM (tested from MPM0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>40/64</td>
<td>MPM0 interrupt latch</td>
</tr>
<tr>
<td></td>
<td></td>
<td>80/128</td>
<td>unused</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100/256</td>
<td>MPM1 DRAM (tested from CPM-global bus)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>200/512</td>
<td>MPM1 DRAM (tested from MPM1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>400/1024</td>
<td>MPM1 interrupt latch</td>
</tr>
<tr>
<td></td>
<td></td>
<td>600/2048</td>
<td>unused</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1000/4096</td>
<td>MPM3 DRAM (tested from CPM-global bus)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2000/8192</td>
<td>MPM3 DRAM (tested from MPM3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4000/16384</td>
<td>MPM3 interrupt latch</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8000/32768</td>
<td>unused</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10000/65536</td>
<td>unused</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20000/131072</td>
<td>unused</td>
</tr>
<tr>
<td></td>
<td></td>
<td>40000/262144</td>
<td>unused</td>
</tr>
<tr>
<td></td>
<td></td>
<td>80000/524288</td>
<td>unused</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100000/1048576</td>
<td>unused</td>
</tr>
<tr>
<td></td>
<td></td>
<td>200000/2097152</td>
<td>unused</td>
</tr>
<tr>
<td></td>
<td></td>
<td>400000/4194304</td>
<td>unused</td>
</tr>
<tr>
<td></td>
<td></td>
<td>800000/8388608</td>
<td>unused</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1000000/16777216</td>
<td>unused</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2000000/33554432</td>
<td>unused</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4000000/67108864</td>
<td>unused</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8000000/134217728</td>
<td>unused</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10000000/268435456</td>
<td>unused</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20000000/536870912</td>
<td>unused</td>
</tr>
<tr>
<td></td>
<td></td>
<td>40000000/1073741824</td>
<td>unused</td>
</tr>
<tr>
<td></td>
<td></td>
<td>80000000/2147483648</td>
<td>unused</td>
</tr>
<tr>
<td>unsigned long</td>
<td>version</td>
<td>6</td>
<td>current value for this version of unit_config structure</td>
</tr>
<tr>
<td>unsigned long</td>
<td>model_number</td>
<td>19c8/6600</td>
<td>INTERVIEW 6600</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1a90/6800</td>
<td>INTERVIEW 6800 TURBO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1b58/7000</td>
<td>INTERVIEW 7000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1c20/7200</td>
<td>INTERVIEW 7200 TURBO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1d4c/7500</td>
<td>INTERVIEW 7500</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1e14/7700</td>
<td>INTERVIEW 7700 TURBO</td>
</tr>
</tbody>
</table>

(\textit{unit\_config} continued on next page)

‡ If \((\text{unit\_config.self\_test\_errors} \& \text{mask}) == \text{mask}\), the error is present.
If \((\text{unit\_config.self\_test\_errors} \& 0xffffffff) == 0\), no errors encountered during power-up.
Table 66-1 (continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned char</td>
<td>feb_type</td>
<td>0</td>
<td>original version</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>version with increased speed of software and faster access to ticks from FEB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>version which supports high-speed RAM recording, specifically aggregate T1 or G.703 data capture</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>version which also supports INTERVIEW 7200 and 7700 TURBOs</td>
</tr>
<tr>
<td>unsigned char</td>
<td>is_turbo</td>
<td>0</td>
<td>unit is not TURBO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>unit it TURBO</td>
</tr>
<tr>
<td>unsigned char</td>
<td>xdram_rev_num</td>
<td>0</td>
<td>XDRAM revision number</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>XDRAM board is present</td>
</tr>
<tr>
<td>unsigned long</td>
<td>xdram_lo_addr</td>
<td></td>
<td>low end of memory range</td>
</tr>
<tr>
<td>unsigned long</td>
<td>xdram_hi_addr</td>
<td></td>
<td>high end of memory range</td>
</tr>
<tr>
<td>unsigned long</td>
<td>spare1</td>
<td></td>
<td>reserved/undefined</td>
</tr>
<tr>
<td>unsigned long</td>
<td>spare2</td>
<td></td>
<td>reserved/undefined</td>
</tr>
<tr>
<td>unsigned long</td>
<td>spare3</td>
<td></td>
<td>reserved/undefined</td>
</tr>
<tr>
<td>unsigned long</td>
<td>spare4</td>
<td></td>
<td>reserved/undefined</td>
</tr>
<tr>
<td>unsigned long</td>
<td>spare5</td>
<td></td>
<td>reserved/undefined</td>
</tr>
<tr>
<td>unsigned long</td>
<td>spare6</td>
<td></td>
<td>reserved/undefined</td>
</tr>
<tr>
<td>unsigned long</td>
<td>spare7</td>
<td></td>
<td>reserved/undefined</td>
</tr>
<tr>
<td>unsigned long</td>
<td>spare8</td>
<td></td>
<td>reserved/undefined</td>
</tr>
<tr>
<td>unsigned long</td>
<td>spare9</td>
<td></td>
<td>reserved/undefined</td>
</tr>
<tr>
<td>unsigned long</td>
<td>sw_version</td>
<td></td>
<td>software version‡‡‡</td>
</tr>
<tr>
<td>unsigned long</td>
<td>fw_version</td>
<td></td>
<td>firmware version‡‡‡</td>
</tr>
</tbody>
</table>

Structure Name: mpm_info

Structure containing information on specific MPM board. Instance of this structure for each MPM board is contained in array named unit_config.mpm. Declared as type extern struct.

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned char</td>
<td>rev_num</td>
<td></td>
<td>MPM revision number</td>
</tr>
<tr>
<td>unsigned char</td>
<td>present</td>
<td>0</td>
<td>specific MPM board (of four) not present</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>specific MPM board (of four) present</td>
</tr>
<tr>
<td>unsigned long</td>
<td>lo_addr</td>
<td></td>
<td>low end of memory range</td>
</tr>
<tr>
<td>unsigned long</td>
<td>hi_addr</td>
<td></td>
<td>high end of memory range</td>
</tr>
</tbody>
</table>

‡‡‡To display the software version in the same format presented on the main menu screen, 5.00 for example, use the following format in a call to display (or tracef):

display("%lu.%02lu%c", ((unit_config.sw_version >> 8)/100), ((unit_config.sw_version >> 8)%100), (char)(unit_config.sw_version & 0xff));

The same format may be used for presentation of the firmware version.
67 Remote Port I/O

The REMOTE RS-232 port is a "spare" serial interface through which the programmer may communicate with other equipment. The remote port is located at the rear of the INTERVIEW next to the printer port. (The REMOTE LED on the front panel of the INTERVIEW is related to remote control of the unit, unimplemented at this time.)

Remote-port functions must be coded in C regions on the Protocol Spreadsheet. There are no spreadsheet-token equivalents of the C variables and routines described in this section. Use these variables and routines in either emulate or monitor mode to transmit and receive data through the remote port.

The remote-communications process on the CPM controls the flow of data between the user's program and the remote port. When data is received through the remote port, this process temporarily buffers it in a 2048-byte input queue. The user’s program makes requests for data from the input queue via the \texttt{rmt_getc}, \texttt{rmt_getl}, and \texttt{rmt_gets} input routines discussed below. When the remote-communications process receives a request, it removes data from the queue and passes it to the task. If there are no outstanding requests at the time data is received, it is discarded from the input queue—i.e., data received between requests cannot be retrieved. This is the default condition of the input queue.

To "lock" all received characters in the input queue, call \texttt{rmt_lock}. When the input queue is locked, the remote-communications process removes data only when 1) a user task has requested data via the \texttt{rmt_getc}, \texttt{rmt_getl}, or \texttt{rmt_gets} routine, 2) the input queue is full and some data must be discarded in order for incoming data to be buffered, or 3) \texttt{rmt_flushi} is executed. “Unlock” the input queue with \texttt{rmt_unlock}. \texttt{rmt_unlock}, \texttt{rmt_flushi}, and \texttt{rmt_flusho} are automatically executed whenever the INTERVIEW returns to Program mode.

\begin{quote}
\textbf{NOTE:} Although requests to receive (or transmit) data from more than one task will be queued by the remote-communications process, a single task can have only one such request outstanding at a time.
\end{quote}

Similarly, when the programmer wants to send data out the remote port, he calls \texttt{rmt_putc}, \texttt{rmt_puts}, or \texttt{rmt_putb}. The remote-communications process temporarily places these requests in an output queue before transmitting them through the remote port.

67.1 Structures

There are no structures associated exclusively with remote functions.
67.2 Variables

Table 67-1 lists the event variables specific to remote port I/O operations. Use most of these variables to detect changes in the status of the input and output queues.

As data is received through the remote port, the remote-communications process temporarily stores it in the input queue. Use `rmt_input_not_empty`, `rmt_input_almost_full`, and `rmt_input_overflow` to monitor how full the input queue is. When the input queue is "almost full," incoming data must be stopped in order to prevent the queue from overflowing.

`rmt_input_almost_empty` and `rmt_input_empty` are significant events as the remote communications process takes data out of the input queue. These events indicate that the input queue is ready to accept more data.
<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern event</td>
<td>rmt_break</td>
<td></td>
<td>True when a break (NULL with a framing error) is received through the remote port. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern event</td>
<td>rmt_input_not_empty</td>
<td></td>
<td>True when remote input-queue transitions from empty to not empty. Beginning to receive characters. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern event</td>
<td>rmt_input_almost_full</td>
<td></td>
<td>True when the remote input-queue transitions from less than 3/4 full to 3/4 full as data is being put into the queue. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern event</td>
<td>rmt_input_overflow</td>
<td></td>
<td>True when remote input-queue transitions from not full to full. At this point, the oldest existing data in the queue is discarded to make room for new data coming in the remote port. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern event</td>
<td>rmt_input_almost_empty</td>
<td></td>
<td>True when the remote input-queue transitions from more than 1/4 full to 1/4 full as data is being taken out of the queue. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern event</td>
<td>rmt_input_empty</td>
<td></td>
<td>True when remote input-queue transitions from not empty to empty. All characters have been read or discarded. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern event</td>
<td>rmt_output_empty</td>
<td></td>
<td>True when remote output-queue transitions from not empty to empty. All data output to the remote port has been transmitted. Line Setup configured for emulate or monitor mode.</td>
</tr>
</tbody>
</table>
67.3 Routines

Remote routines fall into three categories. *Input* routines are used to read data received from the remote port. Use *output* routines to transmit data through the remote port. The last category of routines reads or sets *parameters* for the remote port.

(A) Input Routines

Use *rmt_getc, rmt_getl, and rmt_gets* to read data received through the remote port. Use *rmt_lock* and *rmt_unlock* to control the flow of data from the input queue.

**rmt_getc**

**Synopsis**

```c
extern int rmt_getc(wait);
int wait;
```

**Description**

The *rmt_getc* routine reads the next character (if present) from the remote port.

**Inputs**

If no character is available from the input queue when *rmt_getc* is called, this parameter determines when the routine will return. A non-zero value for this parameter means wait for a character to become available before returning. If another task has already requested data from the queue, this request will be queued.

When the value is zero, the routine will return without a character if none is available. If there is already an outstanding request from another task, a zero value also causes the remote-communications process to return from the routine without checking the input queue.

**NOTE:** More than one test (task) may request data from the input queue. The remote-communications processes queues these requests as they are made. To ensure that requests are processed in turn, use this “wait” parameter consistently across tests. If you set the parameter in a call to *rmt_getc* (or *rmt_gets*) in one test, do the same in all tests.
Returns

If a character is present in the input queue, this routine returns the character (as an int) read. If no character is present and the routine's "wait" parameter is zero, a -1 will be returned. When the parameter is zero, a -1 also will be returned if there is already an outstanding request from another task.

Example

In the following example, the routine will not wait for a character to become available in the remote port before returning. Each time the ⌘ key is pressed, the next character, if present, will be displayed. If a -1 is returned instead of a character, a message to that effect will be displayed on the prompt line.

```
LAYER: 1
STATE: get_next_character
CONDITIONS: ENTER_STATE
ACTIONS:
{
    display_prompt("Press C to get next character.
    rmt_lock();
}
CONDITIONS: KEYBOARD "cC"
ACTIONS:
{
    int character;
    character = rmt_getc(0);
    if(character == -1)
        display_prompt("No character available.");
    else
        displayf("%c", character);
}
```

rmt_getl

Synopsis

```
extern int rmt_getl(string_ptr, max_length);
char * string_ptr;
int max_length;
```

Description

rmt_getl reads from the remote port one line at a time. This routine gets at the most the specified number of characters from the remote port and puts them in an array. Unless a carriage return or linefeed is encountered, the routine will not return until the specified number of characters has been read. A carriage return or linefeed causes the routine to return, even if the specified number of characters has not yet been read. The carriage return or linefeed will be replaced by a terminating NULL character in the array.
Inputs

The first parameter is a pointer to the array into which the characters will be put.

The second parameter is the maximum number of characters to be read.

Returns

This routine returns the number of characters (preceding the terminating NULL) read into the array.

Example

Each time the [u] key is pressed, twenty characters, at the most, will be read from the remote port, put into an array called data, and displayed on the screen.

```
LAYER: 1
STATE: read_line
CONDITIONS: ENTER_STATE
ACTIONS:
{
    display_prompt("Press L to get next line.
    rmt_lock();
}
CONDITIONS: KEYBOARD "IL"
ACTIONS:
{
    int number;
    unsigned char data [25];
    number = rmt_get(data, 20);
    display("\n\n%u characters read: %20s\n", number, data);
}
```

rmt_gets

Synopsis

```
extern int rmt_gets(string_ptr, length, wait);
char * string_ptr;
int length;
int wait;
```

Description

Similar to rmt_getl, this routine gets a specified number of characters from the remote port and puts them in an array. Unlike rmt_getl, characters continue to be read even if a carriage return or linefeed is encountered. The array is not NULL-terminated.
Inputs

The first parameter is a pointer to the array into which the characters will be put.

The second parameter is the number of characters to be read.

If the specified number of characters is not available from the input queue when \textit{rmt_get} is called, the third parameter determines when the routine will return. A non-zero value for this parameter means wait for the specified number of characters to become available before returning. If another task has already requested data from the queue, this request will be queued.

When the value is zero, the routine will return with less than the specified number of characters if all are not available. If there is already an outstanding request from another task, a zero value also causes the remote-communications process to return from the routine without checking the input queue.

\textbf{NOTE:} More than one test (task) may request data from the input queue. The remote-communications processes queues these requests as they are made. To ensure that requests are processed in turn, use this “wait” parameter consistently across tests. If you set the parameter in a call to \textit{rmt_gets} (or \textit{rmt_getc}) in one test, do the same in all tests.

Returns

This routine returns the number of characters read from the remote port.

Example

When the $\text{\$}$ key is pressed, 4000 characters will be read from the remote port, put into an array called \textit{data}, displayed on the screen (until a NULL is encountered—see \%s in \textit{tracef} routine, Section 61), and written to a file named \textit{echo_time}. This is the program that might be run to receive the file transmitted in the \textit{rmt_pub} example.

```c
#define FILE_LENGTH 4000
#define FILENAME "FD1/usr/echo_time"
#include <stdio.h>
#include <trace.h>
extern struct trace_buf \textit{11_trbuf};
FILE * \textit{stream_ptr};
size_t \textit{n};
unsigned char \textit{data}[FILE_LENGTH];
int \textit{count};
```
LAYER: 1
STATE: get_string
CONDITIONS: ENTER_STATE
ACTIONS:
{
    rmt_lock();
    if((stream_ptr = fopen(FILENAME, "w")) == 0)
        display_prompt("Cannot open file.");
    else
    {
        display_prompt("Press S to read string.");
        pos_cursor(1,0);
    }
}
CONDITIONS: KEYBOARD "sS"
ACTIONS:
{
    count = rmt_gets(data, FILE_LENGTH, 1);
    if(count != FILE_LENGTH)
        displayf("Could not read entire string.\n");
    tracef(&ml_trbuf, "%d characters read: \n%s\n", count, data);
    n = fwrite(data, 1, FILE_LENGTH, stream_ptr);
    if(n != FILE_LENGTH)
        displayf("A write error has occurred.\n");
    else
        displayf("File written. \n");
    if(fclose(stream_ptr) != 0)
        displayf("Either file is already closed, or close cannot be executed. \n");
    else
        displayf("File closed. \n");
}

rmt_flushi

Synopsis
extern int rmt_flushi();

Description
If characters have been received in the input queue, but have not been read yet, this routine causes them to be discarded. Whenever the INTERVIEW enters or leaves Run mode, rmt_flushi is automatically executed. This ensures that the input queue is empty.

NOTE: A call to any of the routines which set the parameters of the remote port also causes rmt_flushi to be executed automatically. The routines which only get the current parameters of the remote port have no effect on the input queue.

When the programmer calls rmt_flushi, requests for data from the input queue will be processed before the input queue is flushed. When a call to rmt_flushi is made from another test, however, input routines waiting for characters from the input queue will be returned.
Returns

$rmt\_flushi$ returns a zero when the input queue is flushed successfully. Otherwise, it returns a non-zero value.

Example

This example is the same as that for $rmt\_getc$. Notice that as the program enters the first state, the input queue is flushed.

LAYER: 1
STATE: get_next_character
CONDITIONS: ENTER_STATE
ACTIONS:

```c
{  
  display_prompt("Press C to get next character.
  rmt_lock();
  rmt_flushi();
}
```
CONDITIONS: KEYBOARD *cC*
ACTIONS:

```c
{  
  int character;
  character = rmt_getc(0);
  if(character == -1)
    display_prompt("No character available. ");
  else
    display("%c", character);
}
```

$rmt\_lock$

Synopsis

extern void rmt_lock();

Description

Recall that in its default state, the input queue does not retain characters received through the remote port between requests from user tasks. Data in the queue must either be passed to a user task or be discarded. The $rmt\_lock$ routine "locks" all received characters in the input queue until they are requested. (Refer again to the beginning of this section.)

Example

The following example is the same as the one for the $rmt\_geti$ routine. Notice that a call to $rmt\_lock$ is made as the program begins. The operator makes a request for data from the input queue by pressing C. The next line of data in the input queue will be removed and put in the array named $data$. 

67-9
rmt_unlock

Synopsis

extern void rmt_unlock();

Description

The rmt_unlock routine implements the inverse of the rmt_lock routine. If characters are received in the remote port and there are no outstanding requests for data, the remote-communications process will discard the characters. (Refer also to rmt_lock and to the beginning of this section.)

rmt_unlock is automatically executed when the INTERVIEW returns to Program mode.

Example

In the following example, the input queue is locked as soon as the program begins. It will remain locked until the operator press (or )

```c
extern void rmt_unlock();

int number;
unsigned char data[25];
number = rmt_getl(data, 20);
displayf("n%u characters read:n%.20s", number, data);

rmt_unlock();
```
(B) Output Routines

Use the following routines to transmit data through the remote port.

rmt_putchar

Synopsis

extern int rmt_putchar(character, wait);
unsigned char character;
int wait;

Description

This routine sends a specified character to the output queue of the remote port for transmission.

Inputs

The first parameter is the character to be transmitted. It may be given as a hexadecimal, octal, or decimal constant; as an alphanumeric constant inside single quotes; or as a variable. A hexadecimal value must be preceded by the prefix 0x or 0X; an octal value must be preceded by the prefix 0. If no prefix appears before the input, the number is assumed to be decimal.

If space in the output queue is not available for the character when rmt_putchar is called, the second parameter determines when the routine will return. A non-zero value for this parameter means wait for space in the output queue to become available and return zero when the character is in the queue. If there is already a request from another task, this request will be queued.

When the value is zero and space in the output queue is not available, the routine will return -1. The character will not be in the queue. If another task is already waiting for access to the output queue, a zero value also causes the remote-communications process to return from the routine without checking for available space in the output queue.
NOTE: More than one test (task) may request to send data to the output queue. The remote-communications processes queues these requests as they are made. To ensure that requests to output data are processed in turn, use this "wait" parameter consistently across tests. If you set the parameter in a call to `rmt_putchar (rmt_puts or rmt_putb)` in one test, do the same in all tests.

Returns

If the character is successfully written to the output queue, the routine returns zero. If no space is available in the output queue and the routine's "wait" parameter is zero, a -1 will be returned. When the parameter is zero, a -1 also will be returned if another task is already waiting for access to the output queue.

Example

In the following example, the next character in a fox message will be sent to the output queue of the remote port each time the operator presses `C`. As a character is successfully queued, it will be displayed in the Display Window. If no space is available in the output queue for the character, -1 will be returned and a message to that effect will be displayed on the prompt line. No more characters will be sent.

```c
{  
    unsigned char data[] = "(FOX)\x";
    unsigned char character;
    int i, length, error;
}

LAYER: 1
STATE: transmit_characters
CONDITIONS: ENTER_STATE
ACTIONS:
{
    display_prompt("Press C to transmit character.
    length = sizeof(data) - 1;
}
CONDITIONS: KEYBOARD "cC"
ACTIONS:
{
    for (i = 0; i < length; i++)
    {
        character = data[i];
        error = rmt_putchar(character, 0);
        if(error == -1)
            display_prompt("No space available in output queue.
        else
            display("%c", character);
    }
}
```
rmt_puts

Synopsis

extern int rmt_puts(string_ptr, wait);
const char *string_ptr;
int wait;

Description

This routine outputs a NULL-terminated string to the output queue of the remote port.

Inputs

The first parameter is a pointer to the string to be transmitted.

If space in the output queue is not available for the string when rmt_puts is called, the second parameter determines when the routine will return. A non-zero value for this parameter means wait for space in the output queue to become available and return when the string is in the queue. If there is already a request from another task, this request will be queued.

When the value is zero and space is not available in the output queue, the routine returns the number of characters, if any, put into the queue. If another task is already waiting for access to the output queue, a zero value also causes the remote-communications process to return from the routine without checking for available space in the output queue.

NOTE: More than one test (task) may request to send data to the output queue. The remote-communications processes queues these requests as they are made. To ensure that requests to output data are processed in turn, use this “wait” parameter consistently across tests. If you set the parameter in a call to rmt_puts (rmt_putc or rmt_putb) in one test, do the same in all tests.

Returns

This routine returns the number of characters put into the output queue.

Example

The following example is similar to the one given for rmt_putc. When the key is pressed, the fox message will be sent to the remote port. The difference is that the message will be output to the remote port as a string (rather than
character by character). If the output queue is full, the routine will not wait for space to become available before returning. The number of characters successfully queued will be displayed in the Display Window. If the number of characters queued is less than the length of the string, a message to that effect will be displayed on the prompt line.

```c
{ unsigned char data [] = "((FOX)\x00";
  int count, length;
}
```

**LAYER: 1**

**STATE:** transmit_string

**CONDITIONS:** ENTER_STATE

**ACTIONS:**

```c
{ display_prompt("Press S to transmit string.
  length = sizeof(data) - 1;
}
```

**CONDITIONS:** KEYBOARD "sS"

**ACTIONS:**

```c
{ count = rmt_puts(data, 0);
  if(count != length)
    display_prompt("Could not output entire string.
    pos_cursor(1,0);
    displayf("%d characters transmitted.", count);
}
```

**rmt_putb**

**Synopsis**

```c
extern int rmt_putb(string_ptr, length, wait);
const char * string_ptr;
int length;
int wait;
```

**Description**

This routine sends a string of specified length to the output queue of the remote port.

**Inputs**

The first parameter indicates the string to be output.

The second parameter is the length of the string to be output.

If space in the output queue is not available for the string when `rmt_putb` is called, the third parameter determines when the routine will return. A non-zero value for this parameter means wait for space in the output queue to become
available and return when all characters in the string have been queued. If another task is already waiting for access to the output queue, this request will be queued.

When the value is zero and space is not available in the output queue, the routine returns the number of characters, if any, put into the queue. If there is already an outstanding request from another task, a zero value also causes the remote-communications process to return from the routine without checking for available space in the output queue.

NOTE: More than one test (task) may request to send data to the output queue. The remote-communications processes queues these requests as they are made. To ensure that requests to output data are processed in turn, use this "wait" parameter consistently across tests. If you set the parameter in a call to rmt_putchar (rmt_putc or rmt_puts) in one test, do the same in all tests.

Returns
This routine returns the number of characters put into the output queue.

Example
This is the program that might be run to transmit the file received in the rmt_gets example. The user specifies the filename and its size (shown in the directory listing on the File Maintenance screen) in the two #define preprocessor directives at the beginning of the program. When the program begins, the contents of the file named echo_time will be read into an array called data. When the operator presses the  key, the contents of the array will be transmitted and displayed.

```c
#include <stdio.h>  
#include <trace_buf.h>  
extern struct trace_buf ll_trbuf;  
FILE * stream_ptr;  
unsigned char data [FILE_LENGTH];  
unsigned char size [FILE_LENGTH+100];  
int count;
```
else
{
    pos_cursor(1,0);
    n = fread(data, 1, FILE_LENGTH, stream_ptr);
    if(n != FILE_LENGTH)
        displayf("Either a read error has occurred, or an EOF has been
        encountered. \n");
    if(fclose(stream_ptr) != 0)
        displayf("Either file is already closed, or close cannot be executed. \n");
    else
        displayf("File closed.\n");
    if(n == FILE_LENGTH)
        display_prompt("Press T to transmit characters.");
}

CONDITIONS: KEYBOARD *tT*
ACTIONS:
{
    count = rmt_putchar(data, FILE_LENGTH, 1);
    if(count != FILE_LENGTH)
        displayf("Could not output entire string.\n");
    sprintf(size, "%d characters transmitted: %%.%dH", count, count);
    tracef(&l_trbuf, size, data);
    tracef(&l_trbuf, "\n\n");
}

rmt_flusho

Synopsis

extern int rmt_flusho();

Description

If characters are queued to be output from the remote port, but have not been transmitted yet, this routine causes them to be discarded. This ensures that anything previously in the output queue port will be deleted.

rmt_flusho is automatically executed when the INTERVIEW returns to Program mode.

NOTE: A call to any of the routines which set the parameters of the remote port causes rmt_flusho to be executed automatically. The routines which only get the current parameters of the remote port have no effect on the output queue.

Returns

rmt_flusho returns a zero when the output queue is flushed successfully. Otherwise, it returns a non-zero value.
Example

This example is the same as that for \texttt{rmt\_putc}. Notice that as the program enters the first state, the output queue is flushed.

\begin{verbatim}
{
  unsigned char data[] = "©FOX";
  unsigned char character
  int i, length, error;
}

LAYER: 1
STATE: transmit_a_character
CONDITIONS: ENTER_STATE
ACTIONS:
{
  rmt\_flusho();
  display\_prompt("Press C to transmit character.
  length = sizeo!(data);
}
CONDITIONS: KEYBOARD "cC"
ACTIONS:
{
  for(i = 0; i < length; i++)
  {
    character = data[i];
    error = rmt\_putc(character, 0);
    if(error ==-1)
    {
      display\_prompt("No space available in output queue.
      break;
    }
    else
    displayf("%c", character);
  }
}
\end{verbatim}

\texttt{rmt\_suspendo}

Synopsis

\texttt{extern int rmt\_suspendo();}

Description

If characters are queued to be output from the remote port, but have not been transmitted yet, this routine causes transmitting to be suspended. The output queue will not be flushed. Use this routine only when the remote port handshaking mode is full-duplex without flow control.

Returns

\texttt{rmt\_suspendo} returns a zero when transmitting is successfully suspended. Otherwise, it returns a non-zero value.
Example

When the INTERVIEW receives an X-OFF as a signal to stop sending data, it will suspend transmissions from the remote port.

```c
extern event rmt_input_not_empty;
int character;
}

LAYER: 1
STATE: suspend_output
CONDITIONS: ENTER_STATE
ACTIONS:
{
    rmt_lock();
}
CONDITIONS:
{
    rmt_input_not_empty
}
ACTIONS:
{
    character = rmt_getc(1);
    if(character == 0x13)
        rmt_suspendo();
}
TIMEOUT ck_input RESTART 0.001
CONDITIONS: TIMEOUT ck_input
ACTIONS:
{
    character = rmt_getc(1);
    if(character == 0x13)
        rmt_suspendo();
}
TIMEOUT ck_input RESTART 0.001
```

rmt_resumeo

**Synopsis**

`extern int rmt_resumeo();`

**Description**

This routine resumes transmission of characters from the remote port. Use this routine only when the remote port handshaking mode is full-duplex without flow control.

**Returns**

`rmt_resumeo` returns a zero when transmitting is successfully resumed. Otherwise, it returns a non-zero value.
Example

When the INTERVIEW receives an X-ON as a signal to send data, it will resume transmissions from the remote port.

```c
{    
    int character;
}
```

**Layer: 1**

**State:** resume_output

**Conditions:** ENTER_STATE

**Actions:**

```c
rcm_lock();
```

**Timeout Restart:** ck_input 0.001

**Conditions:** TIMEOUT check_input

**Actions:**

```c
character = rmt_getc(1);
if(character == 0x11)
    rmt_resume();
```

**Timeout Restart:** ck_input 0.001

**rmt_send_break**

**Synopsis**

```c
extern int rmt_send_break(wait);
int wait;
```

**Description**

This routine causes a break, queued as other transmits, to be transmitted.

**Inputs**

If space in the output queue is not available for the break when *rmt_send_break* is called, the only parameter determines when the routine will return. A non-zero value for this parameter means wait for space in the output queue to become available and return zero when the break is in the queue. If there is already a request from another task, this request will be queued.

When the value is zero and space in the output queue is not available, the routine will return -1. The break will not be in the queue. If another task is already waiting for access to the output queue, a zero value also causes the remote-communications process to return from the routine without checking for available space in the output queue.
NOTE: More than one test (task) may request to send data to the output queue. The remote-communications processes queues these requests as they are made. To ensure that requests to output data are processed in turn, use this “wait” parameter consistently across tests. If you set the parameter in a call to `rmt_send_break (rmt_puts, rmt_puts or rmt_puts)` in one test, do the same in all tests.

Returns

If the break is successfully written to the output queue, the routine returns zero. If no space is available in the output queue and the routine’s “wait” parameter is zero, a −1 will be returned. When the parameter is zero, a −1 also will be returned if another task is already waiting for access to the output queue.

Example

In this example, a break will be transmitted each time the operator presses the space bar.

```
LAYER: 1
  STATE: transmit_break
  CONDITIONS: KEYBOARD
  ACTIONS:
    { rmt_send_break(1); }
```

(C) Configuration Routines

The default configuration for the remote port at boot-up is the following:

- Baud rate = 1200
- Bits/character = 8
- Parity = None
- Mode = Full-duplex

Use the first four routines discussed below to change these settings. The programmer’s reconfiguration of the remote port is not affected when the INTERVIEW exits or re-enters Run mode.

A call to any of these `set` routines causes `rmt_flushi` and `rmt_flusho` to be executed automatically before the parameter is set.

Use the remaining four routines to read the current parameter-settings for the remote port. These `get` routines have no effect on the input and output queues.
rmt_set_baud_rate

Synopsis

extern int rmt_set_baud_rate(speed);
int speed;

Description

This routine sets the baud rate for the remote port. The default value at boot-up is 1200.

NOTE: A call to rmt_set_baud_rate causes rmt_flushi and rmt_flusho to be executed automatically before the baud rate is set.

Inputs

The only parameter is the desired baud rate. Values that are multiples of 300 in the range 300 through 19200 are valid.

Returns

If the specified baud rate is valid and successfully set, zero is returned. If the baud rate is valid, but not successfully set, -1 is returned. For an invalid baud rate, the routine returns -2.

Example

In order for two devices to communicate with each other, they must be using the same baud rate. When they are not the same, some devices send a break as a signal for the other to adjust its baud rate. If the following example, the INTERVIEW will change the baud rate for the remote port whenever a break is received.

```c
{ extern event rmt_break;
  int error;
  int speed = 300;
}
LAYER: 1
STATE: adjust_baud_rate
  CONDITIONS:
  {
    rmt_break
  }
```
ACTIONS:
{
  error = rmt_set_baud_rate(speed);
  if(error != -1)
    {
      speed *= 2;
      if(speed > 19200)
        speed = 300;
    }
  else
    display("Unable to set the baud rate to \%d.", speed);
}

rmt_set_bits

Synopsis
extern int rmt_set_bits(value);
int value;

Description
This routine sets the number of bits per character for the remote port. The default setting at boot-up is 8 bits/character.

NOTE: A call to rmt_set_bits causes rmt_flushi and rmt_flusho to be executed automatically before the number of bits/character is set.

Inputs
The only parameter is the number of bits/character. Valid values are five through eight.

Returns
If the specified number of bits/character is valid and successfully set, zero is returned. If the number is valid, but not successfully set, -1 is returned. For an invalid value, the routine returns -2.

Example
In this example, the number of bits/character for the remote port is set to 7 and displayed on the Display Window screen.

LAYER: 1
  STATE: set_parameters
  CONDITIONS: ENTER_STATE
  ACTIONS:
  {
    display("Bits = \%d\", rmt_set_bits(7));
  }
rmt_set_parity

Synopsis

extern int rmt_set_parity(parity);
int parity;

Description

This routine sets the parity for the remote port. The default setting at boot-up is no parity.

NOTE: A call to rmt_set_parity causes rmt_flushi and rmt_flusho to be executed automatically before the parity for the remote port is set.

Inputs

The only parameter is a value designating the desired parity. Valid values are the following: none (0), odd (1), even (2), mark (3), or space (4).

Returns

If the specified parity value is valid and successfully set, zero is returned. If the value is valid, but not successfully set, -1 is returned. For an invalid parity value, the routine returns -2.

Example

In this example, the number of bits/character for the remote port is set to 7 and parity is even. Both settings are displayed on the Display Window screen.

LAYER: 1
STATE: set_parameters
CONDITIONS: ENTER_STATE
ACTIONS:
{
    displayf("Bits = %d Parity = %d ", rmt_set_bits(7), rmt_set_parity(2));
}
rmt_set_mode

Synopsis

extern int rmt_set_mode(mode);
int mode;

Description

This routine sets the handshaking mode for the remote port. The default setting at boot-up is FDX with no flow control.

NOTE: A call to rmt_set_mode causes rmt_flushi and rmt_flusho to be executed automatically before the mode for the remote port is set.

Inputs

The only parameter is a value designating the mode. Valid values are the following:

0 = Full-duplex with no flow control (FDX)
1 = Half-duplex (HDX)
2 = Full-duplex with X-ON/X-OFF characters for flow control
3 = Full-duplex with DTR and CTS EIA leads for flow control. Use a special null-modem cable for direct connections. See Figure 67-1.

Returns

If the specified mode value is valid and successfully set, zero is returned. If the value is valid, but not successfully set, -1 is returned. For an invalid mode value, the routine returns -2.
Example

In this example, the number of bits/character for the remote port is set to 7, parity is even, and the mode is set for FDX with X-ON/X-OFF. All three settings are displayed on the Display Window screen.

```
LAYER: 1
STATE: set_parameters
    CONDITIONS: ENTER_STATE
    ACTIONS:
    {
        display("Bits = %d Parity = %d Mode = %d ", rmt_set_bits(7),
            rmt_set_parity(2), rmt_set_mode(2));
    }
```

**rmt_get_baud_rate**

**Synopsis**

`extern int rmt_get_baud_rate();`

**Description**

This routine gets the current baud-rate setting for the remote port.

**Returns**

The baud rate for the remote port is returned.

**Example**

As the program begins, the current baud-rate setting for the remote port is displayed on the Display Window screen.

```
LAYER: 1
STATE: baud_rate
    CONDITIONS: ENTER_STATE
    ACTIONS:
    {
        display("Baud = %d ", rmt_get_baud_rate());
    }
```

**rmt_get_bits**

**Synopsis**

`extern int rmt_get_bits();`

**Description**

This routine tells how many bits there are per character. Possible values are five through eight.

**Returns**

The current number of bits per character for the remote port is returned.
Example

In this example, the current baud-rate setting and the number of bits/character for the remote port are displayed on the Display Window screen.

```c
STATE: current_parameters
CONDITIONS: ENTER_STATE
ACTIONS:
{
    display("Baud = %d Bits = %d Parity = %d ", rmt_get_baud_rate(), rmt_get_bits(), rmt_get parity());
}
```

**rmt_get_parity**

**Synopsis**

```c
extern int rmt_get_parity();
```

**Description**

This routine gets the current parity setting for the remote port.

**Returns**

The current number of bits per character for the remote port is returned.

Example

In this example, the current baud-rate setting, number of bits/character, and the parity for the remote port are displayed on the Display Window screen.

```c
STATE: current_parameters
CONDITIONS: ENTER_STATE
ACTIONS:
{
    display("Baud = %d Bits = %d Parity = %d ", rmt_get_baud_rate(), rmt_get_bits(), rmt_get_parity());
}
```

**rmt_get_mode**

**Synopsis**

```c
extern int rmt_get_mode();
```

**Description**

This routine gets the current handshaking mode for the remote port.
Returns

The current handshaking mode for the remote port is returned:

0 = Full-duplex with no flow control (FDX)
1 = Half-duplex (HDX)
2 = Full-duplex with X-ON/X-OFF characters for flow control
3 = Full-duplex with DTR and CTS EIA leads for flow control Requires
    a special null-modem cable for INTERVIEW-to-INTERVIEW direct
    connections. Refer to Figure 67-1.

Example

In this example, the current baud-rate setting, number of bits/character, parity,
and handshaking mode for the remote port are displayed on the Display Window
screen.

LAYER: 1
STATE: current_parameters
CONDITIONS: ENTER_STATE
ACTIONS:
{
    display("Baud = %d Bits = %d Parity = %d Mode = %d ", rmt_get_baud_rate(),
            rnt_get_bits(), rmt_get_parity(), rmt_get_mode());
}
### Transmitter's AUX Port Lead Configuration

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pin Number</td>
<td>16</td>
<td>14</td>
<td>12</td>
<td>10</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>15</td>
<td>13</td>
<td>11</td>
<td>9</td>
<td>7</td>
<td>5</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Bit will be used for</td>
<td>C</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>C</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
</tbody>
</table>

### Receiver's AUX Port Lead Configuration

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pin Number</td>
<td>16</td>
<td>14</td>
<td>12</td>
<td>10</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>15</td>
<td>13</td>
<td>11</td>
<td>9</td>
<td>7</td>
<td>5</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

| O | Output/Non-control |
| I | Input/Non-control   |
| I/C | Input/Control       |
| C | Control             |
| D | Data                |
| U | Unassigned          |

Figure 68-1 Sample AUX port lead configurations for two INTERVIEWs connected by their AUX interfaces. Assume one-way data transmission (i.e., one device is controlling the other).
68 AUX Port I/O

The Auxiliary (AUX) port is a "spare" interface through which the programmer may communicate with other lab equipment. The AUX port is located at the rear of the INTERVIEW, between the printer and RGB connectors. It is controlled by a Zilog CIO (Counter/Timer, Parallel Input/Output Unit) chip. The AUX port may be used as a serial or parallel interface. When it is operated as a parallel port, up to sixteen bits (one bit on each of sixteen leads) may be transmitted simultaneously.

AUX-port control must be coded in C regions on the Protocol Spreadsheet. There are no spreadsheet-token equivalents of the C variables and routines described in this section.

A normal configuration of equipment using the AUX port will involve two INTERVIEWs with AUX port setups that mirror each other to some extent, as in Figure 68-1. The transmitting INTERVIEW will use one of its output leads as a "strobe" to signal to the receiving INTERVIEW that an AUX word is available to be read. The receiver will detect this strobe as an aux_change event.

The receiving INTERVIEW will use one of its output leads to acknowledge each AUX word received. The transmitting INTERVIEW will detect this acknowledgment as an aux_change event.

NOTE: The AUX port is not controlled by the same CPU that handles the user program. The need for interprocessor communication without data buffering makes rapid, successive transmissions difficult to handle. It is recommended, therefore, that control bits be set aside for flow control—a bit set by the transmitter as input/control is set by the receiver as output/non-control, and vice versa—and that every output word be acknowledged before a succeeding word is output.

68.1 Variables

Table 68-1 lists the variables specific to AUX I/O operations. The fast-event variable, aux_change, detects a change in a lead that has been configured as a control lead. Any or all of the sixteen leads in the interface may be designated control leads. Section 68.2 explains how to configure control leads.
aux_change does not establish which control lead(s) has changed. Two associated
variables, curr_aux_value and prev_aux_value, indicate the status of all sixteen leads.
These are two-byte (short) variables. Each lead is represented by a different bit in
the short. If the bit-value of a given lead is zero, the lead is on. If the bit-value is
one, the lead is off.

Whenever a control lead changes, the value in curr_aux_value is written to
prev_aux_value. Then curr_aux_value is updated.

Table 68-1
AUX Port I/O Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern fast_event</td>
<td>aux_change</td>
<td>True when the status of a lead designated as control (and input) changes. Is automatically made to come true by the CIO chip as soon as leads have been configured via set_aux_direction and set_aux_ctl_leads routines. Therefore, condition must be tested again in a different state. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned short</td>
<td>curr_aux_value</td>
<td>Each bit designates a different lead. A bit-value of one indicates a given lead is on. When value of curr_aux_value is exclusive or'd (') with prev_aux_value, result indicates those leads whose status has changed. Updated when aux_change comes true. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned short</td>
<td>prev_aux_value</td>
<td>Value of previous curr_aux_value. Updated when control leads change, but only after logic has had a chance to compare current and previous leads. Line Setup configured for emulate or monitor mode.</td>
</tr>
</tbody>
</table>
68.2 Routines

In the examples for the following routines, assume that two INTERVIEW's are connected and that data flows in one direction.

CAUTION: You may damage the AUX interface if the same lead is designated as output on both units. We suggest that you set the leads on each unit as input/output and control/non-control before you connect the AUX interfaces. See Figure 68-1.

set_aux_direction

Synopsis

extern void set_aux_direction(input_or_output);
unsigned short input_or_output;

Description

This routine designates leads on the AUX port as input or output. Designated output leads for the transmitter are set as input leads by the receiver.

Inputs

The only input is a sixteen-bit variable. Each bit in the variable designates one lead and may be set to zero (output) or one (input).

Example

Both sides of the connection may be transmitter or receiver. But for simplification in examples, let's designate only one side as the transmitter and the other as the receiver. In this example, the transmitter sets all 8 bits of the low-order byte as output bits for data, the low-order bit of the high byte as input (for handshaking), the next 6 bits of the high byte as input (unused), and the high-order bit as output (the receiver will designate this bit as input for handshaking).

LAYER: 1
STATE: set_input_leads
CONDITIONS: ENTER_STATE
ACTIONS:

set_aux_direction(0x7f00);
The other (receiver) INTERVIEW sets a bit as input (for handshaking). It must be one that was designated as output by the transmitter, the highest-order bit of the high byte. The data bits set as output by the transmitter must be set as input by the receiver. The receiver's `set_aux_direction` routine would look like this:

```
LAYER: 1
    STATE: set_input_leads
        CONDITIONS: ENTER_STATE
        ACTIONS:
            {
                set_aux_direction(0xff);
            }
```

`set_aux_ctl_leads`

**Synopsis**

```c
extern void set_aux_ctl_leads(ctl_or_not);
unsigned short ctl_or_not;
```

**Description**

This routine determines whether or not leads will be control leads. Control leads must also be input leads, but input leads do not necessarily have to be control leads. Output leads can never be control leads.

**Inputs**

The only input is a sixteen-bit variable. Each bit in the variable designates one lead and may be set to zero (non-control) or one (control).

**Example**

Assuming the input/output bits set in the previous example, the transmitter sets all 8 data bits (output) as non-control, the low-order input bit of the high byte as control (for handshaking), the next 6 input bits of the high byte as non-control (unused), and the high-order output bit as non-control (the receiver will designate this bit as control for handshaking).
The "receiver" INTERVIEW sets one input bit as control for handshaking purposes. It must be one that was designated as output by the transmitter, the highest-order bit of the high byte. The receiver's set_aux_ctl_leads routine would look like this:

**write_aux**

**Synopsis**

extern void write_aux(output_word);
unsigned short output_word;

**Description**

This routine sends a combination of data, control, and (perhaps) unused bits as output. Input bits are not transmitted by the CIO.

**Inputs**

The only input is a sixteen-bit variable. Each bit designates one lead and may represent data or control information, or be unused. If a given lead was designated as a control lead, it is an input lead and the CIO will not transmit the status of the bit in any case, so its setting of 1 or 0 does not matter. If the lead was designated as a non-control lead, it might contain data, be unused, or contain an alternating value to indicate acknowledgment (if the other side designated it as a control lead).

**Example**

The transmitting INTERVIEW is going to send data to the receiving INTERVIEW. Before the next transmission can be sent, an acknowledgment must be received. The acknowledgment is detected by the fast-event variable aux_change.

**NOTE:** The CIO chip automatically generates a true aux_change condition when the set_aux_ctl_leads routine has been executed. The aux_change condition, therefore, should be placed in a separate programming state from the set_aux_ctl_leads routine.
The transmitter's program might look like this:

```c
LAYER: 1
{
    extern fast_event aux_change;
    extern volatile const unsigned short curr_aux_value;
    volatile unsigned short curr;
    unsigned short mask;
    unsigned char data;
    
    STATE: configure_leads
    CONDITIONS: ENTER_STATE
    ACTIONS:
    {
        set_aux_direction(0x7f00);
        set_aux_ctl_leads(0x0100);
        curr = curr_aux_value;
        data = 0x01;
        mask = curr ^ 0x8000;
        display_prompt("Connect cable. Press spacebar to transmit.
                        pos_cursor(1,0);"
    }
    NEXT_STATE: send_data

    STATE: send_data
    CONDITIONS: KEYBOARD * "
    ACTIONS:
    {
        if(data <= 10)
        {
            write_aux(mask | data);
            displayf("Transmission %d waiting for ACK.
                    \n", data);
        }
    }
    NEXT_STATE: waiting

    STATE: waiting
    CONDITIONS: {aux_change}
    ACTIONS:
    {
        data++;
        mask = (mask ^ 0x8000);
        displayf("ACK received: %04x Press spacebar to transmit.
                    \n", curr);
    }
    NEXT_STATE: send_data
    CONDITIONS: {data > 10}
    ACTIONS:
    {
        display_prompt("End of test.
                    ");
    }
```
The receiver's program would look like this:

```
LAYER: 1
{
    extern fast_event aux_change;
    extern volatile const unsigned short curr_aux_value;
    volatile unsigned short curr;
    unsigned short mask;
    int count;
}

STATE: configure_leads
CONDITIONS: ENTER_STATE
ACTIONS:
{
    set_aux_direction(0xeff);
    set_aux_ctl_leads(0x8000);
}

CONDITIONS: {aux_change}
ACTIONS:
{
    curr = curr_aux_value;
    count = 1;
    mask = curr & 0x0100;
    display_prompt("Connect cable. Ready to receive.
                   pos_cursor(1,0);
}

NEXT_STATE: receive_data

STATE: receive_data
CONDITIONS: {aux_change}
ACTIONS:
{
    display("Transmission %d received: %04x Press spacebar to send ACK.
            count, curr); \
    }

NEXT_STATE: send_ack
CONDITIONS: {count > 10}
ACTIONS:
{
    display_prompt("End of test.");
}

STATE: send_ack
CONDITIONS: KEYBOARD " "
ACTIONS:
{
    if(count <= 10)
    {
        write_aux(mask);
        count++;
        mask = (mask & 0x0100);
    }
}

NEXT_STATE: receive_data
```
NOTE: If you designate more than one lead as control, you might need to compare \textit{prev\_aux\_value} with \textit{curr\_aux\_value} to determine if the lead you are interested in is the one that changed. Here, since there is only one input-control lead on each side, the event \textit{aux\_change} is sufficient to signal and to acknowledge transmission. The value of \textit{prev\_aux\_value} does not have to be checked.

\texttt{set\_aux\_reg}

\textbf{Synopsis}

\texttt{extern void set\_aux\_reg(reg\_value\_word);}
\texttt{unsigned short reg\_value\_word;}

\textbf{Description}

The CIO chip may be reconfigured by the user via the \texttt{set\_aux\_reg} routine.

\textbf{NOTE:} At present, the initial configuration of the Master Interrupt Control Register is (0x0082). The initial configuration of the Master Configuration Control Register is (0x0194).

\textbf{Inputs}

The only input is a sixteen-bit variable. The high byte is the CIO register number; the low byte is the value to store in the register number. For register numbers and their values, consult Appendix B in Zilog's \textit{Z8036 Z-CIO/Z8536 CIO Counter/Timer and Parallel I/O Unit Technical Manual}, March 1982.

\textbf{Example}

The Master Configuration Control Register allows for selective enabling/disabling of the CIO ports. Port A's input/output is reflected in the least-significant byte of \texttt{reg\_value\_word}. Port B's input/output is reflected in the most-significant byte of \texttt{reg\_value\_word}.

\textbf{NOTE:} Port C of the CIO chip is used internally and is not available to the user of the INTERVIEW.
Suppose you want to disable port B input, output, and interrupts (ports A and C enabled) in one state, and in another state restore the original configuration (ports A, B, and C enabled):

```c
LAYER: 1
    STATE: reconfigure_chip
        CONDITIONS: ENTER_STATE
        ACTIONS:
        {
            set_aux_reg(0x0114);
        }
    STATE: restore_original_config
        CONDITIONS: ENTER_STATE
        ACTIONS:
        {
            set_aux_reg(0x0194);
        }
```
69 Other Library Tools

The C structures, variables, and routines in this section provide additional programming tools not specific to any particular protocol. Most of these tools approximate layer-independent conditions or actions. Refer to Section 27 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.

69.1 Structures

Use the structures tm, crnt_tm, and prev_tm listed in Table 69-1 to monitor the current and previous date and time. Each minute the values in crnt_tm are copied to prev_tm. Then crnt_tm is updated. These structures are used to produce the date/time displays at the top of Run-mode screens and the Date/Time Setup screen.

The variables flag_struct.prev, flag_struct.current, and flag_struct.old (in the flag_struct structure) are used each time a flag is incremented, decremented, or set to a particular value. The current, previous, and old values these variables represent work the same way as their counterparts in the counter structure, discussed fully in Section 62.1(A).

NOTE: The purpose of flags is to make it easy for the user to isolate selected bits in a variable. The translator does most of the work of flags by taking the user’s flag masks and coding them in C. Flags constructed entirely in C bypass the translator and require the programmer to create the flag-mask code normally generated by the translator.

Before using the timeout routines included in this section, declare an instance of the timeout structure shown in Table 69-1. Refer to the timeout_restart_action and timeout_stop_action routines for examples of how to use this structure.

The keyboard structure stores the value of the most recent ASCII key used. The structure variable keyboard.value is updated only by the fast-event variable keyboard_new_key.
### Table 69-1

Structure Fields—Other Library Tools

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Structure Name:</strong> keyboard</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>char</td>
<td>value</td>
<td></td>
<td>ASCII value of key just executed.</td>
</tr>
<tr>
<td><strong>Structure Name:</strong> tm</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>int</td>
<td>tm_sec</td>
<td>0-3b/0-59</td>
<td>Seconds after the minute. Not currently updated; always set to -1.</td>
</tr>
<tr>
<td>int</td>
<td>tm_min</td>
<td>0-3b/0-59</td>
<td>Minutes after the hour.</td>
</tr>
<tr>
<td>int</td>
<td>tm_hour</td>
<td>0-17/0-23</td>
<td>Hours since midnight.</td>
</tr>
<tr>
<td>int</td>
<td>tm_mday</td>
<td>1-1/1-31</td>
<td>Days of month.</td>
</tr>
<tr>
<td>int</td>
<td>tm_mon</td>
<td>0-b/0-11</td>
<td>Months since January.</td>
</tr>
<tr>
<td>int</td>
<td>tm_year</td>
<td>0-6</td>
<td>Days since Sunday. Not currently updated; always set to -1.</td>
</tr>
<tr>
<td>int</td>
<td>tm_wday</td>
<td>0-6</td>
<td>Days since January 1. Not currently updated; always set to -1.</td>
</tr>
<tr>
<td>int</td>
<td>tm_yday</td>
<td>0-16d/0-365</td>
<td>Daylight Savings Time flag. Not currently updated; always set to -1.</td>
</tr>
<tr>
<td><strong>Structure Name:</strong> crnt_tm</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Structure Name:</strong> prev_tm</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Structure Name:</strong> flag_struct</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>unsigned short</td>
<td>prev</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unsigned short</td>
<td>current</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unsigned short</td>
<td>old</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Declared as type `extern struct`. Declared automatically if program KEYBOARD condition is used. Updated by `keyboard_new_key` event variable. Reference the structure variable as follows: `keyboard.value`.

Structure of time of day. Declared as type `extern struct`. Reference a structure variable as follows: `tm.tm_sec`.

Structure of current time of day. Updated every minute. Declared as type `extern struct tm`.

Structure of previous time of day, one minute ago. Declared as type `extern struct tm`.

Structure of a flag. Declared as type `struct`. Declared automatically if a program flag is used. Program flags assigned to structure as follows: `struct flag_struct flag_name`. Reference a structure variable as follows: `flag_name.current`.

When converting a flag action to C, the translator compares `prev` with `current` to determine whether flag has changed. Then `prev` is updated to `current` and `flag_name_change` is signaled.

This value of flag is acted on directly by program actions.

When converting a flag condition to C, the translator compares `old` with `current` to determine whether true condition is new (transitional). After program logic has examined flag, `old` is updated to `prev`. 
Table 69-1 (continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Structure Name:</strong></td>
<td>timeout</td>
<td></td>
<td>Structure of a timeout. Declared as type struct. Declared automatically if a program timeout is used. Program timeouts assigned to structure as follows: struct timeout name. Reference a structure variable as follows: timeout_name.event_id.</td>
</tr>
<tr>
<td>unsigned long</td>
<td>event_id</td>
<td></td>
<td>Four bytes of a 6-byte timeout, containing the segment number and offset. Timeout_name_stop routines set this event id to zero.</td>
</tr>
<tr>
<td>unsigned short</td>
<td>event_id_uid</td>
<td></td>
<td>Two bytes of a 6-byte timeout which uniquely identify (uid) the timeout. Do not try to assign a value to this variable.</td>
</tr>
</tbody>
</table>

69.2 Variables

All of the variables in Table 69-2 are valid in either emulate or monitor mode.

(A) Monitoring Events

The event variables in Table 69-2 are `fevar_time_of_day`, `flag_name_change`, `timeout_name_expired`, `signal_name`, `keyboard_new_key`, and `keyboard_new_any_key`.

Event variable `fevar_time_of_day` comes true once a minute. An example of how to use this variable is provided in Section 54.1. This event variable is part of the spreadsheet TIME condition.

The event variable `keyboard_new_key` is used by the translator in a spreadsheet KEYBOARD condition. It comes true when any ASCII key is pressed. The event `keyboard_new_any_key`, on the other hand, comes true when an ASCII or other keyboard key is pressed. The only keys which will not trigger this event are `~`, `rf`, `fl`-`fl`, `sonkeys`, and `dun`.

(B) Status Variables

Status variables are those in Table 69-2 that do not include `event` in the Type column. Their associated event variables guarantee that they are updated and tested.

Time and date variables are updated by `fevar_time_of_day`. Variables `crnt_time_of_day`, `prev_time_of_day`, `crnt_date_of_day`, and `prev_date_of_day` are older versions of variables that belong to the `crnt_tm` and `prev_tm` structures. The C translator uses these older versions when it construct time-of-day conditions (e.g., `CONDITIONS: TIME 1614`).

The status variable `keyboard_any_key` is updated by the fast-event variable `keyboard_new_any_key`. 
### Table 69-2
Other Library Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern fast_event</td>
<td>fevar_time_of_day</td>
<td></td>
<td>True once per minute. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern event</td>
<td>flag_name_change</td>
<td></td>
<td>This event must be signaled by the program itself; it is not &quot;external&quot; to the program. The translator signals this event as part of the FLAG increment, decrement, or set action. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern event</td>
<td>timeout_name_expired</td>
<td></td>
<td>This event must be signaled by the program itself. It is not &quot;external&quot; to the C program. The translator signals this event as part of the timeout_restart_action routine. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern event</td>
<td>signal_name</td>
<td></td>
<td>True when the named signal is the argument in a signal routine. Spreadsheet-token equivalent is ON_SIGNAL name. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>external volatile unsigned short</td>
<td>crnt_time_of_day</td>
<td>0-937/0-2359</td>
<td>Current time is stored in this variable. Updated as soon as time changes. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>external volatile unsigned short</td>
<td>prev_time_of_day</td>
<td>0-937/0-2359</td>
<td>Current time is stored in this variable. Updated when time changes, but only after logic has had a chance to compare current and previous time. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>external volatile const unsigned char</td>
<td>crnt_date_of_day</td>
<td>1-1/1/1-31</td>
<td>Current date is stored in this variable. Updated as soon as date changes. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>external volatile const unsigned char</td>
<td>prev_date_of_day</td>
<td>1-1/1/1-31</td>
<td>Current date is stored in this variable. Updated when date changes, but only after logic has had a chance to compare current and previous date. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern fast_event</td>
<td>keyboard_new_key</td>
<td></td>
<td>True when any ASCII key is pressed. Line Setup configured for emulate or monitor mode.</td>
</tr>
</tbody>
</table>
# Table 69-2 (continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern volatile unsigned short</td>
<td>keyboard_any_key</td>
<td>True when any key is pressed. The only exceptions are [Esc], [F1]–[F2] softkeys, and [Enter]. Line Setup configured for emulate or monitor mode. Identifies last key or key-combination executed. Line Setup configured for emulate or monitor mode. ASCII keys not used.</td>
<td>0–7/0–127, 80–1711, 128–383</td>
</tr>
<tr>
<td></td>
<td>180/384</td>
<td>HEX</td>
<td>Field entry keys:</td>
</tr>
<tr>
<td></td>
<td>18f/399</td>
<td>HEX</td>
<td>Field entry keys:</td>
</tr>
<tr>
<td></td>
<td>190/400</td>
<td>HEX</td>
<td>Field entry keys:</td>
</tr>
<tr>
<td></td>
<td>191/401</td>
<td>HEX</td>
<td>Field entry keys:</td>
</tr>
<tr>
<td></td>
<td>192/402</td>
<td>HEX</td>
<td>Field entry keys:</td>
</tr>
<tr>
<td></td>
<td>193/403</td>
<td>HEX</td>
<td>Field entry keys:</td>
</tr>
<tr>
<td></td>
<td>194/404</td>
<td>HEX</td>
<td>Field entry keys:</td>
</tr>
<tr>
<td></td>
<td>195/405</td>
<td>HEX</td>
<td>Field entry keys:</td>
</tr>
<tr>
<td></td>
<td>196/406</td>
<td>HEX</td>
<td>Field entry keys:</td>
</tr>
<tr>
<td></td>
<td>10d/269</td>
<td>HEX</td>
<td>Field entry keys:</td>
</tr>
<tr>
<td></td>
<td>1a0/416</td>
<td>HEX</td>
<td>Field entry keys:</td>
</tr>
<tr>
<td></td>
<td>1a1/417</td>
<td>HEX</td>
<td>Field entry keys:</td>
</tr>
<tr>
<td></td>
<td>1a2/418</td>
<td>HEX</td>
<td>Field entry keys:</td>
</tr>
<tr>
<td></td>
<td>1a3/419</td>
<td>HEX</td>
<td>Field entry keys:</td>
</tr>
<tr>
<td></td>
<td>1a4/420</td>
<td>HEX</td>
<td>Field entry keys:</td>
</tr>
<tr>
<td></td>
<td>1a5/421</td>
<td>HEX</td>
<td>Field entry keys:</td>
</tr>
<tr>
<td></td>
<td>1a6/422</td>
<td>HEX</td>
<td>Field entry keys:</td>
</tr>
<tr>
<td></td>
<td>1a7/423</td>
<td>HEX</td>
<td>Field entry keys:</td>
</tr>
<tr>
<td></td>
<td>1a8/424</td>
<td>HEX</td>
<td>Field entry keys:</td>
</tr>
</tbody>
</table>
**Table 69-2 (continued)**

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(keyboard_any_key continued)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1a9/425</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1aa/426</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1ab/427</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1ac/428</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1ad/429</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Editing Keypad Keys (cont):</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1b0/432</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1b1/433</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1b2/434</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1b3/435</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1b4/436</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1b5/437</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1b7/439</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1b8/440</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1ba/442</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Utility Keys:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1bb/443</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1bc/444</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1bd/445</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1be/446</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1bf/447</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1c0/448</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1c1/449</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1c2/450</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1c3/451</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1c4/452</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1c5/453</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1c6/454</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1c7/455</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1c8/456</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1c9/457</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1ca/458</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1cb/459</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1cc/460</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1cd/461</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1ce/462</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pure Cursor Keys:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1d0/464</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1d1/465</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1d2/466</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1d3/467</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1d4/468</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1d5/469</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1d6/470</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*(keyboard_any_key variable continued on next page)*
### Table 69-2 (continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(keyboard_any_key continued)</td>
<td>1d7/471</td>
<td>1d7/471</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>1d8/472</td>
<td>1d8/472</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>1d9/473</td>
<td>1d9/473</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>1da/474</td>
<td>1da/474</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>1db/475</td>
<td>1db/475</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>1dc/476</td>
<td>1dc/476</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>1dd/477</td>
<td>1dd/477</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>1de/478</td>
<td>1de/478</td>
<td>#</td>
</tr>
<tr>
<td>Pure Cursor Keys (cont):</td>
<td>1e0/480</td>
<td>1e0/480</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>1e1/481</td>
<td>1e1/481</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>1e2/482</td>
<td>1e2/482</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>1e3/483</td>
<td>1e3/483</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>1e4/484</td>
<td>1e4/484</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>1e5/485</td>
<td>1e5/485</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>1e6/486</td>
<td>1e6/486</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>1e7/487</td>
<td>1e7/487</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>1e8/488</td>
<td>1e8/488</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>1e9/489</td>
<td>1e9/489</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>1ea/490</td>
<td>1ea/490</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>1eb/491</td>
<td>1eb/491</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>1ec/492</td>
<td>1ec/492</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>1ed/493</td>
<td>1ed/493</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>1ee/494</td>
<td>1ee/494</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>1ef/495</td>
<td>1ef/495</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>1f0/496</td>
<td>1f0/496</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>1f1/497</td>
<td>1f1/497</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>1f2/498</td>
<td>1f2/498</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>1f3/499</td>
<td>1f3/499</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>1f4/500</td>
<td>1f4/500</td>
<td>#</td>
</tr>
<tr>
<td>Cursor Keypad Keys:</td>
<td>1f5/501</td>
<td>1f5/501</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>1f6/502</td>
<td>1f6/502</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>1f7/503</td>
<td>1f7/503</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>1f8/504</td>
<td>1f8/504</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>1f9/505</td>
<td>1f9/505</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>1fa/506</td>
<td>1fa/506</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>1fb/507</td>
<td>1fb/507</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>1fc/508</td>
<td>1fc/508</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>1fd/509</td>
<td>1fd/509</td>
<td>#</td>
</tr>
<tr>
<td>Other Keys:</td>
<td>1fe/50a</td>
<td>1fe/50a</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>1f1/50b</td>
<td>1f1/50b</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>1f2/50c</td>
<td>1f2/50c</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>1f3/50d</td>
<td>1f3/50d</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>1f4/50e</td>
<td>1f4/50e</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>1f5/50f</td>
<td>1f5/50f</td>
<td>#</td>
</tr>
</tbody>
</table>
69.3 Routines

timeout_restart_action

Synopsis

extern void timeout_restart_action(timeout_name_ptr, value, function);
struct * timeout_name_ptr
{
    unsigned long event_id;
    unsigned short event_id_uid;
};
unsigned short value;
void function ();

Description

This routine starts a named timeout timer running down, starting at a specified value. When the timer reaches zero, a named function is called. The timeout_restart_action routine, preceded by a call to the timeout_stop_action routine, is the equivalent of the softkey TIMEOUT name RESTART action on the Protocol Spreadsheet.

Inputs

The first parameter is a pointer to the timeout structure. See Table 69-1 for further explanation of the timeout structure.

The second parameter is the starting value of the timeout timer in milliseconds.

The third parameter is the name of a routine to be called when the timeout expires. The routine may include the following statement: timeout_name.event_id = 0;
Timeout-stop actions set this event ID to zero. This action is not strictly necessary here, since the timeout has already expired; but the action may make the processing of subsequent stop actions slightly more efficient.

The body of the routine to be called may also include this statement: signal(timeout_name_expired);. In a softkey-entered TIMEOUT RESTART action, both statements are included in a routine called timeout_name_isp.

NOTE: The routine named in the third parameter is an interrupt service process (isp). A long definition for this routine makes the processing of timeout_restart_action unpredictable.

Example

When a frame is sent, start a timeout timer at 2 seconds. When it expires, sound the alarm. If another frame is sent before the 2 seconds expires, stop the current timer and restart the timeout.
Here is a version of the program that accomplishes the same result without an action to signal the timeout event:

```c

{  
struct timeout
  {  
    unsigned long event_id;
    unsigned short event_id_uid;
  };
struct timeout timeout_example;
extern event timeout_example_expired;
void timeout_example_isp ()
{  
  timeout_example.event_id = 0;
  signal(timeout_example_expired);
}
}

LAYER: 2
STATE: example_of_timeout
CONDITIONS: FRAME_SENT
ACTIONS:
{  
  timeout_stop_action(&timeout_example);
  timeout_restart_action(&timeout_example, 2000, timeout_example_isp);
}
CONDITIONS:
{  
  timeout_example_expired
}
ACTIONS: ALARM
```
timeout_stop_action

Synopsis

extern void timeout_stop_action(timeout_name_ptr);
struct * timeout_name_ptr
{
    unsigned long event_id;
    unsigned short event_id_uid;
};

Description

This routine stops a named timeout timer, preventing it from expiring. The softkey equivalent of this routine is the TIMEOUT name STOP action on the Protocol Spreadsheet. timeout_stop_action also precedes the call to the timeout_restart_action in the spreadsheet TIMEOUT name RESTART action.

Inputs

The only parameter is a pointer to the timeout structure. See Table 69-1 for further explanation of the timeout structure.

Example

In this example, if the user presses the S key, the timeout timer will not expire and the alarm will not sound (until another frame is sent and the timeout is restarted).

{  
    struct timeout
    {
        unsigned long event_id;
        unsigned short event_id_uid;
    },
    struct timeout timeout_example;
    extern void sound_alarm();
}

LAYER: 2
STATE: stop_a_timeout
CONDITIONS: FRAME_SENT
ACTIONS:
{
    timeout_stop_action(&timeout_example);
    timeout_restart_action(&timeout_example, 2000, sound_alarm);
}
CONDITIONS: KEYBOARD "Ss"
ACTIONS:
{
    timeout_stop_action(&timeout_example);
}
index

Synopsis

extern char * index(string, character);
char * string;
char character;

Description

This routine searches for an instance of a character starting at the beginning of a specified list. The routine is used by the C translator to convert CONDITIONS:
KEYBOARD softkey entries into C. This routine must be declared.

Inputs

The first parameter is a list of characters to be searched.
The second parameter is the character to be searched for in the list.

Returns

This routine returns a pointer to the first instance of the specified character, or zero if it does not occur.

Example

In the example below, the following test is established: when a key is pressed on the keyboard, search for a match to the keyboard character in the string “abc”. If it is found, sound the alarm.

```
{ extern char * index();
 extern fast_event keyboard_new_key;
 extern struct keyboard
  {
   char value;
  };
 extern struct keyboard keyboard;
} LAYER: 1
 STATE: index_example
 CONDITIONS:
  {
   (keyboard_new_key && index(" abc ", keyboard.value))
  }
 ACTIONS: ALARM
```

Let’s suppose that the user presses the space bar. In this case, the returned pointer will be pointing to the blank preceding the “a.” If rindex had been used, the returned pointer would be pointing to the blank following the “c.” As long as any non-null character is returned, the condition is true.
rindex

Synopsis

eextern char * rindex(string, character);
char * string;
char character;

Description

This routine searches for an instance of a character starting at the end of a specified list. This routine must be declared.

Inputs

See index.

Returns

See index.

Example

See index.

load_program

Synopsis

eextern void load_program(filename_ptr)
const char * filename_ptr;

Description

The load_program routines allows you to link programs together while the unit is in Run mode. When a call to load_program is encountered in a spreadsheet program, the current program is exited. The program named as the argument in the routine is loaded and run. When you return to Program mode, the program displayed on the Protocol Spreadsheet will be the one just loaded. If load_program fails, you are returned to the main menu screen in Program mode.

Inputs

The only input is the absolute pathname, prefixed by the device name, of the file to be loaded. Valid device names are “HRD,” “FD1,” and “FD2.”

Example

In the example below, at the successful conclusion of the last of a series of tests in module 18, a program for module 19 will be loaded and run.
LOCK: 3
STATE: test_26
CONDITIONS: ENTER_STATE
ACTIONS: SEND DIYAG
CONDITIONS: RCV CLEAR_CONF
ACTIONS: TRACE "Test 26 passed"
{
load_program("FD1/usr/module_19");
}

Synopsis

#include <stdio.h>
extern void lock(lock_variable_ptr);
int * lock_variable_ptr;

Description

The lock routine implements a lock using the integer variable pointed to by the routine parameter. If the lock variable is currently locked, the task goes to sleep. When an unlock on the same variable occurs (within an independent task), the task invoking the lock function will attempt to claim the lock. If successful, the task is executed; otherwise, it goes back to sleep until the next unlock.

NOTE: If locking is used at any place in the program, all related or possibly concurrent routines must also use the locking functions.

NOTE: The lock variable should always be defined as a global integer, never as local to a function. The lock variable should never be altered by the user program or deadlock can occur. Deadlock also results if the lock is invoked twice within the same task without an intervening unlock.

Inputs

The only parameter is a pointer to the lock variable.

Example

Two tasks concurrently write to their own file streams. (The file streams are local to the routine write_fox, making them independent of each other even though they have the same name.) However, during the fclose operation (which automatically calls fflush), both tasks need to write to the same file. The locking routines ensure that the writes to the file occur sequentially, not concurrently.
INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

```c
#include <stdio.h>
const char data[] = "((FOX))\n";
int key;
void write_fox()
{
    FILE *stream_ptr;
    size_t n;
    lock(&key);
    if((stream_ptr = fopen("FD2/usr/buff01", "a")) == 0)
        display_prompt("Cannot open file.");
    else
        display_prompt("File opened.");
    n = fwrite(data, 1, sizeof(data)-1, stream_ptr);
    pos_cursor(1,0);
    if(n != (sizeof(data)-1))
        display("Write error.");
    else
        display("Write completed.");
    if(fclose(stream_ptr) != 0)
        display("Either file is already closed, or close cannot be executed.");
    else
        display("File closed.");
    unlock(&key);
}
```

LAYER: 1

TEST: a
STATE: write_and_signal
CONDITIONS: RECEIVE STRING "THE QUICK BROWN FOX"
ACTIONS: SIGNAL xyz
{
    write_fox();
}

TEST: b
STATE: write_only
CONDITIONS: ON_SIGNAL xyz
ACTIONS:
{
    write_fox();
}

unlock

Synopsis

#include <stdio.h>
extern void unlock(lock_variable_ptr);
int *lock_variable_ptr;

Description

The `unlock` routine implements the inverse of the lock routine using the same integer variable. Sleeping tasks will be woken up to retry their attempt to claim the lock. One will succeed, and the rest will go back to sleep. See also lock routine.
Inputs
The only parameter is a pointer to the lock variable.

Example
See lock routine.

signal

Synopsis
extern void signal(signal_name);

Description
This routine conveys instructions to other tests and layers where conditions are monitoring the signal by name. The softkey equivalent of this routine is the SIGNAL action on the Protocol Spreadsheet.

Inputs
The only parameter is a name descriptive of the event being signaled.

Example
LAYER: 2
  STATE: signal_routine
  CONDITIONS: RCV FRMR
  ACTIONS:
  {
    signal(signal_link_down);
  }
  CONDITIONS: ON_SIGNAL link_down
  ACTIONS: ALARM

Here is a related example, this time with the signal detection also given in C. Note that a signal automatically generates an “event” that can be detected alone in a waitfor clause.

{ extern event link_down;
  }
LAYER: 2
  STATE: signal_event
  CONDITIONS: RCV FRMR
  ACTIONS:
  {
    signal(link_down);
  }
  CONDITIONS:
  {
    link_down
  }
  ACTIONS: ALARM
sound_alarm

Synopsis

extern void sound_alarm();

Description

This routine will sound the alarm. The softkey equivalent of this routine is the ALARM action on the Protocol Spreadsheet.

Example

When a bad BCC is detected on the DTE side of the link, sound the alarm.

LAYER: 1
STATE: example
CONDITIONS: DTE BAD_BCC
ACTIONS:
{
    sound_alarm();
}

start_rcrd_play

Synopsis

extern void start_rcrd_play();

Description

Depending on the Line Setup configuration, this routine activates data recording or playback. If the Line Setup menu shows Mode: MONITOR, Source: DISK, the routine controls playback. In all other cases, it initiates recording.

Unless your recording source is RAM, make a call to fclose in programs containing disk I/O routines (Section 65) before you start to record (or resume playback). If you don’t, the file will be closed automatically as soon as recording (or playback) begins, even if processes on the file have not been completed. (Using the MON key to activate recording or resume playback will have the same effect.)

Example

LAYER: 1
STATE: example
CONDITIONS: KEYBOARD
ACTIONS:
{
    start_rcrd_play();
}
suspend_rcrd_play

Synopsis

extern void suspend_rcrd_play();

Description

Depending on the Line Setup configuration, this routine suspends data recording or playback. If the Line Setup menu shows **Mode: MONITOR, Source: DISK**, the routine controls playback. In all other cases, it suspends recording. Once recording or playback is suspended, resume it with a call to start_rcrd_play.

Unless your recording source is RAM, do not call disk I/O routines (Section 65) until you suspend recording (or playback). If you do, the disk I/O operation will fail.

NOTE: Although playback is immediately suspended when suspend_rcrd_play is executed, the screen display continues until the character buffer’s contents are fully displayed. (For bit-image data, the FIFO must empty.) At slower playback speeds, you may notice a slight delay before the display actually freezes.

Example

LAYER: 2
STATE: example
CONDITIONS: KEYBOARD · ·
ACTIONS:
{}
  suspend_rcrd_play();

send_key

Synopsis

extern void send_key(number_of_keys, keys_ptr);
unsigned char number_of_keys;
unsigned short * keys_ptr;

Description

This routine sends a specific keystroke (or sequence of keys) during Run mode, as though the operator pressed the key. It also may be used to change the Run–mode display.

Inputs

The first parameter specifies the number of keys to be sent.
The second parameter is a pointer to an array of *shorts*. This array lists the keys to be sent. To send keyboard keys, use the values listed in Table 69-2 for the *keyboard_any_key* variable. To change the Run-mode display, send two keys. The first "key" always has a value of 0xff75. The second "key" identifies the desired display-screen. Use the values listed in Table 61-1 for the *crnt_display_screen* variable.

**Example**

For this example, assume you are playing back data from a disk and that the initial Run-mode screen is the dual-line data display. After a five-second pause, playback is slowed as though you pressed 0. As soon as a bad BCC is detected on the DTE side, the data display will change to the Layer 2 Protocol Trace screen.

```c
{  
    unsigned short keys [] = {0xff75, 0x42};
    unsigned short slow_down [] = {0x1dc};
}
LAYER: 2
STATE: change_displays
    CONDITIONS: ENTER_STATE
    ACTIONS: TIMEOUT pause RESTART 5
    CONDITIONS: TIMEOUT pause
    {
        send_key(1, slow_down);
    }
    CONDITIONS: DTE BDBCC
    ACTIONS:
    {
        send_key(2, keys);
    }
```
70 X.21 Library

The Test Interface Module (TIM) located in the rear of the INTERVIEW determines the leads available for monitoring and control (Section 10). 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 60.

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 7.1(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 32 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.

70.1 Structures

Use the structure *xmit_list*, shown in Table 70-1, when transmitting line data via the *x21_transmit_call* routine. Refer to *x21_transmit_call* in Section 70.3(A) for an example of how to use this structure.

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Structure Name:</strong> xmit_list</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>unsigned char * string</td>
<td>pointer to the location of the transmit string—the transmit string is declared separately</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unsigned short string_length</td>
<td>length of the transmit string</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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*
70.2 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 70-2.

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 70-2 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 70-2.

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 70-2 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-ORed with `previous_eia_leads` to identify all the leads that changed. Then the result was bitwise ANDed 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`.

70-2
Table 70-2
X.21 Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern fast_event</td>
<td>fevar_eia_changed</td>
<td></td>
<td>True when the status changes for an EIA lead. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern const volatile unsigned short</td>
<td>current_eia_leads</td>
<td>1</td>
<td>C-valid</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>B (RS-232 equivalent is SQ)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>I-valid (RI)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>R-valid (DSR)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10/16</td>
<td>T-valid (DTR)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>40/64</td>
<td>I (CTS)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>80/128</td>
<td>C (RTS)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100/256</td>
<td>R (RD)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>200/512</td>
<td>UA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>400/1024</td>
<td>T (TD)</td>
</tr>
</tbody>
</table>

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).

Examples:

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.

Same values as current_eia_leads. Updated when current_eia_changed has updated the 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 anded with `current_eia_leads`, zero will mean that the lead is on. If the result equals the mask, the lead is off. If a mask testing for validity is anded 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:

```c
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:

```c
CONDITIONS:
{
    (event_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.

70.3 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."

```c
cntl_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("R");
    }
```
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+".

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 anded 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:
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**

```c
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.

```c
{  
  extern fast_event fevar_eia_changed;
  extern const volatile unsigned short current_eia_leads;
}
```

**enter_data_phase**

**Synopsis**

```c
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.

```c
{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();
}
```
71 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 33 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.

71.1 Structures

The structure send_frame_structure defines the format of transmitted X.25 frames. See Table 71-1. Use this structure to send frames via the send_frame routine in emulate mode. See Section 71.3(B). Each variable in the structure relates to some softkey selection or user entry in the SEND action.

71.2 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 71-2 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 71-1**  
X.25 Layer 2 Structures

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structure Name: send_frame_structure</td>
<td>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};</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unsigned char</td>
<td>addr_type</td>
<td>0</td>
<td>command</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>response</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>other</td>
</tr>
<tr>
<td>unsigned char</td>
<td>frame_type</td>
<td>(The codes for frame_type are the same as for the X.25-variable rcvd_frame_type.)</td>
<td></td>
</tr>
<tr>
<td>unsigned char</td>
<td>nr_type</td>
<td>0</td>
<td>auto</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>received ns plus 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>last nr sent</td>
</tr>
<tr>
<td>unsigned char</td>
<td>ns_type</td>
<td>0</td>
<td>auto</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>skip</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>last nr received</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>value</td>
</tr>
<tr>
<td>unsigned char</td>
<td>p_f_type</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>loopback</td>
</tr>
<tr>
<td>unsigned char</td>
<td>bcc_type</td>
<td>0</td>
<td>default (bad bcc)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>good bcc</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>bad bcc</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>abort</td>
</tr>
<tr>
<td>unsigned char</td>
<td>addr_value</td>
<td>1</td>
<td>to DCE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>to DTE</td>
</tr>
<tr>
<td>unsigned char</td>
<td>cntrl_byte</td>
<td>(actual value of the control byte)</td>
<td></td>
</tr>
<tr>
<td>unsigned char</td>
<td>nr_value</td>
<td>0–7 (MOD 8)</td>
<td>If nr_type = 1</td>
</tr>
<tr>
<td>unsigned char</td>
<td>ns_value</td>
<td>0–7 (MOD 8)</td>
<td>If ns_type = 3</td>
</tr>
<tr>
<td>Type</td>
<td>Variable</td>
<td>Value (hex/decimal)</td>
<td>Meaning</td>
</tr>
<tr>
<td>----------</td>
<td>---------------</td>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>extern</td>
<td>dte_frame</td>
<td>True</td>
<td>when a DTE frame is detected. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern</td>
<td>dce_frame</td>
<td>True</td>
<td>when a DCE frame is detected. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern</td>
<td>dte_good_bcc</td>
<td>True</td>
<td>when a good BCC is calculated for a DTE frame. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern</td>
<td>dce_good_bcc</td>
<td>True</td>
<td>when a good BCC is calculated for a DCE frame. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern</td>
<td>dte_bad_bcc</td>
<td>True</td>
<td>when a bad BCC is calculated for a DTE frame. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern</td>
<td>dce_bad_bcc</td>
<td>True</td>
<td>when a bad BCC is calculated for a DCE frame. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern</td>
<td>dte_abort</td>
<td>True</td>
<td>when an abort is detected for a DTE frame. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern</td>
<td>dce_abort</td>
<td>True</td>
<td>when an abort is detected for a DCE frame. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern</td>
<td>rcvd_frame</td>
<td>True</td>
<td>when a frame is received. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern</td>
<td>invalid_frame</td>
<td>True</td>
<td>when an invalid frame is detected. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern</td>
<td>I2_T1</td>
<td>True</td>
<td>when the T1 timeout-timer has expired. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern</td>
<td>bcc_error</td>
<td>True</td>
<td>when a BCC error is detected. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern</td>
<td>nr_error</td>
<td>True</td>
<td>when an N(R) error is detected in a received INFO or supervisory frame. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern</td>
<td>ns_error</td>
<td>True</td>
<td>when an N(S) error is detected in a received INFO frame. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>Type</td>
<td>Variable</td>
<td>Value (hex/decimal)</td>
<td>Meaning</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------</td>
<td>---------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>extern event</td>
<td>frame_sent</td>
<td></td>
<td>True when frame is passed down to Layer 1. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_frame_addr</td>
<td>1</td>
<td>to DCE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>to DTE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_frame_type</td>
<td></td>
<td>(same as rcvd_frame_type-Line Setup configured for emulate or monitor mode)</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_frame_cntl_byte_1</td>
<td></td>
<td>(actual value of control byte-Line Setup configured for emulate or monitor mode)</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_frame_pf</td>
<td>0</td>
<td>pf=0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10/16</td>
<td>pf=1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_frame_bcc_type</td>
<td>1</td>
<td>good</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>bad</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>abort</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>rcvd_frame_addr</td>
<td>1</td>
<td>to DCE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>to DTE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>rcvd_frame_type</td>
<td>0</td>
<td>info</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>rr</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>rnr</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
<td>rej</td>
</tr>
<tr>
<td></td>
<td></td>
<td>d/13</td>
<td>srej</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2f/47</td>
<td>sabm</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6f/111</td>
<td>sabme</td>
</tr>
<tr>
<td></td>
<td></td>
<td>43/67</td>
<td>disc</td>
</tr>
<tr>
<td></td>
<td></td>
<td>f/15</td>
<td>dm</td>
</tr>
<tr>
<td></td>
<td></td>
<td>f/15</td>
<td>sarm</td>
</tr>
<tr>
<td></td>
<td></td>
<td>63/99</td>
<td>ua</td>
</tr>
<tr>
<td></td>
<td></td>
<td>87/135</td>
<td>fmr</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ff/255</td>
<td>other</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ff/255</td>
<td>unknown</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>rcvd_frame_cntl_byte_1</td>
<td></td>
<td>(actual value of control byte-Line Setup configured for emulate mode only)</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>rcvd_frame_pf</td>
<td>0</td>
<td>pf=0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10/16</td>
<td>pf=1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>rcvd_frame_bcc_type</td>
<td>1</td>
<td>good</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>bad</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>abort</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>rcvd_frame_nr</td>
<td>0-7 (MOD 8)</td>
<td>Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>Type</td>
<td>Variable</td>
<td>Value (hex/decimal)</td>
<td>Meaning</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>------------------------</td>
<td>---------------------</td>
<td>---------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>rcvd_frame_ns</td>
<td>0-7 (MOD 8)</td>
<td>Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>rcvd_frame_buff_seg</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>rcvd_frame_sdu_offset</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>rcvd_frame_sdu_size</td>
<td></td>
<td>Size of service data unit in a received frame. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>l2_current_window_edge</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>l2_lower_window_edge</td>
<td></td>
<td>see l2_current_window_edge</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>l2_upper_window_edge</td>
<td></td>
<td>see l2_current_window_edge</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>l2_resend_edge</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>extern unsigned char</td>
<td>l2_enhance</td>
<td>0</td>
<td>normal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>reverse</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>low</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>reverse low</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>blink</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
<td>reverse blink</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12/18</td>
<td>blink low</td>
</tr>
<tr>
<td></td>
<td>l2_suppress</td>
<td>0</td>
<td>Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Line Setup configured for emulate or monitor mode.</td>
</tr>
</tbody>
</table>
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`, `I2_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 71-2 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:
{ rcdv_frame && (rcdv_frame_type == 0) }
```

1. **Frame characteristics.** All status variables in Table 71-2 containing an `m_` prefix are valid in either emulate or monitor mode: `m_frame_addr`, `m_frame_type`, `m_frame_ctrl_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 71-2 containing a `rcvd_` prefix are valid in emulate mode only: `rcvd_frame_addr`, `rcvd_frame_type`, `rcvd_frame_ctrl_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 63.1 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 71-2. 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
{
    12_enhance == 1
}
ACTIONS:
{
    12_enhance = 0;
}

or an ACTIONS block:

CONDITIONS: RCV INFO
ACTIONS:
{
    if(12_enhance == 1)
        12_enhance = 0;
}

71.3 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 12_give_data."

(A) Receive

12_give_data

Synopsis

extern void 12_give_data();

Description

The 12_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:
(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 (PROTOCOL) 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, 1, or 2 (for loopback).

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:
    { resed_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 (PROTOCOL) 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 and supervisory frames is reset to zero and the transmit window is cleared. The softkey equivalent of this routine is the (PROTOCOL) 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 pf_type;
  unsigned char bcc_type;
  unsigned char addr_value;
  unsigned char ctrl_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 63.3(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 63.3(A).

The third parameter is the offset from the beginning of the buffer to the start of the service data unit. See Section 63.3(A).
The fourth parameter is a pointer to the frame structure to be sent. For a description of `send_frame_structure`, see Table 71-1.

Example

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

```c
{
    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:
```c
    _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);
```
### 72 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 34 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.

#### 72.1 Structures

The `send_packet_structure` defines the format of transmitted X.25 packets. See Table 72-1. Use this structure to send packets via the `send_packet` routine in emulate mode. See Section 72.3(B). Each variable in the structure relates to some softkey selection or user entry in the SEND action.

#### 72.2 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:

   ```c
   dte_packet && (m_packet_type == 0)
   ```
### Table 72-1
**X.25 Layer 3 Structures**

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structure Name: send_packet_structure</td>
<td>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, 1, 0, 0, 1, 1, 0, 2, &amp;facilities_string[0], 0, 0};</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned char</td>
<td>path_num</td>
<td>0-8</td>
<td>path number</td>
</tr>
<tr>
<td>unsigned char</td>
<td>packet_type</td>
<td>(The codes for packet_type are the same as for the X.25-variable m_packet_type.)</td>
<td>use path number of last received packet</td>
</tr>
<tr>
<td>unsigned char</td>
<td>packet_type_byte</td>
<td>(actual value of the packet type byte)</td>
<td></td>
</tr>
<tr>
<td>unsigned char</td>
<td>m_bit</td>
<td>0</td>
<td>m = 0</td>
</tr>
<tr>
<td>unsigned char</td>
<td>m_bit</td>
<td>1</td>
<td>m = 1</td>
</tr>
<tr>
<td>unsigned char</td>
<td>d_bit</td>
<td>0</td>
<td>d = 0</td>
</tr>
<tr>
<td>unsigned char</td>
<td>d_bit</td>
<td>40/64</td>
<td>d = 1</td>
</tr>
<tr>
<td>unsigned char</td>
<td>q_bit</td>
<td>0</td>
<td>q = 0</td>
</tr>
<tr>
<td>unsigned char</td>
<td>q_bit</td>
<td>80/128</td>
<td>q = 1</td>
</tr>
<tr>
<td>unsigned char</td>
<td>pr_type</td>
<td>0</td>
<td>auto</td>
</tr>
<tr>
<td>unsigned char</td>
<td>pr_type</td>
<td>1</td>
<td>value</td>
</tr>
<tr>
<td>unsigned char</td>
<td>pr_type</td>
<td>2</td>
<td>received ps plus 1</td>
</tr>
<tr>
<td>unsigned char</td>
<td>pr_type</td>
<td>3</td>
<td>last pr sent</td>
</tr>
<tr>
<td>unsigned char</td>
<td>ps_type</td>
<td>0</td>
<td>auto</td>
</tr>
<tr>
<td>unsigned char</td>
<td>ps_type</td>
<td>1</td>
<td>skip</td>
</tr>
<tr>
<td>unsigned char</td>
<td>ps_type</td>
<td>2</td>
<td>received pr</td>
</tr>
<tr>
<td>unsigned char</td>
<td>ps_type</td>
<td>3</td>
<td>value</td>
</tr>
<tr>
<td>unsigned char</td>
<td>pr_value</td>
<td>0-7 (MOD 8)</td>
<td>If pr_type = 1</td>
</tr>
<tr>
<td>unsigned char</td>
<td>ps_value</td>
<td>0-7 (MOD 8)</td>
<td>If ps_type = 3</td>
</tr>
<tr>
<td>unsigned char</td>
<td>cause</td>
<td>(value of cause byte—see Figure 36-15)</td>
<td></td>
</tr>
<tr>
<td>unsigned char</td>
<td>diag_flag</td>
<td>0</td>
<td>diagnostic field not present</td>
</tr>
<tr>
<td>unsigned char</td>
<td>diag_flag</td>
<td>1</td>
<td>diagnostic field is present</td>
</tr>
<tr>
<td>unsigned char</td>
<td>diag</td>
<td>(value of diagnostic byte—consult CCITT Recommendation X.25, pp. 237-8)</td>
<td></td>
</tr>
<tr>
<td>unsigned char</td>
<td>spare</td>
<td>0</td>
<td>reserved space</td>
</tr>
<tr>
<td>unsigned char</td>
<td>facilities_len</td>
<td>0-ff/0-255</td>
<td>length of the facilities field</td>
</tr>
<tr>
<td>char *</td>
<td>facilities</td>
<td></td>
<td>pointer to the location of the facilities field—the facilities field is declared separately</td>
</tr>
<tr>
<td>unsigned char</td>
<td>data_len</td>
<td></td>
<td>reserved for future use</td>
</tr>
<tr>
<td>char *</td>
<td>data</td>
<td></td>
<td>reserved for future use</td>
</tr>
<tr>
<td>Type</td>
<td>Variable</td>
<td>Value (hex/decimal)</td>
<td>Meaning</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------</td>
<td>---------------------</td>
<td>---------</td>
</tr>
<tr>
<td>extern event</td>
<td>dte_packet</td>
<td></td>
<td>True when a DTE packet is detected. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern event</td>
<td>dce_packet</td>
<td></td>
<td>True when a DCE packet is detected. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern event</td>
<td>rcvd_packet</td>
<td></td>
<td>True when a packet is received from Layer 2. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern event</td>
<td>invalid_packet</td>
<td></td>
<td>True when an invalid packet is detected. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern event</td>
<td>pr_error</td>
<td></td>
<td>True when an P(R) error is detected in a data or supervisory packet. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern event</td>
<td>ps_error</td>
<td></td>
<td>True when an P(S) error is detected in a data packet. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern event</td>
<td>packet_sent</td>
<td></td>
<td>True when a packet has been passed down to Layer 2. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>m_packet_lcn</td>
<td>0–ffff/0–4095</td>
<td>Logical channel number. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile unsigned char</td>
<td>m_packet_lcn_grp</td>
<td>0–f/0–15</td>
<td>Logical channel group number. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_packet_q</td>
<td>0 80/128</td>
<td>q=0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>q=1</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_packet_d</td>
<td>0 40/64</td>
<td>d=0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>d=1</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_packet_m</td>
<td>0 10/16</td>
<td>m=0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>m=1</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_packet_pr</td>
<td>0–7 (MOD 8)</td>
<td>Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_packet_ps</td>
<td>0–7 (MOD 8)</td>
<td>Line Setup configured for emulate or monitor mode.</td>
</tr>
</tbody>
</table>
Table 72-2 (continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_packet_cause</td>
<td>(same as rcvd_pkt_cause—Line Setup configured for emulate or monitor mode)</td>
<td></td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_packet_diag_code</td>
<td>(same as rcvd_pkt_diag—Line Setup configured for emulate or monitor mode)</td>
<td></td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_packet_type_byte</td>
<td>(actual value of packet type byte—Line Setup configured for emulate or monitor mode)</td>
<td></td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_packet_type</td>
<td>b/11 call</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>f/15 call acc</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>13/19 clear</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>17/23 clear conf</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 data</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>23/35 int</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>27/39 int conf</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 rr</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>5 rrn</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>9 rej</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1b/27 reset</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1f/31 reset conf</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>fb/251 restart</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ff/255 restart conf</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>f1/241 diag</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>f3/243 reg</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>f7/247 reg conf</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>11/17 other pkt</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>11/17 unknown pkt</td>
<td></td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>rcvd_pkt_lcn</td>
<td>0-ff/0-4095</td>
<td>Logical channel number in a received packet. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>rcvd_pkt_q</td>
<td>0 q=0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>80/128 q=1</td>
<td></td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>rcvd_pkt_d</td>
<td>0 d=0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>40/64 d=1</td>
<td></td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>rcvd_pkt_m</td>
<td>0 m=0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>10/16 m=1</td>
<td></td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>rcvd_pkt_pr</td>
<td>0-7 (MOD 8)</td>
<td>Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>rcvd_pkt_ps</td>
<td>0-7 (MOD 8)</td>
<td>Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>rcvd_pkt_cause</td>
<td>(see Figure 36-15—Line Setup configured for emulate mode only)</td>
<td></td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>rcvd_pkt_diagn</td>
<td>(consult CCITT Recommendation X.25, pp.237-8—Line Setup configured for emulate mode only)</td>
<td></td>
</tr>
</tbody>
</table>
Table 72-2 (continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern volatile const unsigned char</td>
<td>rcvd_pkt_type_byte</td>
<td>b/11</td>
<td>call</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>rcvd_packet_type</td>
<td>1f/15</td>
<td>call acc</td>
</tr>
<tr>
<td></td>
<td></td>
<td>13/19</td>
<td>clear</td>
</tr>
<tr>
<td></td>
<td></td>
<td>17/23</td>
<td>clear conf</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>23/35</td>
<td>int</td>
</tr>
<tr>
<td></td>
<td></td>
<td>27/39</td>
<td>int conf</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>rr</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>rnr</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
<td>rej</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1b/27</td>
<td>reset</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1f/31</td>
<td>reset conf</td>
</tr>
<tr>
<td></td>
<td></td>
<td>fb/251</td>
<td>restart</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ff/255</td>
<td>restart conf</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11/241</td>
<td>diag</td>
</tr>
<tr>
<td></td>
<td></td>
<td>13/243</td>
<td>reg</td>
</tr>
<tr>
<td></td>
<td></td>
<td>17/247</td>
<td>reg conf</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11/17</td>
<td>other pkt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11/17</td>
<td>unknown pkt</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>m_packet_buff_seg</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>m_packet_info_seg</td>
<td></td>
<td>Same as m_packet_buff_seg.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>m_packet_sdu_offset</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>m_packet_info_offset</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>m_packet_length</td>
<td></td>
<td>Length of the packet, including header. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>m_packet_info_length</td>
<td></td>
<td>Length of the packet information, excluding the header. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>Type</td>
<td>Variable</td>
<td>Value (hex/decimal)</td>
<td>Meaning</td>
</tr>
<tr>
<td>------</td>
<td>----------</td>
<td>---------------------</td>
<td>---------</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>rcvd_pkt_buff_seg</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>rcvd_pkt_info_seg</td>
<td></td>
<td>Same as rcvd_pkt_buff_seg.</td>
</tr>
<tr>
<td>extern unsigned short</td>
<td>rcvd_pkt_sdu_offset</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>rcvd_pkt_info_offset</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>extern unsigned short</td>
<td>rcvd_pkt_length</td>
<td></td>
<td>Length of a received packet, including header information. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>rcvd_pkt_info_length</td>
<td></td>
<td>Length of the information in a received packet, excluding the header. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile unsigned char *</td>
<td>m_packet_ptr</td>
<td></td>
<td>Pointer to the packet, beginning at the first byte in the header. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile unsigned char *</td>
<td>m_packet_info_ptr</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>extern volatile unsigned char *</td>
<td>rcvd_packet_ptr</td>
<td></td>
<td>Pointer to the packet, beginning at the first byte in the header. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile unsigned char *</td>
<td>rcvd_pkt_info_ptr</td>
<td></td>
<td>Pointer to the packet information, initially located at the byte immediately following the packet header. Line Setup configured for emulate mode only.</td>
</tr>
</tbody>
</table>
Table 72-2 (continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern volatile const unsigned char</td>
<td>rcvd_device_path</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>extern unsigned char</td>
<td>l3_enhance</td>
<td>0</td>
<td>normal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>reverse</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>low</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>reverse low</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>blink</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
<td>reverse blink</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12/18</td>
<td>blink low</td>
</tr>
<tr>
<td></td>
<td>l3_suppress</td>
<td>0</td>
<td>off</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>on</td>
</tr>
</tbody>
</table>

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 \texttt{pr\_error}, \texttt{ps\_error}, and \texttt{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 \texttt{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 72-2 that do not include \texttt{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 \texttt{RCV DATA}. The C version of the same condition should look like this:

\begin{verbatim}
CONDITIONS:
{
  rcvd_packet && (rcvd_packet_type == 0)
}
\end{verbatim}

1. \textit{Packet characteristics}. All status variables in Table 72-2 containing an \texttt{m}-prefix are valid in either emulate or monitor mode: \texttt{m\_packet\_lcn}, \texttt{m\_packet\_lcn\_grp}, \texttt{m\_packet\_q}, \texttt{m\_packet\_d}, \texttt{m\_packet\_m}, \texttt{m\_packet\_pr}, \texttt{m\_packet\_ps}, \texttt{m\_packet\_cause}, \texttt{m\_packet\_diag\_code}, \texttt{m\_packet\_type}, and \texttt{m\_packet\_type\_byte}.

All status variables in Table 72-2 containing a \texttt{rcvd}-prefix are valid in emulate mode only: \texttt{rcvd\_pkt\_lcn}, \texttt{rcvd\_pkt\_q}, \texttt{rcvd\_pkt\_d}, \texttt{rcvd\_pkt\_m}, \texttt{rcvd\_pkt\_pr}, \texttt{rcvd\_pkt\_ps}, \texttt{rcvd\_pkt\_cause}, \texttt{rcvd\_pkt\_diag\_code}, \texttt{rcvd\_pkt\_type}, and \texttt{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 \texttt{rcvd\_packet\_type}.”

2. \textit{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 \texttt{rcvd\_pkt\_buff\_seg}, \texttt{m\_packet\_buff\_seg}, \texttt{rcvd\_pkt\_sdu\_offset}, \texttt{m\_packet\_sdu\_offset}, \texttt{rcvd\_pkt\_length}, and \texttt{m\_packet\_length}. See Section 63.1 for a more detailed discussion of the buffer components to which these variables refer.

3. \textit{Pointers}. Two variables, \texttt{rcvd\_pkt\_info\_ptr} and \texttt{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 13_enhance or 13_suppress.
The values are listed in Table 72-2. 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:
{  
  13_enhance = 1;
}

CONDITIONS: RCV INVALID
ACTIONS:
{  
  13_suppress = 1;
}

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

CONDITIONS: RCV DATA
{  
  13_enhance == 1
}
ACTIONS:
{  
  13_enhance = 0;
}

or an ACTIONS block:

CONDITIONS: RCV DATA
ACTIONS:
{  
  if(13_enhance == 1)
    13_enhance = 0;
}
72.3 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 13_give_data."

(A) Receive

13_give_data

Synopsis
extern void 13_give_data();

Description

The 13_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:

AYER: 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
AYER: 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:
       { 13_give_data(); }
AYER: 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 (PROTOCOL) WINDOW NOT FULL or RCV RR (PROTOCOL) 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 && (!3_window_full(0xfe) != 0) }
ACTIONS: SEND DATA "(FOX)"

13_window_empty

Synopsis
extern unsigned char 13_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 (PROTOCOL) WINDOW NOT_EMPTY or RCV RR (PROTOCOL) 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  
  ACTIONS:  
  {  
    resend_packet(0xfe, 0);  
  }  
  NEXT_STATE: recover  

STATE: recover  
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 (PROTOCOL) 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 
63.3(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 63.3(A).

The third parameter is the offset from the beginning of the buffer to the start of 
the service data unit. See See Section 63.3(A).

The fourth parameter is a pointer to the packet structure to be sent. For a 
description of send_packet_structure see Table 72-1.
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.

```c
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 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[] = "FOXYJ"
static char facilities_string[] = "00000000"

struct send_packet_structure transmit_packet = {0, 0x13, 0x13, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 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: GIVEDATA

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};
```
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 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

The structure `send_frame_structure` defines the format of transmitted SDLC frames. See Table 73-1. Use this structure to send frames via the `send_frame` routine in emulate mode. See Section 73.3(B). Each variable in the structure relates to some softkey selection or user entry in the SEND action.

### 73.2 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 73-2 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 73-1**

SDLC Structures

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Structure Name:</strong> send_frame_structure</td>
<td>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};</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>addr_type</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>frame_type</td>
<td>(The codes for frame_type are the same as for the SDLC-variable rcvd_frame_type.)</td>
<td></td>
</tr>
<tr>
<td>nr_type</td>
<td>0</td>
<td>auto value</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>received ns plus 1</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>last nr sent</td>
</tr>
<tr>
<td>ns_type</td>
<td>0</td>
<td>auto value</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>skip</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>last nr received</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>value</td>
</tr>
<tr>
<td>p_f_type</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>loopback</td>
</tr>
<tr>
<td>bcc_type</td>
<td>0</td>
<td>default (bad bcc)</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>good bcc</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>bad bcc</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>abort</td>
</tr>
<tr>
<td>addr_value</td>
<td>00-ff/0-255</td>
<td>(actual value of the control byte)</td>
</tr>
<tr>
<td>cntri_byte</td>
<td>(actual value of the control byte)</td>
<td></td>
</tr>
<tr>
<td>nr_value</td>
<td>0-7 (MOD 8)</td>
<td>If nr_type = 1</td>
</tr>
<tr>
<td>ns_value</td>
<td>0-7 (MOD 8)</td>
<td>If ns_type = 3</td>
</tr>
<tr>
<td>Type</td>
<td>Variable</td>
<td>Value (hex/decimal)</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>extern event</td>
<td>dte_frame</td>
<td></td>
</tr>
<tr>
<td>extern event</td>
<td>dce_frame</td>
<td></td>
</tr>
<tr>
<td>extern event</td>
<td>dte_good_bcc</td>
<td></td>
</tr>
<tr>
<td>extern event</td>
<td>dce_good_bcc</td>
<td></td>
</tr>
<tr>
<td>extern event</td>
<td>dte_bad_bcc</td>
<td></td>
</tr>
<tr>
<td>extern event</td>
<td>dce_bad_bcc</td>
<td></td>
</tr>
<tr>
<td>extern event</td>
<td>dte_abort</td>
<td></td>
</tr>
<tr>
<td>extern event</td>
<td>dce_abort</td>
<td></td>
</tr>
<tr>
<td>extern event</td>
<td>rcvd_frame</td>
<td></td>
</tr>
<tr>
<td>extern event</td>
<td>invalid_frame</td>
<td></td>
</tr>
<tr>
<td>extern event</td>
<td>l2_T1</td>
<td></td>
</tr>
<tr>
<td>extern event</td>
<td>bcc_error</td>
<td></td>
</tr>
<tr>
<td>extern event</td>
<td>nr_error</td>
<td></td>
</tr>
<tr>
<td>extern event</td>
<td>ns_error</td>
<td></td>
</tr>
</tbody>
</table>
### Table 73-2 (continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern event</td>
<td>frame_sent</td>
<td></td>
<td>True when frame is passed down to Layer 1. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_frame_addr</td>
<td>00-ff/0-255</td>
<td>Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_frame_type</td>
<td>(same as rcvd_frame_type—Line Setup configured for emulate or monitor mode)</td>
<td></td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_frame_cntrl_byte_1</td>
<td>(actual value of control byte—Line Setup configured for emulate or monitor mode)</td>
<td></td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_frame_pf</td>
<td>0/10 16</td>
<td>Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_frame_bcc_type</td>
<td>1/2/3</td>
<td>good/bad/abort Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>rcvd_frame_addr</td>
<td>00-ff/0-255</td>
<td>Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>rcvd_frame_cntrl_byte_1</td>
<td>(actual value of control byte—Line Setup configured for emulate mode only)</td>
<td></td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>rcvd_frame_pf</td>
<td>0/10 16</td>
<td>pf=0/1 pf=1 Line Setup configured for emulate mode only.</td>
</tr>
</tbody>
</table>
### Table 73-2 (continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern volatile const unsigned char</td>
<td>rcvd_frame_bcc_type</td>
<td>1</td>
<td>good</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>bad</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>abort</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>rcvd_frame_nr</td>
<td>0-7 (MOD 8)</td>
<td>Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>rcvd_frame_ns</td>
<td>0-7 (MOD 8)</td>
<td>Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>rcvd_frame_buff_seg</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>rcvd_frame_edu_offset</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>rcvd_frame_edu_size</td>
<td></td>
<td>Size of service data unit in a received frame. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>l2_current_window_edge</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>l2_lower_window_edge</td>
<td></td>
<td>see l2_current_window_edge</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>l2_upper_window_edge</td>
<td></td>
<td>see l2_current_window_edge</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>l2_resend_edge</td>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>
INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

Table 73-2 (continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern unsigned char</td>
<td>l2_enhance</td>
<td>0</td>
<td>normal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>reverse</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>low</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>reverse low</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>blink</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
<td>reverse blink</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12/18</td>
<td>blink low</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern unsigned char</td>
<td>l2_suppress</td>
<td>0</td>
<td>off</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>on</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Line Setup configured for emulate or monitor mode.</td>
</tr>
</tbody>
</table>

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:

```c
{ 
  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:

```c
{ 
  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:

```c
{ 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:

```c
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 73-2 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:

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

1. Frame characteristics. All status variables in Table 73-2 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 73-2 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 63.1 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).

```c
{ 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: \( \text{i2}_{\text{current}} \text{window}_{\text{edge}} = \text{i2}_{\text{upper}} \text{window}_{\text{edge}} \)
- WINDOW EMPTY: \( \text{i2}_{\text{current}} \text{window}_{\text{edge}} = \text{i2}_{\text{lower}} \text{window}_{\text{edge}} \)
- WINDOW NOT FULL: \( \text{i2}_{\text{current}} \text{window}_{\text{edge}} \neq \text{i2}_{\text{upper}} \text{window}_{\text{edge}} \)
- WINDOW NOT EMPTY: \( \text{i2}_{\text{current}} \text{window}_{\text{edge}} \neq \text{i2}_{\text{lower}} \text{window}_{\text{edge}} \)
- MORE_TO_RESEND: \( \text{i2}_{\text{resend}_{\text{edge}}} \neq \text{i2}_{\text{lower}} \text{window}_{\text{edge}} \)
- NO_MORE_TO_RESEND: \( \text{i2}_{\text{resend}_{\text{edge}}} = \text{i2}_{\text{lower}} \text{window}_{\text{edge}} \)

(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 \( \text{i2}_{\text{enhance}} \) or \( \text{i2}_{\text{suppress}} \). The values are listed in Table 73-2. 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:
{
12_enhance = 1;
}
CONDITIONS: RCV INVALID
ACTIONS:
{
12_suppress = 1;
}

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

CONDITIONS: RCV INFO
{
12_enhance == 1
}
ACTIONS:
{
12_enhance = 0;
}
or an ACTIONS block:

CONDITIONS: RCV INFO
ACTIONS:
{
if(12_enhance == 1)
12_enhance = 0;
}

73.3 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 12_give_data."

(A) Receive

12_give_data

Synopsis
extern void 12_give_data();

Description
The 12_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 SDLC personality package is loaded at Layer 2, enter the following program:

LAYER: 2
STATE: datalnk
CONDITIONS: DL_CONNECT REQ
ACTIONS: DL_CONNECT CONF
CONDITIONS: DL_DATA REQ
ACTIONS: SEND INFO "DL_DATA"
CONDITIONS: RCV INFO
ACTIONS:
{
12_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, 1, or 2 (for loopback).

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 (PROTOCOL) 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 and supervisory frames is reset to zero and the transmit window is cleared. The softkey equivalent of this routine is the (PROTOCOL) 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 ctrl_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 63.3(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 63.3(A).

The third parameter is the offset from the beginning of the buffer to the start of the service data unit. See Section 63.3(A).

The fourth parameter is a pointer to the frame structure to be sent. For a description of `send_frame_structure` see Table 73-1.

**Example**

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

```c
{  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 pf_type;
    unsigned char bcc_type;
    unsigned char addr_value;
    unsigned char ctrl_byte;
    unsigned char nr_value;
    unsigned char ns_value;
  };  
  struct send_frame_structure transmit_frame;
  static char transmit_string[ ] = "\(FOX\)";
}
```

**STATE: send_a_frame**

**CONDITIONS: KEYBOARD**

**ACTIONS:**

```c
  _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);
}
74 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 73.

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.

74.1 Structures

Use the SDLC send_frame_structure shown in Table 73-1.

74.2 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 73.2(A).

(B) Status Variables

All SNA variables in Table 74-1 are status variables. Also refer to the SDLC status variables listed in Table 73-2.

There are no softkey tokens on the spreadsheet that are equivalent to the SNA variables listed in Table 74-1. 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 74-1
SNA Variables†

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern volatile unsigned short</td>
<td>m_packet_length</td>
<td></td>
<td>Length of the packet, including the transmission and request/response headers. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_packet_fid_type</td>
<td>0-ffff10-65535</td>
<td>Transmission Header:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FID0; TH 10 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FID1; TH 10 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FID2; TH 6 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FID3; TH 2 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FID4; TH 26 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FIDF; TH 26 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned short</td>
<td>m_packet_daf</td>
<td>0-ffffff0-65535</td>
<td>Destination address field—2 bytes in FID0 and FID1; 1 byte in FID2. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned short</td>
<td>m_packet_def</td>
<td>0-ffffff0-65535</td>
<td>Destination element field—2 bytes; FID4 only. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned long</td>
<td>m_packet_dsaq</td>
<td>0-ffffffff100-65535</td>
<td>Destination subarea address field—4 bytes; FID4 only. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_packet_isid</td>
<td></td>
<td>Local Session Identification:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FID3 only</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SSCP-PU session</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SSCP-LU session</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>LU-LU session</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned short</td>
<td>m_packet_oaf</td>
<td>00-ff10-255</td>
<td>Origin address field—2 bytes in FID0 and FID1; 1 byte in FID2. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned short</td>
<td>m_packet_eof</td>
<td>00-ff10-255</td>
<td>Origin element field—2 bytes; FID4 only. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned long</td>
<td>m_packet_osaf</td>
<td>0-ffffff100-65535</td>
<td>Origin subarea address field—4 bytes; FID4 only. Line Setup configured for emulate or monitor mode.</td>
</tr>
</tbody>
</table>

† Refer to Table 73-2 for SDLC variables.
<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern volatile unsigned char *</td>
<td>th_ptr</td>
<td></td>
<td>Pointer for the transmission header; begins at the byte containing FID type. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_packet_ru_category</td>
<td>0</td>
<td>Function Management Data (FMD)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20/32</td>
<td>Network Control (NC)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>40/64</td>
<td>Data Flow Control (DFC)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>60/96</td>
<td>Session Control (SC)</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_packet_fl</td>
<td>0</td>
<td>User data without header in RU</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>In LU-LU frame, indicates header follows the RH. In SC, NC, or DFC RU, indicates a formatted RU beginning with a request code.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_packet_rri</td>
<td>0</td>
<td>Function Management Data (FMD)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>80/128</td>
<td>Network Control (NC)</td>
</tr>
<tr>
<td>extern volatile unsigned char</td>
<td>m_packet_rtl</td>
<td>0</td>
<td>Data Flow Control (DFC)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10/16</td>
<td>Session Control (SC)</td>
</tr>
<tr>
<td>extern volatile unsigned char</td>
<td>m_packet_sdl</td>
<td>0</td>
<td>Sense Data Indicator:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>sense data not included</td>
</tr>
<tr>
<td>extern volatile unsigned char *</td>
<td>rh_ptr</td>
<td></td>
<td>Pointer for the request/response header; begins at the byte containing the request/response indicator. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile unsigned char *</td>
<td>ru_ptr</td>
<td></td>
<td>Pointer for the request/response unit; begins at the first byte in the unit. Line Setup configured for emulate or monitor mode.</td>
</tr>
</tbody>
</table>

Table 74-1 (continued)
1. **Info frame characteristics.** Most status variables in Table 74-1 contain an \textit{m}_\textunderscore prefix, indicating that they are valid in emulate or monitor mode. Some variables are associated with the transmission header: \textit{m\textunderscore packet\textunderscore fid\textunderscore type}, \textit{m\textunderscore packet\textunderscore daf}, \textit{m\textunderscore packet\textunderscore def}, \textit{m\textunderscore packet\textunderscore dsaf}, \textit{m\textunderscore packet\textunderscore lsid}, \textit{m\textunderscore packet\textunderscore oaf}, \textit{m\textunderscore packet\textunderscore of}, and \textit{m\textunderscore packet\textunderscore osaf}. Other variables are associated with the request/response header: \textit{m\textunderscore packet\textunderscore ru\textunderscore category}, \textit{m\textunderscore packet\textunderscore fi}, \textit{m\textunderscore packet\textunderscore rri}, \textit{m\textunderscore packet\textunderscore rti}, and \textit{m\textunderscore packet\textunderscore sdi}.

2. **Pointers.** There are three pointers to SNA fields. \textit{th\textunderscore ptr} points to first byte of the transmission header, \textit{rh\textunderscore ptr} points to the first byte of request/response header, and \textit{ru\textunderscore 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 \textit{l2\_enhance} or \textit{l2\_suppress}. The values are listed in Table 73-2. To assign a value to either of these variables, place the statement in an \textit{ACTIONS} block. For example, display only Info frames with FID2 transmission headers. Of these, display frames with sense data in reverse-video.

**CONDITIONS:**

\{ \\
\quad dte\_frame \&\& (m\_frame\_type == 0) \&\& (m\_packet\_fid\_type != 2) \\
\}

**ACTIONS:**

\{ \\
\quad l2\_suppress = 1; \\
\}

**CONDITIONS:**

\{ \\
\quad dte\_frame \&\& (m\_frame\_type == 0) \&\& (m\_packet\_fid\_type == 2) \&\& (m\_packet\_sdi == 4) \\
\}

**ACTIONS:**

\{ \\
\quad l2\_enhance = 1; \\
\}

Check the value of these display-control variables in a \textit{CONDITIONS} block.

**CONDITIONS:**

\{ \\
\quad dte\_frame \&\& (m\_frame\_type == 0) \&\& (m\_packet\_fid\_type != 2) \&\& (l2\_suppress == 0) \\
\}

**ACTIONS:**

\{ \\
\quad l2\_suppress = 1; \\
\}
or an ACTIONS block:

CONDITIONS:
{
    dtc_frame && (m_frame_type == 0) && (m_packet_id_type != 2)
}
ACTIONS:
{
    if(l2_suppress == 0)
        l2_suppress = 1;
}

74.3 Routines

There are no routines associated exclusively with SNA. Use the SDLC routines discussed in Section 73.3. To send a frame including SNA protocol, for example, include a transmit_string of SNA data in the send_frame routine.
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.

### 75.1 Structures

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

### 75.2 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 75-1.

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

```c

    LAYER: 1
    STATE: count_all_bad_dte_bccs
    CONDITIONS: DTE_BAD_BCC
    ACTIONS: COUNTER t_bd_bccs INC

    CONDITIONS:
    {  fevar_bd_bcc2_td
     }  ACTIONS: COUNTER t_bd_bccs INC
```

### 75.3 Routines

There are no routines associated exclusively with DDCMP.
### Table 75-1  
**DDCMP Variables**

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern fast_event</td>
<td>fevar_gd_bcc_rd</td>
<td></td>
<td>True when a good header BCC is received on RD. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern fast_event</td>
<td>fevar_gd_bcc_td</td>
<td></td>
<td>True when a bad header BCC is received on TD. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern fast_event</td>
<td>fevar_bd_bcc_rd</td>
<td></td>
<td>True when a good data BCC is received on RD. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern fast_event</td>
<td>fevar_gd_bcc2_td</td>
<td></td>
<td>True when a bad data BCC is received on TD. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern fast_event</td>
<td>fevar_gd_bcc2_rd</td>
<td></td>
<td>True when a good data BCC is received on RD. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern fast_event</td>
<td>fevar_bd_bcc2_td</td>
<td></td>
<td>True when a bad data BCC is received on TD. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern fast_event</td>
<td>fevar_bd_bcc2_rd</td>
<td></td>
<td>True when a good data BCC is received on RD. Line Setup configured for emulate or monitor mode.</td>
</tr>
</tbody>
</table>
76 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 10. 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 48.5.

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).

76.1 Structures

Use the structure xmit_list, shown in Table 76-1, when transmitting on the D channel via the send_d_frame routine. Refer to send_d_frame in Section 76.3 for an example of how to use this structure.

<table>
<thead>
<tr>
<th>xmit_list</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structure Name: xmit_list</td>
<td>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. pointer to the location of the transmit string—the transmit string is declared separately</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unsigned char *</td>
<td>string</td>
<td>0-ffff/0-65535</td>
<td>length of the transmit string</td>
</tr>
<tr>
<td>unsigned short</td>
<td>string_length</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### 76.2 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 76-2.

**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>
<thead>
<tr>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>d_dte_frame</code></td>
<td></td>
<td>True when a DTE frame is detected on the D channel. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td><code>d_dce_frame</code></td>
<td></td>
<td>True when a DCE frame is detected on the D channel. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td><code>d_rcv_frame</code></td>
<td></td>
<td>True when a frame is received on the D channel. Line Setup configured for emulate mode only.</td>
</tr>
</tbody>
</table>

### 76.3 Routines

There are two routines associated with the ISDN_D package: `send_d_frame` and `send_d_frame İl`. 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 İl`.”
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:

```c
{ 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, 5, &message, sizeof(message) - 1};
};
```
INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

**Synopsis**

extern void send_message(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_message. Refer to Section 63.3(A) for a description of the _get_il_msg_buff, _start_il_buff_list, and _insert_il_buff_list_cnt_ routines.

```c
{ unsigned short il_buffer_number;
  unsigned short relay_baton;
  unsigned short data_start_offset;
  unsigned char message [] = "a\0x00b\0x00\0x00\0x00\0FOX)";
}
```
LAYER: 1
STATE: send_message
CONDITIONS: KEYBOARD
ACTIONS:
{
    set_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, &message[0],
                            (sizeof(message) - 1));
    send_d_frame_il(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.
77 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 39 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.

77.1 Structures

The structure *send_frame_structure* defines the format of transmitted LAPD frames. See Table 77-1. Use this structure to send frames via the *send_frame* routine in emulate mode. See Section 77.3(B). Each variable in the structure relates to some softkey selection or user entry in the SEND action.

77.2 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 77-2 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:

```plaintext
CONDITIONS:
{
  dte_bad_bcc || dce_bad_bcc
}
ACTIONS: COUNTER bad_bcc INC
```
**Table 77-1**

LAPD Structures

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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
```

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
```

- **sapL_type**
  - 1: no other value valid—indicates a value is given
- **tel_type**
  - 1: no other value valid—indicates a value is given
- **cr_type**
  - 0: none
  - 1: 0
  - 2: loopback
- **frame_type**
  - (The codes for frame_type are the same as for the LAPD-variable rcvd_frame_type.)
  - 0: auto value
  - 1: received ns plus 1
  - 2: last nr sent
- **nr_type**
  - 0: auto
  - 1: skip
  - 2: last nr received
  - 3: value
- **ns_type**
  - 0: auto
  - 1: skip
  - 2: last nr received
  - 3: value
- **p_f_type**
  - 0: 0
  - 1: 1
  - 2: loopback
- **bcc_type**
  - 0: default (bad bcc)
  - 1: good bcc
  - 2: bad bcc
  - 3: abort
- **sapL_value**
  - 00-31/0-63
- **tel_value**
  - 00-7F/0-127
- **cntrl_byte**
  - (actual value of the control byte)
- **nr_value**
  - 0-7 (MOD 8) if nr_type = 1
- **ns_value**
  - 0-7 (MOD 8) if ns_type = 3
<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern event</td>
<td>dte_frame</td>
<td>True when a DTE frame is detected. Line Setup configured for emulate or monitor mode.</td>
<td></td>
</tr>
<tr>
<td>extern event</td>
<td>dce_frame</td>
<td>True when a DCE frame is detected. Line Setup configured for emulate or monitor mode.</td>
<td></td>
</tr>
<tr>
<td>extern event</td>
<td>dte_good_bcc</td>
<td>True when a good BCC is calculated for a DTE frame. Line Setup configured for emulate or monitor mode.</td>
<td></td>
</tr>
<tr>
<td>extern event</td>
<td>dce_good_bcc</td>
<td>True when a good BCC is calculated for a DCE frame. Line Setup configured for emulate or monitor mode.</td>
<td></td>
</tr>
<tr>
<td>extern event</td>
<td>dte_bad_bcc</td>
<td>True when a bad BCC is calculated for a DTE frame. Line Setup configured for emulate or monitor mode.</td>
<td></td>
</tr>
<tr>
<td>extern event</td>
<td>dce_bad_bcc</td>
<td>True when a bad BCC is calculated for a DCE frame. Line Setup configured for emulate or monitor mode.</td>
<td></td>
</tr>
<tr>
<td>extern event</td>
<td>dte_abort</td>
<td>True when an abort is detected for a DTE frame. Line Setup configured for emulate or monitor mode.</td>
<td></td>
</tr>
<tr>
<td>extern event</td>
<td>dce_abort</td>
<td>True when an abort is detected for a DCE frame. Line Setup configured for emulate or monitor mode.</td>
<td></td>
</tr>
<tr>
<td>extern event</td>
<td>rcvd_frame</td>
<td>True when a frame is received. Line Setup configured for emulate mode only.</td>
<td></td>
</tr>
<tr>
<td>extern event</td>
<td>invalid_frame</td>
<td>True when an invalid frame is detected. Line Setup configured for emulate mode only.</td>
<td></td>
</tr>
<tr>
<td>extern event</td>
<td>l2_T1</td>
<td>True when the T1 timeout-timer has expired. Line Setup configured for emulate mode only.</td>
<td></td>
</tr>
<tr>
<td>extern event</td>
<td>bcc_error</td>
<td>True when a BCC error is detected. Line Setup configured for emulate mode only.</td>
<td></td>
</tr>
<tr>
<td>extern event</td>
<td>nr_error</td>
<td>True when an N(R) error is detected in a received INFO or supervisory frame. Line Setup configured for emulate mode only.</td>
<td></td>
</tr>
<tr>
<td>extern event</td>
<td>ns_error</td>
<td>True when an N(S) error is detected in a received INFO frame. Line Setup configured for emulate mode only.</td>
<td></td>
</tr>
</tbody>
</table>
Table 77-2 (continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern event</td>
<td>frame_sent</td>
<td></td>
<td>True when frame is passed down to Layer 1. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_frame_addr_sapi</td>
<td>00-3f/0-63</td>
<td>Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_frame_addr_tei</td>
<td>00-7f/0-127</td>
<td>Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_frame_addr_cr</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_frame_type</td>
<td>(same as rcvd_frame_type—Line Setup configured for emulate or monitor mode)</td>
<td></td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_frame_cntl_byte_1</td>
<td>(actual value of control byte—Line Setup configured for emulate or monitor mode)</td>
<td></td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_frame_pf</td>
<td>0</td>
<td>pf=0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10/16</td>
<td>pf=1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_frame_bcc_type</td>
<td>1</td>
<td>good.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>bad</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>abort</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_frame_nr</td>
<td>0-7 (MOD 8)</td>
<td>Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_frame_ns</td>
<td>0-7 (MOD 8)</td>
<td>Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>rcvd_frame_addr_sapi</td>
<td>00-3f/0-63</td>
<td>Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>rcvd_frame_addr_tei</td>
<td>00-7f/0-127</td>
<td>Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>rcvd_frame_addr_cr</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>loopback</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>rcvd_frame_type</td>
<td>0</td>
<td>info</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>rr</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
<td>rej</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2f/37</td>
<td>sabm</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6f/111</td>
<td>sabme</td>
</tr>
<tr>
<td></td>
<td></td>
<td>43/67</td>
<td>disc</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1/15</td>
<td>dm</td>
</tr>
<tr>
<td></td>
<td></td>
<td>f/15</td>
<td>sarm</td>
</tr>
<tr>
<td></td>
<td></td>
<td>63/99</td>
<td>ua</td>
</tr>
<tr>
<td></td>
<td></td>
<td>67/103</td>
<td>sio</td>
</tr>
<tr>
<td></td>
<td></td>
<td>87/135</td>
<td>frmr</td>
</tr>
<tr>
<td></td>
<td></td>
<td>e7/224</td>
<td>s1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ff/255</td>
<td>other</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ff/255</td>
<td>unknown</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>Type</td>
<td>Variable</td>
<td>Value (hex/decimal)</td>
<td>Meaning</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>---------------------------------</td>
<td>--------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>rcvd_frame_cntrl_byte_1</td>
<td>(actual value of control byte—Line Setup configured for emulate mode only)</td>
<td></td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>rcvd_frame_pf</td>
<td>0</td>
<td>pf=0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10/16</td>
<td>pf=1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>rcvd_frame_bcc_type</td>
<td>1</td>
<td>good</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>bad</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>abort</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>rcvd_frame_nr</td>
<td>0-7 (MOD 8)</td>
<td>Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>rcvd_frame_ns</td>
<td>0-7 (MOD 8)</td>
<td>Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>rcvd_frame_buff_seg</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>rcvd_frame_sdu_offset</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>rcvd_frame_sdu_size</td>
<td></td>
<td>Size of service data unit in a received frame. Line Setup configured for emulate mode only.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>l2_current_window_edge</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>l2_lower_window_edge</td>
<td></td>
<td>see l2_current_window_edge</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>l2_upper_window_edge</td>
<td></td>
<td>see l2_current_window_edge</td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>l2_resend_edge</td>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>
Table 77-2 (continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern unsigned char</td>
<td>l2_enhance</td>
<td>0</td>
<td>normal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>reverse</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>low</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>reverse low</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>blink</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
<td>reverse blink</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12/18</td>
<td>blink low</td>
</tr>
<tr>
<td>extern unsigned char</td>
<td>l2_suppress</td>
<td>0</td>
<td>off</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>on</td>
</tr>
</tbody>
</table>

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:

```c
{  
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:

```c
{  
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:

\[
\{ 
\text{rcvd\_frame} \&\& (\text{rcvd\_frame\_type} == 0) 
\}
\]

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:
\[
\{ 
\text{rcvd\_frame} 
\}
\]

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 77-2 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:
\[
\{ 
\text{rcvd\_frame} \&\& (\text{rcvd\_frame\_type} == 0) 
\}
\]

1. **Frame characteristics.** All status variables in Table 77-2 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\_l, m\_frame\_pf, m\_frame\_bcc\_type, m\_frame\_nr, and m\_frame\_ns.

All status variables in Table 77-2 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\_l, 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 63.1 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).

```c
{ 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 77-2. 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:
{ 12_enhance = 1; }

CONDITIONS: RCV INVALID
ACTIONS:
{ 12_suppress = 1; }

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

CONDITIONS: RCV INFO
{ 12_enhance == 1 }
ACTIONS:
{ 12_enhance = 0; }

or an ACTIONS block:

CONDITIONS: RCV INFO
ACTIONS:
{ if(12_enhance == 1) 12_enhance = 0; }

77.3 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:

```c
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

```c
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 (PROTOCOL) 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, 1, or 2 (for loopback).

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 and supervisory frames is reset to zero and the
transmit window is cleared. The softkey equivalent of this routine is the
(PROTOCOL) 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(ii_buffer_number, relay_baton, data_start_offset, transmit_frame_ptr);
unsigned short ii_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 ctrl_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 63.3(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 63.3(A).

The third parameter is the offset from the beginning of the buffer to the start of the service data unit. See Section 63.3(A).

The fourth parameter is a pointer to the frame structure to be sent. For a description of send_frame_structure see Table 77-1.

Example

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

```c
{  
  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 tel_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 tel_value;
    unsigned char cntrl_byte;
    unsigned char nr_value;
    unsigned char ns_value;
  };  
  struct send_frame_structure transmit_frame;
  static char transmit_string[10] = "(FOX)";
}
```

```c
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);
  }
```
78 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 38 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.

78.1 Structures

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

78.2 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 78-1. 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:

```c
{ 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:

```c
CONDITIONS:
{
    dte_packet
}
```
Table 78-1
Q.931 Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern event</td>
<td>dte_packet</td>
<td></td>
<td>True when a DTE packet is detected. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern event</td>
<td>dce_packet</td>
<td></td>
<td>True when a DCE packet is detected. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_packet_bcc_type</td>
<td>1</td>
<td>good</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>bad</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>abort</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_prot_disc</td>
<td>00-ff/0-255</td>
<td>Actual value of protocol discriminator—should be 8. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_call_ref_flag</td>
<td>0</td>
<td>origination side</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>destination side</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_message_type_defined</td>
<td>0</td>
<td>Actual value received is not a defined value for a LAPD message type.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Actual value received is one of the following valid values for a LAPD message type:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 alerting</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2 call proceeding</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5 setup</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>7 connect</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>d/13 setup ack</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>f/15 connect ack</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>20/32 user info</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>21/33 suspend rej</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>22/34 resume rej</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>25/37 suspend</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>26/38 resume</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2d/45 suspend ack</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2e/46 resume ack</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>40/64 detach</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>45/69 disconnect</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>48/72 detach ack</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4d/77 release</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5a/90 release complete</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>60/96 cancel</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>62/98 facility</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>64/100 register</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>66/104 cancel ack</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>6a/106 facility ack</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>6c/108 register</td>
</tr>
</tbody>
</table>

(m_message_type_defined continued on next page)
<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_message_type</td>
<td>00-ff/0-255</td>
<td>Actual value of the message-type byte. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_call_ref_len</td>
<td>0-15</td>
<td>Length of the call-reference value field. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_info_element_len</td>
<td></td>
<td>Length of information element field. The total includes all information elements. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_ptr_to_call_ref</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_ptr_to_info_element</td>
<td></td>
<td>Pointer to the information element field. Begins at the first byte after the message-type byte. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern unsigned char</td>
<td>l3_enhance</td>
<td>0</td>
<td>normal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>reverse</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>low</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>reverse low</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>blink</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
<td>reverse blink</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12/18</td>
<td>blink low</td>
</tr>
<tr>
<td></td>
<td>l3_suppress</td>
<td>0</td>
<td>off</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>on</td>
</tr>
</tbody>
</table>

Table 78-1 (continued)
(B) Status Variables

Status variables are those in Table 78-1 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:

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

1. Packet characteristics. All status variables in Table 78-1 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 78-1. 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:

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

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

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

```c
CONDITIONS: DTE INFO
{
  l3_enhance == 1
}
ACTIONS:
{
  l3_enhance = 0;
}
```
or an ACTIONS block:

CONDITIONS: DTE INFO
ACTIONS:
{
  if(\_3\_enhance == 1)
  \_3\_enhance = 0;
}

78.3 Routines

There are no routines associated exclusively with Q.931.
79 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 79-2 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 79-1.

The Layer 2 variables approximate SS#7 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 is sufficient for identifying its related spreadsheet token. When this is not the case, the information is provided below.

79.1 Structures

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

79.2 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 79-1 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 79-1
SS#7 Layer 2 Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern event</td>
<td>dte_frame</td>
<td></td>
<td>True when a non-suppressed DTE frame is detected. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern event</td>
<td>dce_frame</td>
<td></td>
<td>True when a non-suppressed DCE frame is detected. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern event</td>
<td>dte_good_bcc</td>
<td></td>
<td>True when a non-suppressed good BCC is calculated for a DTE frame. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern event</td>
<td>dce_good_bcc</td>
<td></td>
<td>True when a non-suppressed good BCC is calculated for a DCE frame. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern event</td>
<td>dte_bad_bcc</td>
<td></td>
<td>True when a bad BCC is calculated for a DTE frame. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern event</td>
<td>dce_bad_bcc</td>
<td></td>
<td>True when a bad BCC is calculated for a DCE frame. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern event</td>
<td>dte_abort</td>
<td></td>
<td>True when an abort is detected for a DTE frame. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern event</td>
<td>dce_abort</td>
<td></td>
<td>True when an abort is detected for a DCE frame. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_unit_type</td>
<td>1</td>
<td>Fill-In Signal Unit (FI)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>Link Status Signal Unit (LSU)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>Message Signal Unit (MSU)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_bib</td>
<td>0</td>
<td>non-zero</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_fib</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Table 79-1 (continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_li</td>
<td>0</td>
<td>FI</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1-2</td>
<td>LSU</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3-3f/63</td>
<td>MSU</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_so0</td>
<td>0</td>
<td>out of alignment</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>normal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>emergency</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>out of service</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>processor out</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>busy</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_frame_bcc_type</td>
<td>1</td>
<td>good bcc</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>bad bcc</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>abort</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern unsigned char</td>
<td>l2 Enhance</td>
<td>0</td>
<td>normal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>reverse</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>reverse low</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>blink</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
<td>reverse blink</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12/18</td>
<td>blink low</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern unsigned char</td>
<td>l2_suppress</td>
<td>0</td>
<td>off</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>on</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Line Setup configured for emulate or monitor mode.</td>
</tr>
</tbody>
</table>

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 79-1 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 79-1 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 79-2 are also status variables, valid in either emulate or monitor mode. Any of the Layer 2 event variables in Table 79-1 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 79-1. 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 79-2
SS#7 Layer 1 Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern unsigned short</td>
<td>dte_frames_suppressed</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>extern unsigned short</td>
<td>dce_frames_suppressed</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>extern unsigned short</td>
<td>dte_flags</td>
<td></td>
<td>Number of DTE flags received since the last non-suppressed frame. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern unsigned short</td>
<td>dce_flags</td>
<td></td>
<td>Number of DCE flags received since the last non-suppressed frame. Line Setup configured for emulate or monitor mode.</td>
</tr>
</tbody>
</table>

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

```c
CONDITIONS:
{
    dte_frame && (m_unit_type == 2) && (m_so0 == 2) && (12_enhance == 0)
}
ACTIONS:
{
    12_enhance = 1;
}
```

or an ACTIONS block:

```c
CONDITIONS:
{
    dte_frame && (m_unit_type == 2) && (m_so0 == 2)
}
ACTIONS:
{
    if(12_enhance == 0)
    12_enhance = 1;
}
```

### 79.3 Routines

There are no routines associated exclusively with SS#7.
80 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 43 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.

80.1 Structures

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

80.2 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 80-1. 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:

```c
{ 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:

```c
CONDITIONS:
{
 dte_packet
}
```
Table 80-1
SS#7 Layer 3 Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern event</td>
<td>dce_packet</td>
<td></td>
<td>True when a DCE packet is detected. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern event</td>
<td>dte_packet</td>
<td></td>
<td>True when a DTE packet is detected. Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_sio_nl</td>
<td>0</td>
<td>International 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>40/64</td>
<td>International 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>80/128</td>
<td>national 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c0/192</td>
<td>national 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_sio_priority</td>
<td>0</td>
<td>priority=0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10/16</td>
<td>priority=1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20/32</td>
<td>priority=2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>30/48</td>
<td>priority=3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_sio_si</td>
<td>0-7</td>
<td>User Part:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>netm</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>ntr</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>nts</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>sccp</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>tup</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>lsdn</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>dup0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
<td>dup1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8-18-15</td>
<td>spare</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_code_type</td>
<td></td>
<td>Test headers:†</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>(high 4 bits not defined)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>ltm</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>lta</td>
</tr>
</tbody>
</table>

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

(m_code_type continued on next page)
### Table 80-1 (continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(m_code_type continued)</td>
<td>TUP headers:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>anu</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>reserved</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>11/17</td>
<td>iam</td>
<td>11/17</td>
<td></td>
</tr>
<tr>
<td>12/18</td>
<td>gsm</td>
<td>12/18</td>
<td></td>
</tr>
<tr>
<td>13/19</td>
<td>grq</td>
<td>13/19</td>
<td></td>
</tr>
<tr>
<td>14/20</td>
<td>acm</td>
<td>14/20</td>
<td></td>
</tr>
<tr>
<td>15/21</td>
<td>sec</td>
<td>15/21</td>
<td></td>
</tr>
<tr>
<td>16/22</td>
<td>anc</td>
<td>16/22</td>
<td></td>
</tr>
<tr>
<td>17/23</td>
<td>rig</td>
<td>17/23</td>
<td></td>
</tr>
<tr>
<td>18/24</td>
<td>mgb</td>
<td>18/24</td>
<td></td>
</tr>
<tr>
<td>19/25</td>
<td>cfm</td>
<td>19/25</td>
<td></td>
</tr>
<tr>
<td>21/33</td>
<td>fcm</td>
<td>21/33</td>
<td></td>
</tr>
<tr>
<td>24/36</td>
<td>chg</td>
<td>24/36</td>
<td></td>
</tr>
<tr>
<td>25/37</td>
<td>ggc</td>
<td>25/37</td>
<td></td>
</tr>
<tr>
<td>26/38</td>
<td>ann</td>
<td>26/38</td>
<td></td>
</tr>
<tr>
<td>27/39</td>
<td>bio</td>
<td>27/39</td>
<td></td>
</tr>
<tr>
<td>28/40</td>
<td>mba</td>
<td>28/40</td>
<td></td>
</tr>
<tr>
<td>29/41</td>
<td>cpm</td>
<td>29/41</td>
<td></td>
</tr>
<tr>
<td>31/49</td>
<td>sam</td>
<td>31/49</td>
<td></td>
</tr>
<tr>
<td>32/50</td>
<td>cot</td>
<td>32/50</td>
<td></td>
</tr>
<tr>
<td>35/53</td>
<td>nnc</td>
<td>35/53</td>
<td></td>
</tr>
<tr>
<td>36/54</td>
<td>cbk</td>
<td>36/54</td>
<td></td>
</tr>
<tr>
<td>37/55</td>
<td>bia</td>
<td>37/55</td>
<td></td>
</tr>
<tr>
<td>38/56</td>
<td>mgu</td>
<td>38/56</td>
<td></td>
</tr>
<tr>
<td>39/57</td>
<td>cpa</td>
<td>39/57</td>
<td></td>
</tr>
<tr>
<td>41/65</td>
<td>sao</td>
<td>41/65</td>
<td></td>
</tr>
<tr>
<td>42/66</td>
<td>ccf</td>
<td>42/66</td>
<td></td>
</tr>
<tr>
<td>45/69</td>
<td>adi</td>
<td>45/69</td>
<td></td>
</tr>
<tr>
<td>46/70</td>
<td>cif</td>
<td>46/70</td>
<td></td>
</tr>
<tr>
<td>47/71</td>
<td>ubl</td>
<td>47/71</td>
<td></td>
</tr>
<tr>
<td>48/72</td>
<td>muu</td>
<td>48/72</td>
<td></td>
</tr>
<tr>
<td>49/73</td>
<td>csv</td>
<td>49/73</td>
<td></td>
</tr>
<tr>
<td>55/85</td>
<td>cff</td>
<td>55/85</td>
<td></td>
</tr>
<tr>
<td>56/86</td>
<td>ran</td>
<td>56/86</td>
<td></td>
</tr>
<tr>
<td>57/87</td>
<td>uba</td>
<td>57/87</td>
<td></td>
</tr>
<tr>
<td>58/88</td>
<td>hgb</td>
<td>58/88</td>
<td></td>
</tr>
<tr>
<td>59/89</td>
<td>cvm</td>
<td>59/89</td>
<td></td>
</tr>
<tr>
<td>65/101</td>
<td>sbb</td>
<td>65/101</td>
<td></td>
</tr>
<tr>
<td>66/102</td>
<td>fot</td>
<td>66/102</td>
<td></td>
</tr>
<tr>
<td>67/103</td>
<td>ccr</td>
<td>67/103</td>
<td></td>
</tr>
<tr>
<td>68/104</td>
<td>hba</td>
<td>68/104</td>
<td></td>
</tr>
<tr>
<td>69/105</td>
<td>crn</td>
<td>69/105</td>
<td></td>
</tr>
<tr>
<td>75/117</td>
<td>unn</td>
<td>75/117</td>
<td></td>
</tr>
<tr>
<td>76/118</td>
<td>ccl</td>
<td>76/118</td>
<td></td>
</tr>
<tr>
<td>77/119</td>
<td>rsc</td>
<td>77/119</td>
<td></td>
</tr>
<tr>
<td>78/120</td>
<td>hgu</td>
<td>78/120</td>
<td></td>
</tr>
<tr>
<td>79/121</td>
<td>cli</td>
<td>79/121</td>
<td></td>
</tr>
<tr>
<td>85/133</td>
<td>los</td>
<td>85/133</td>
<td></td>
</tr>
<tr>
<td>88/136</td>
<td>hua</td>
<td>88/136</td>
<td></td>
</tr>
</tbody>
</table>
Table 80-1 (continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(m_code_type continued)</td>
<td></td>
<td>(TUP headers continued)</td>
<td></td>
</tr>
<tr>
<td>95/149</td>
<td>sst</td>
<td></td>
<td></td>
</tr>
<tr>
<td>98/152</td>
<td>grs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a5/165</td>
<td>acb</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a8/168</td>
<td>gra</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b5/181</td>
<td>dpn</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b8/184</td>
<td>sgb</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c5/197</td>
<td>mpr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c8/200</td>
<td>sba</td>
<td></td>
<td></td>
</tr>
<tr>
<td>d8/216</td>
<td>sgu</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e8/232</td>
<td>sua</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f5/245</td>
<td>eum</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f6/246</td>
<td>eam</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| ISDN headers: | |
| 1 | lam | |
| 2 | sam | |
| 3 | inr | |
| 4 | inf | |
| 5 | cot | |
| 6 | aom | |
| 8 | fot | |
| 9 | anm | |
| a/10 | ubm | |
| b/11 | rel | |
| d/12 | pau | |
| e/14 | res | |
| f/15 | risd | |
| 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 | cgua | |
| 1a/26 | cgba | |
| 1b/27 | cmr | |
| 1c/28 | cmc | |
| 1d/29 | cmo | |
| 1e/30 | rcm | |
| 1f/31 | far | |
| 20/32 | faa | |
| 21/33 | frj | |
| 22/34 | fad | |
| 23/35 | faf | |
| 25/37 | csdr | |
| 26/38 | cvsss | |
| 27/39 | drs | |
| 28/40 | parr | |
| 29/41 | gra | |

Line Setup configured for emulate or monitor mode.
### Table 80-1 (continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value (hex/decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern volatile unsigned long</td>
<td>m_label_dpc</td>
<td>0-3fff</td>
<td>CCITT format (2 bytes)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0-16383</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0-ffffff</td>
<td>ANSI format (3 bytes)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0-16777215</td>
<td>Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile unsigned long</td>
<td>m_label_opc</td>
<td>0-3fff</td>
<td>CCITT format (2 bytes)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0-16383</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0-ffffff</td>
<td>ANSI format (3 bytes)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0-16777215</td>
<td>Line Setup configured for emulate or monitor mode.</td>
</tr>
<tr>
<td>extern volatile const unsigned char</td>
<td>m_label_sis</td>
<td>0-ff</td>
<td>CCITT format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0-1ff/0-31</td>
<td>ANSI format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Line Setup configured for emulate or monitor mode.</td>
<td></td>
</tr>
<tr>
<td>extern volatile unsigned short</td>
<td>m_clc</td>
<td>0-fff/0-4095</td>
<td>TUP MSUs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0-fff/0-65535</td>
<td>ISDN MSUs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Line Setup configured for emulate or monitor mode.</td>
<td></td>
</tr>
<tr>
<td>extern unsigned char</td>
<td>i3_enhance</td>
<td>0</td>
<td>normal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>reverse</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>low</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>reverse low</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>blink</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
<td>reverse blink</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12/18</td>
<td>blink low</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Line Setup configured for emulate or monitor mode.</td>
<td></td>
</tr>
<tr>
<td>extern unsigned char</td>
<td>i3_suppress</td>
<td>0</td>
<td>off</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>on</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Line Setup configured for emulate or monitor mode.</td>
<td></td>
</tr>
</tbody>
</table>

### (B) Status Variables

Status variables are those in Table 80-1 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:

```c
CONDITIONS:
{
  dte_packet && (m_sio_si == 0)
}
```
Most status variables in Table 80-1 contain an _m_ prefix: _m_sio_ni_, _m_sio_priority_, _m_sio_si_, _m_code_type_, _m_label_dpc_, _m_labelopc_, _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 80-1. 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;
}
80.3 Routines

There are no routines associated exclusively with SS#7.
Appendix A: Operator Messages
## Appendix A1: Interactive Messages

The following messages are displayed on the second line of the screen, normally during execution of menu-screen functions.

<table>
<thead>
<tr>
<th>MESSAGE</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attempt to transfer source to itself</td>
<td>Source selections in the From and To fields on the Data Transfer screen are the same. Change one selection. To use one drive to perform data transfer involving two disks, change To selection to NEW.</td>
</tr>
<tr>
<td>Attempted to mount uninitialized disk</td>
<td>Check disk. It may require formatting.</td>
</tr>
<tr>
<td>Attempting to initialize link</td>
<td>Physical link being established prior to transfer.</td>
</tr>
<tr>
<td>BNDX message request failed</td>
<td>Message should not normally appear. If it recurs, contact Customer Service.</td>
</tr>
<tr>
<td>Backup complete, no errors</td>
<td>Duplication process is successful.</td>
</tr>
<tr>
<td>Bad object file format</td>
<td>Data is not recognized in format of object file. Try again to save the source file as an object file.</td>
</tr>
<tr>
<td>Can't load object file – Incompatible FEB installed</td>
<td>Current hardware is different from hardware of unit on which object file was saved. Save the source file as an object file on the unit on which it will be loaded and run.</td>
</tr>
<tr>
<td>Can't load object file – Incompatible MPM addressing</td>
<td>Current hardware is different from hardware of unit on which object file was saved. Save the source file as an object file on the unit on which it will be loaded and run.</td>
</tr>
<tr>
<td>Issue Description</td>
<td>Possible Solution</td>
</tr>
<tr>
<td>--------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Can’t load object file – Incompatible mux installed</td>
<td>Current hardware is different from hardware of unit on which object file was saved. Save the source file as an object file on the unit on which it will be loaded and run.</td>
</tr>
<tr>
<td>Can’t load object file – Insufficient MPMs</td>
<td>Current hardware is different from hardware of unit on which object file was saved. Save the source file as an object file on the unit on which it will be loaded and run.</td>
</tr>
<tr>
<td>Can’t load object file – No mux installed</td>
<td>Current hardware is different from hardware of unit on which object file was saved. Save the source file as an object file on the unit on which it will be loaded and run.</td>
</tr>
<tr>
<td>Can’t read disk</td>
<td>Make sure disk is correctly inserted. If message recurs, disk may be bad.</td>
</tr>
<tr>
<td>Cannot append to a wrapped DAT</td>
<td>Record Setup menu shows Stop at: ENDLESS LOOP. End of data acquisition tracks was reached, so wrapping occurred. Then, Data Transfer command attempted to append data from source to the end of DAT on destination disk. Select Start At: BEGIN on Data Transfer screen.</td>
</tr>
<tr>
<td>Cannot copy a file to itself</td>
<td>Be certain that name of destination file on File Maintenance screen differs from name of source file.</td>
</tr>
<tr>
<td>Cannot copy directory tree into itself</td>
<td>Attempt to copy a directory into one of its subdirectories. For example, a command to copy /usr into /usr/programs will fail.</td>
</tr>
<tr>
<td>Cannot delete a non-empty directory</td>
<td>File named for deletion is directory containing files. Before deleting directory, delete or move files.</td>
</tr>
<tr>
<td>Cannot open file</td>
<td>In attempt to load or save a file, the file could not be opened. Check the write–protect window. It should be closed to write to the disk.</td>
</tr>
<tr>
<td>Cannot open redirect file</td>
<td>Printer Setup menu shows that output will be redirected to a file. Check to make sure that the disk is properly inserted in the correct drive and is not write–protected.</td>
</tr>
<tr>
<td>Interactive Messages</td>
<td></td>
</tr>
<tr>
<td>----------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Cannot move file across disk boundaries</td>
<td></td>
</tr>
<tr>
<td>Attempt to rename a file from one disk to another.</td>
<td></td>
</tr>
<tr>
<td>Make sure only one drive is specified on File Maintenance</td>
<td></td>
</tr>
<tr>
<td>screen.</td>
<td></td>
</tr>
<tr>
<td>Cannot remove an open file</td>
<td></td>
</tr>
<tr>
<td>Attempt was made to copy a directory into itself.</td>
<td></td>
</tr>
<tr>
<td>Files being copied also need to be deleted, but cannot</td>
<td></td>
</tr>
<tr>
<td>be since they are open. Copy the directory to another</td>
<td></td>
</tr>
<tr>
<td>source. In general, close files before attempting to</td>
<td></td>
</tr>
<tr>
<td>delete them.</td>
<td></td>
</tr>
<tr>
<td>Cannot unmount disk, files open</td>
<td></td>
</tr>
<tr>
<td>Attempt was made to remove disk before operation</td>
<td></td>
</tr>
<tr>
<td>was completed.</td>
<td></td>
</tr>
<tr>
<td>Cannot write to redirect file</td>
<td></td>
</tr>
<tr>
<td>Printer Setup menu shows that output will be</td>
<td></td>
</tr>
<tr>
<td>redirected to a file. Error in trying to write to the</td>
<td></td>
</tr>
<tr>
<td>file. Check disk.</td>
<td></td>
</tr>
<tr>
<td>Change floppy disk 1</td>
<td></td>
</tr>
<tr>
<td>During multi-disk recording operation, disk in drive 1</td>
<td></td>
</tr>
<tr>
<td>has been filled. Remove old disk and insert new one.</td>
<td></td>
</tr>
<tr>
<td>Change floppy disk 2</td>
<td></td>
</tr>
<tr>
<td>During multi-disk recording operation, disk in drive 2</td>
<td></td>
</tr>
<tr>
<td>has been filled. Remove old disk and insert new one.</td>
<td></td>
</tr>
<tr>
<td>Character buffer not yet allocated</td>
<td></td>
</tr>
<tr>
<td>From field on the Data Transfer screen shows CHARBUFFER</td>
<td></td>
</tr>
<tr>
<td>but unless Run has been executed, there is no character</td>
<td></td>
</tr>
<tr>
<td>buffer. Press [Esc], and then try the transfer</td>
<td></td>
</tr>
<tr>
<td>operation again.</td>
<td></td>
</tr>
<tr>
<td>Compilation aborted</td>
<td></td>
</tr>
<tr>
<td>User has pressed ABORT softkey or [F1] to arrest the</td>
<td></td>
</tr>
<tr>
<td>Compile operation (from the File Maintenance menu).</td>
<td></td>
</tr>
<tr>
<td>Destination file may have been partially overwritten if</td>
<td></td>
</tr>
<tr>
<td>compile was to an existing file.</td>
<td></td>
</tr>
<tr>
<td>Compilation completed</td>
<td></td>
</tr>
<tr>
<td>The Compile command (from the File Maintenance menu)</td>
<td></td>
</tr>
<tr>
<td>has been executed.</td>
<td></td>
</tr>
<tr>
<td>Compilation failed – Errors detected</td>
<td></td>
</tr>
<tr>
<td>The Compile command (from the File Maintenance menu)</td>
<td></td>
</tr>
<tr>
<td>has been aborted because of errors. Go to the Protocol</td>
<td></td>
</tr>
<tr>
<td>Spreadsheet and press [Esc], [F1] to display the</td>
<td></td>
</tr>
<tr>
<td>first error message.</td>
<td></td>
</tr>
<tr>
<td>Compilation is in progress</td>
<td></td>
</tr>
<tr>
<td>The Compile command (from the File Maintenance menu)</td>
<td></td>
</tr>
<tr>
<td>is being executed to compile and save a file of</td>
<td></td>
</tr>
<tr>
<td>standard C code as object code.</td>
<td></td>
</tr>
</tbody>
</table>

Appendix A1
<table>
<thead>
<tr>
<th>Error Description</th>
<th>Possible Cause and Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy completed</td>
<td>Selected file(s) has (have) been copied successfully.</td>
</tr>
<tr>
<td>Copy is in progress</td>
<td>Selected file(s) is (are) being copied.</td>
</tr>
<tr>
<td>Could not load a layer personality package</td>
<td>Attempt to load a protocol package from the Layer Setup screen has failed. Make sure that correct disk is installed in drive indicated on menu. If attempt still fails, package may have been corrupted.</td>
</tr>
<tr>
<td>Current test invalidated</td>
<td>Changes have been made to the menu screens or Protocol Spreadsheet which invalidate a loaded object file.</td>
</tr>
<tr>
<td>Data transfer source and destination are the same</td>
<td>Source selections in the From and To fields on the Data Transfer screen are the same. Change one selection. To use one drive to perform data transfer involving two disks, change To selection to NEW.</td>
</tr>
<tr>
<td>Destination disk does not contain user file system</td>
<td>Check file contents on File Maintenance screen. If disk is intended for user files, you may need to allocate disk space to the filing system. Use Disk Utilities screen to check disk allocation.</td>
</tr>
<tr>
<td>Destination file is a directory</td>
<td>Copy or save operation not complete because file named to receive copy is a directory. Change destination filename and re-execute.</td>
</tr>
<tr>
<td>Destination file is write protected</td>
<td>Change destination filename or write-enable file. Then repeat save or copy operation.</td>
</tr>
<tr>
<td>Directory file expected</td>
<td>Check directory named in Change Directory command. Use only names labeled DIR in file listings.</td>
</tr>
<tr>
<td>Directory is empty</td>
<td>Attempted to copy the contents of an empty directory.</td>
</tr>
<tr>
<td>Directory is not empty</td>
<td>Directory cannot be deleted until all of the files it contains have been deleted.</td>
</tr>
<tr>
<td>Disk corrupted</td>
<td>Disk is worn out or damaged and should not be used for future operations.</td>
</tr>
<tr>
<td>Disk duplication aborted</td>
<td>Operator has aborted disk duplication. Data on destination disk may have been partially overwritten.</td>
</tr>
<tr>
<td>Error Message</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Disk duplication in progress</td>
<td>Disk is being duplicated. Do not remove disks from active drives.</td>
</tr>
<tr>
<td>Disk formatted</td>
<td>Formatting operation is complete.</td>
</tr>
<tr>
<td>Disk full</td>
<td>No space left on disk to perform operation. Use a new disk, or remove unneeded data from disk.</td>
</tr>
<tr>
<td>Disk not mounted</td>
<td>Re-insert disk and attempt operation again. Also try to power-up again. If message recurs, the disk may be bad.</td>
</tr>
<tr>
<td>Disk record error (controller error) -- Aborted</td>
<td>Disk may be write protected or recording too fast. Also may be an internal error or bad disk. Try again with a new disk. Contact Customer Service if it recurs.</td>
</tr>
<tr>
<td>Disk record error (timeout) -- Aborted</td>
<td>May be an internal error or bad disk. Try again with a new disk. Disk may be write protected or recording too fast.</td>
</tr>
<tr>
<td>Display screen command failed</td>
<td>Message should not normally appear. Contact Customer Service if it recurs.</td>
</tr>
<tr>
<td>Entering testprep</td>
<td>First status message entering Run mode. Test preparation mode precedes compilation of program.</td>
</tr>
<tr>
<td>Error during load of code file</td>
<td>Check code files. Message indicates code file is not found, or code file has been modified.</td>
</tr>
<tr>
<td>Error trying to print file</td>
<td>Try printing again. If attempt fails, disk file is probably corrupted.</td>
</tr>
<tr>
<td>Error trying to print screen</td>
<td>Try printing again. If problem recurs, contact Customer Service.</td>
</tr>
<tr>
<td>Errors occurred during load</td>
<td>Try loading again. If attempt fails, disk file is probably corrupted.</td>
</tr>
<tr>
<td>Errors occurred during save</td>
<td>Try saving again. If attempt fails, disk is probably corrupted.</td>
</tr>
<tr>
<td>Errors occurred during testprep</td>
<td>Attempt to perform Save command as a object file before the program had ever been compiled. An error was detected as compilation was attempted. Go to the Protocol Spreadsheet and search for errors. The Line Setup menu, for example, may show Mode: MONITOR and Source: DISK, but no disk is present in the selected drive.</td>
</tr>
<tr>
<td>Error Description</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Fatal Hardware Error</td>
<td>Invalid hardware setup. Contact Customer Service.</td>
</tr>
<tr>
<td>Fatal Software Error</td>
<td>Message should not normally appear. If it recurs, contact Customer Service.</td>
</tr>
<tr>
<td>FE buffer overflowed – Incoming data halted</td>
<td>Data is coming in faster than it can be received.</td>
</tr>
<tr>
<td>File access error</td>
<td>File named cannot be accessed. Check disk.</td>
</tr>
<tr>
<td>File copy aborted</td>
<td>User has pressed ABORT softkey to arrest copy operation. Destination file may have been partially overwritten if copy was to an existing file.</td>
</tr>
<tr>
<td>File is a directory</td>
<td>Operation, View for example, could not be performed on a directory. Select or enter the name of a file. See listings on File Maintenance screen for entry NOT labeled DIR.</td>
</tr>
<tr>
<td>File is write-protected</td>
<td>File named cannot be deleted or saved. Check name. To perform operation on named file, write-enable it from the File Maintenance screen.</td>
</tr>
<tr>
<td>File loaded</td>
<td>File has been loaded successfully as a Program, Setup, or Object file.</td>
</tr>
<tr>
<td>File name not found</td>
<td>Filename (or directory) as entered does not appear in listings. Check spelling of entry. Make sure you are operating in correct directory.</td>
</tr>
<tr>
<td>File name already exists</td>
<td>Attempt to use the Make Directory command, naming a directory that already exists.</td>
</tr>
<tr>
<td>File saved</td>
<td>File has been saved successfully as a Program, Setup, or Object file.</td>
</tr>
<tr>
<td>File size can’t be increased, index block full</td>
<td>File is larger than the file system can handle.</td>
</tr>
<tr>
<td>Formatting disk – max floppy disk DAT allowed = 1422K bytes</td>
<td>Too much space specified for data acquisition tracks. Maximum space has been allocated.</td>
</tr>
<tr>
<td>Formatting disk – max hard disk DAT allowed = 20774K bytes</td>
<td>Too much space specified for data acquisition tracks. Maximum space has been allocated.</td>
</tr>
<tr>
<td>Interactive Messages</td>
<td></td>
</tr>
<tr>
<td>----------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Formatting disk</td>
<td></td>
</tr>
<tr>
<td>Formatting in progress. Do not remove disk from active</td>
<td></td>
</tr>
<tr>
<td>drive.</td>
<td></td>
</tr>
<tr>
<td>Formatting will destroy data</td>
<td></td>
</tr>
<tr>
<td>Message appears when a formatted disk has been inserted</td>
<td></td>
</tr>
<tr>
<td>for reformatting. Press ABORT to avoid overwriting data.</td>
<td></td>
</tr>
<tr>
<td>Depress F1 key to continue.</td>
<td></td>
</tr>
<tr>
<td>Function failed -- Check media</td>
<td></td>
</tr>
<tr>
<td>Attempt operation again. If operation still fails, disk</td>
<td></td>
</tr>
<tr>
<td>may be bad. Try new disk.</td>
<td></td>
</tr>
<tr>
<td>Function(s) not yet implemented</td>
<td></td>
</tr>
<tr>
<td>Operation attempted is not available with the software</td>
<td></td>
</tr>
<tr>
<td>version installed.</td>
<td></td>
</tr>
<tr>
<td>II buffer services error</td>
<td></td>
</tr>
<tr>
<td>Error in using OSI variables or routines. May occur, for</td>
<td></td>
</tr>
<tr>
<td>example, if operation is attempted on buffer which no</td>
<td></td>
</tr>
<tr>
<td>longer exists. Set maintain bits at each layer that needs</td>
<td></td>
</tr>
<tr>
<td>to reserve the buffer for subsequent operations.</td>
<td></td>
</tr>
<tr>
<td>Illegal device name</td>
<td></td>
</tr>
<tr>
<td>Message should not normally appear. If it recurs,</td>
<td></td>
</tr>
<tr>
<td>contact Customer Service.</td>
<td></td>
</tr>
<tr>
<td>Illegal expansion unit, not 1-255</td>
<td></td>
</tr>
<tr>
<td>Message should not normally appear. If it recurs,</td>
<td></td>
</tr>
<tr>
<td>contact Customer Service.</td>
<td></td>
</tr>
<tr>
<td>Illegal file number passed to open</td>
<td></td>
</tr>
<tr>
<td>Message should not normally appear. If it recurs,</td>
<td></td>
</tr>
<tr>
<td>contact Customer Service.</td>
<td></td>
</tr>
<tr>
<td>Illegal major device number given</td>
<td></td>
</tr>
<tr>
<td>Message should not normally appear. If it recurs,</td>
<td></td>
</tr>
<tr>
<td>contact Customer Service.</td>
<td></td>
</tr>
<tr>
<td>Illegal parameter to volume init. function</td>
<td></td>
</tr>
<tr>
<td>Message should not normally appear. If it recurs,</td>
<td></td>
</tr>
<tr>
<td>contact Customer Service.</td>
<td></td>
</tr>
<tr>
<td>Illegal pathname</td>
<td></td>
</tr>
<tr>
<td>Pathname provided is incomplete or invalid. Check entry.</td>
<td></td>
</tr>
<tr>
<td>Illegal position parameter, not 0-2</td>
<td></td>
</tr>
<tr>
<td>Message should not normally appear. If it recurs,</td>
<td></td>
</tr>
<tr>
<td>contact Customer Service.</td>
<td></td>
</tr>
<tr>
<td>Illegal synchronization mode, not 0-2</td>
<td></td>
</tr>
<tr>
<td>Message should not normally appear. If it recurs,</td>
<td></td>
</tr>
<tr>
<td>contact Customer Service.</td>
<td></td>
</tr>
<tr>
<td>Indirect stat update msg received</td>
<td></td>
</tr>
<tr>
<td>Message should not normally appear. If it recurs,</td>
<td></td>
</tr>
<tr>
<td>contact Customer Service.</td>
<td></td>
</tr>
</tbody>
</table>
Insert next disk -- Depress F1 key to continue

More than one disk required to perform duplication.

Insert destination disk, depress F1 key to continue

Operation involving more than one disk being performed using one drive.

Insert source disk, depress F1 key to continue

Operation involving more than one disk being performed using one drive.

Internal disk sub-system error

Message should not normally appear. If it recurs, contact Customer Service.

Inter-processor communication overrun

Communication from MPM to CPM occurring too fast for CPM. Available buffer space exceeded.

Invalid contents in field

Entry made in menu field is illegal.

Invalid DAT block version number

Each block in DAT has a version number. If it is wrong, the disk may be corrupted.

Invalid DAT version number

Header in DAT has a version number. If it is wrong, the disk may be corrupted.

Invalid data type

During attempted playback, INTERVIEW did not recognize type of data. Be certain recorded data rather than program data is being accessed.

Invalid destination

Destination file in a Copy command is a relative pathname on a drive which is not the current drive.

Invalid disk sub-system function number

Message should not normally appear. If it recurs, contact Customer Service.

Invalid file identifier, no such open file

Message should not normally appear. If it recurs, contact Customer Service.

Invalid file identifier, out of range

Message should not normally appear. If it recurs, contact Customer Service.

Invalid filetype

Command cannot be used on file type indicated. A Load command, for example, is not valid for SYS files.
**Appendix A1  Interactive Messages**

<table>
<thead>
<tr>
<th>Message Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalid filetype for viewing</td>
<td>View command cannot be used for data in file indicated. Files with type SYS, for example, cannot be viewed.</td>
</tr>
<tr>
<td>Invalid layer number</td>
<td>User has entered layer number out of valid range.</td>
</tr>
<tr>
<td>Invalid object code version</td>
<td>Object file was saved under a different version than current software. Save the source file as an object file using the same software with which is will be loaded and run.</td>
</tr>
<tr>
<td>Invalid section name</td>
<td>Message should not normally appear. If it recurs, contact Customer Service.</td>
</tr>
<tr>
<td>Invalid stat update msg received</td>
<td>Message should not normally appear. If it recurs, contact Customer Service.</td>
</tr>
<tr>
<td>Load aborted</td>
<td>Operator has aborted Load operation. Program already residing in INTERVIEW may have been altered.</td>
</tr>
<tr>
<td>Load is in progress</td>
<td>Selected Program, Setup, or Object file is being loaded.</td>
</tr>
<tr>
<td>Loaded package and configuration screen don’t match</td>
<td>Message should not normally appear. If it recurs, contact Customer Service.</td>
</tr>
<tr>
<td>Marked entry not copied</td>
<td>Too many items have been marked for single operation. Not all files marked have been copied. Check listings on the File Maintenance screen. Files still marked are not yet copied. Repeat copy operation on remaining files.</td>
</tr>
<tr>
<td>Marked entry not deleted</td>
<td>Too many items have been marked for single operation. Not all files marked have been deleted. Check listings on the File Maintenance screen. Files still marked are not yet deleted. Repeat delete operation on remaining files.</td>
</tr>
<tr>
<td>Marked entry not printed</td>
<td>Too many items have been marked for single operation. Not all files marked have been printed. Check listings on the File Maintenance screen. Files still marked are not yet printed. Repeat print operation on remaining files.</td>
</tr>
<tr>
<td>Error Message</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Maximum number of entries exceeded</td>
<td>Error on Tabular Statistics screen. Maximum number of entries is 100.</td>
</tr>
<tr>
<td>Maximum disks already mounted</td>
<td>Message should not normally appear. If it recurs, contact Customer Service.</td>
</tr>
<tr>
<td>Memory has not been unlocked yet</td>
<td>Message should not normally appear. If it recurs, contact Customer Service.</td>
</tr>
<tr>
<td>Message exchange full</td>
<td>Message should not normally appear. If it recurs, contact Customer Service.</td>
</tr>
<tr>
<td>Message ID too big</td>
<td>Message should not normally appear. If it recurs, contact Customer Service.</td>
</tr>
<tr>
<td>Move would destroy directory tree structure</td>
<td>Attempt to rename a directory the same as one of its subdirectories.</td>
</tr>
<tr>
<td>MPM -- Bus error</td>
<td>May indicate a hardware problem, but check program for logic errors relating to storage allocation. May have attempted to access something that doesn’t exist.</td>
</tr>
<tr>
<td>MPM -- Divide fault</td>
<td>Program may include an attempt to divide by zero. Check for other logic errors in program.</td>
</tr>
<tr>
<td>MPM -- Processor fault</td>
<td>May indicate a hardware problem, but check program for logic errors relating to storage allocation. May have attempted to access something that doesn’t exist.</td>
</tr>
<tr>
<td>MPM -- Memory fault</td>
<td>Logic error in program, typically relating to storage allocation. May have attempted to access something that doesn’t exist, accessing an array outside of its range, for example. May also indicate type mismatches.</td>
</tr>
<tr>
<td>MPM -- Stack fault</td>
<td>Logic error in program, typically relating to storage allocation. May have attempted to access something that doesn’t exist, accessing an array outside of its range, for example. May also indicate type mismatches.</td>
</tr>
<tr>
<td>NEWDISK illegal with source of hard disk</td>
<td>When To field for a Copy command is NEW, it means that the same drive will be used to perform a copy involving two disks. Change From field to FD1 or FD2.</td>
</tr>
<tr>
<td>Message</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NEWDISK illegal with source of RAM or hard disk</td>
<td>When the field for a Data Transfer command is <strong>NEW</strong>, it means that the same drive will be used to perform a transfer involving two disks. Change <strong>From</strong> field to <strong>FD1</strong> or <strong>FD2</strong>.</td>
</tr>
<tr>
<td>No DAT RAM currently allocated</td>
<td>Attempt to transfer data from RAM without it having been recorded previously to RAM.</td>
</tr>
<tr>
<td>No default directory set</td>
<td>Message should not normally appear. If it recurs, contact Customer Service.</td>
</tr>
<tr>
<td>No file name specified</td>
<td>Enter or indicate file on which operation attempted is to be performed.</td>
</tr>
<tr>
<td>No package loaded for this layer</td>
<td>Selection has been made on the Layer Setup screen, but no protocol package has been loaded. Return to Layer Setup, check selection, and press <strong>SET</strong>.</td>
</tr>
<tr>
<td>No packages loaded</td>
<td>Selections have been made on the Layer Setup screen, but no protocol packages have been loaded. Return to Layer Setup, check selections, and press <strong>SET</strong>.</td>
</tr>
<tr>
<td>No RAM recording memory available</td>
<td>Program is too large to be recorded into available RAM.</td>
</tr>
<tr>
<td>No start of section indicator</td>
<td>Operation on file cannot be performed because (1) file is not a program or setup, (2) format of the file is invalid, or (3) file has been corrupted.</td>
</tr>
<tr>
<td>No message entered in message buffer</td>
<td>Check BERT screen. Configured menu indicated a message would be sent, but none was entered.</td>
</tr>
<tr>
<td>Obsolete object program – Source must be recompiled</td>
<td>Object file is incompatible with current software.</td>
</tr>
<tr>
<td>Out of memory</td>
<td>Insufficient memory to perform operation. (Program is too large to run.)</td>
</tr>
<tr>
<td>Operation not allowed on specified file</td>
<td>Selected command cannot be used on file indicated.</td>
</tr>
<tr>
<td>Parent directory of file is write–protected</td>
<td>Parent directory must be write–enabled before you can modify or delete this file.</td>
</tr>
<tr>
<td>Parent directory of target file does not exist</td>
<td>Check spelling of directory.</td>
</tr>
<tr>
<td>Interference</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Play underrun</td>
<td>Data could not be output at speed requested.</td>
</tr>
<tr>
<td>Premature end of section</td>
<td>Operation on file cannot be performed because (1) file is not a program or setup, (2) format of the file is invalid, or (3) file has been corrupted.</td>
</tr>
<tr>
<td>Previous lock user has died</td>
<td>Message should not normally appear. If it recurs, contact Customer Service.</td>
</tr>
<tr>
<td>Print queue is full</td>
<td>Maximum number of print jobs has been requested. Wait for some requests to be completed. Then repeat print operation.</td>
</tr>
<tr>
<td>Printing is done</td>
<td>Print jobs requested are completed.</td>
</tr>
<tr>
<td>Record overrun</td>
<td>Data being received too rapidly for capture to RAM.</td>
</tr>
<tr>
<td>Remove screen command failed</td>
<td>Message should not normally appear. If it recurs, contact Customer Service.</td>
</tr>
<tr>
<td>Replace screen command failed</td>
<td>Message should not normally appear. If it recurs, contact Customer Service.</td>
</tr>
<tr>
<td>Resetting compiled test</td>
<td>Program being run again without recompiling. Menus can be viewed and selected changes can be made to menus without forcing a recompile.</td>
</tr>
<tr>
<td>Routine calling save_prog_setup is unknown</td>
<td>Message should not normally appear. If it recurs, contact Customer Service.</td>
</tr>
<tr>
<td>Save aborted</td>
<td>Operator has aborted save operation. If save was to an existing file, the file may have been partially overwritten.</td>
</tr>
<tr>
<td>Save is in progress</td>
<td>Selected Program, Setup, or Object file is being saved.</td>
</tr>
<tr>
<td>Seek attempted before beginning of DAT</td>
<td>Data Transfer screen shows transfer from disk with Start At Block entry that precedes the block number at which data actually begins. DAT may begin at block 20, for example. If you enter Start At Block: 2, this error message will be displayed. To guarantee that data transfer starts from the beginning of DAT, enter zero in the Start At Block field. Zero is a special entry. It references the beginning of DAT, regardless of what the actual block number may be. Any other entry is interpreted as a literal block number.</td>
</tr>
</tbody>
</table>
### Appendix A1 Interactive Messages

<table>
<thead>
<tr>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seek attempted past end of DAT</td>
<td>Data Transfer screen shows transfer from disk with Start At Block entry that exceeds the block number at which data actually ends. DAT may end at block 100, for example. If you enter Start At Block: 101, this error message will be displayed. To guarantee that data transfer starts from the beginning of DAT, enter zero in the Start At Block field. Zero is a special entry. It references the beginning of DAT, regardless of what the actual block number may be. Any other entry is interpreted as a literal block number.</td>
</tr>
<tr>
<td>Seek attempted past end of file</td>
<td>Message should not normally appear. If it recurs, contact Customer Service.</td>
</tr>
<tr>
<td>Source &amp; destination can’t be the same disk</td>
<td>Error in entries made for disk duplication. Check disks selected.</td>
</tr>
<tr>
<td>Source disk does not contain user file system</td>
<td>Check disk contents on the Disk Utility screen. If disk is intended for user files, you may need to allocate space to the filing system.</td>
</tr>
<tr>
<td>Source file not found</td>
<td>In Interview 10/15/20 file transfer, the source file which was specified does not exist.</td>
</tr>
<tr>
<td>Stopped at end of DAT</td>
<td>During playback, stopped at end of recorded data. During recording, stopped at end of data acquisition tracks.</td>
</tr>
<tr>
<td>TEST PREPARATION Phase 1</td>
<td>Program is being compiled.</td>
</tr>
<tr>
<td>TEST PREPARATION Phase 2</td>
<td>Program is being compiled.</td>
</tr>
<tr>
<td>TEST PREPARATION Phase 3</td>
<td>Program is being compiled.</td>
</tr>
<tr>
<td>TEST PREPARATION Phase 4</td>
<td>Program is being compiled.</td>
</tr>
<tr>
<td>TEST PREPARATION Phase 5</td>
<td>Program is being compiled.</td>
</tr>
<tr>
<td>TEST PREPARATION Phase 6</td>
<td>Program is being compiled.</td>
</tr>
<tr>
<td>TEST PREPARATION Phase 7</td>
<td>Program is being compiled.</td>
</tr>
<tr>
<td>There are no free locks left</td>
<td>Message should not normally appear. If it recurs, contact Customer Service.</td>
</tr>
</tbody>
</table>

A1–13
<table>
<thead>
<tr>
<th>Error Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Token in load file is invalid</td>
<td>Operation on file cannot be performed because (1) file is not a program, setup, or object, (2) format of the file is invalid, or (3) file has been corrupted.</td>
</tr>
<tr>
<td>Token is incomplete</td>
<td>Operation on file cannot be performed because (1) file is not a program, setup, or object, (2) format of the file is invalid, or (3) file has been corrupted.</td>
</tr>
<tr>
<td>Too many files in directory</td>
<td>Maximum number of files that can be displayed is 200. If the current directory contains more than 200 files, this message is displayed.</td>
</tr>
<tr>
<td>Too many files on disk, FLIST full</td>
<td>Directory area on disk is full, although there may be more space available for recording. Delete unnecessary filenames to gain access to free space remaining on disk.</td>
</tr>
<tr>
<td>Too many open files for process</td>
<td>Each process is limited to a maximum of ten files open at one time.</td>
</tr>
<tr>
<td>Too many open files for system</td>
<td>There is a system-wide limit of 20 files open at one time.</td>
</tr>
<tr>
<td>Too many processes using disk sub-system</td>
<td>There can be no more than twelve processes using file I/O simultaneously.</td>
</tr>
<tr>
<td>Too many source files selected</td>
<td>Operator has used <code>E3</code> to select multiple source files for executing the Compile command (from the File Maintenance menu). Select only one source file to compile and save as a linkable-object (LOBJ) file.</td>
</tr>
<tr>
<td>Transfer aborted</td>
<td>Data transfer operation has been aborted. Partial transfer of data may have occurred, overwriting storage medium at destination.</td>
</tr>
<tr>
<td>Transfer complete</td>
<td>Data transfer has been completed successfully.</td>
</tr>
<tr>
<td>Transfer in progress</td>
<td>Data transfer being executed.</td>
</tr>
<tr>
<td>Transmit overrun</td>
<td>Attempt to transmit data faster than unit can transmit.</td>
</tr>
<tr>
<td>Unable to access disk</td>
<td>No file named in a Data Transfer to a file, disk not in drive, disk is write-protected, or disk is unformatted.</td>
</tr>
<tr>
<td>Message</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>Unable to access disk in selected drive.</td>
<td>During multi-disk recording, the next drive in sequence does not contain a disk, contains a write-protected disk, or contains an unformatted disk. During file maintenance operations or disk duplication, source disk is not present in selected drive, is write-protected, or is unformatted.</td>
</tr>
<tr>
<td>Unable to access m_list</td>
<td>Disk error. Check disk and try operation again. If message recurs, disk may need reformatting.</td>
</tr>
<tr>
<td>Unable to execute XEQ key</td>
<td>Attempt to execute a File Maintenance command before the current directory is displayed.</td>
</tr>
<tr>
<td>Unable to open DAT</td>
<td>There are no data acquisition tracks on the disk being accessed.</td>
</tr>
<tr>
<td>Unable to open file</td>
<td>Disk error. Check disk and try operation again. If message recurs, disk may need reformatting.</td>
</tr>
<tr>
<td>Unable to open next disk</td>
<td>Recording using more than one disk. Next disk may not be installed.</td>
</tr>
<tr>
<td>Unable to read DAT info block</td>
<td>DAT block not where indicated. Disk may be corrupted.</td>
</tr>
<tr>
<td>Unable to read file</td>
<td>Check disk.</td>
</tr>
<tr>
<td>Unable to read from DAT</td>
<td>Check disk.</td>
</tr>
<tr>
<td>Unable to write to DAT</td>
<td>1) May have attempted to transfer more data to destination disk than the space allocated for data during disk formatting. A Summary of the disk may show no free space remaining for data acquisition. 2) Disk may be write-protected.</td>
</tr>
<tr>
<td>Unhandled CPM interrupt</td>
<td>Message should not normally appear. If it recurs, contact Customer Service.</td>
</tr>
<tr>
<td>Unhandled MPM interrupt</td>
<td>Press to recover from this error. Message should not normally appear, however. If it recurs, contact Customer Service.</td>
</tr>
<tr>
<td>Unknown DAT type</td>
<td>Data acquisition tracks may have been recorded on a unit with more recent software than is installed in the unit being used to playback the data.</td>
</tr>
<tr>
<td>Issue</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Unknown filetype</td>
<td>Check entry in file listing. Check disk. Try operation again. If message recurs, delete and recreate file (if possible). Some file types may not be known for certain operations. A Print command on a SYS file, for example, generates this error message.</td>
</tr>
<tr>
<td>Unprintable screen</td>
<td>Screen requested cannot be printed. Refer to section on printing for printable screens.</td>
</tr>
<tr>
<td>Unrecoverable error during format</td>
<td>Format operation failed. Make sure disk is inserted properly in selected drive. Check disk type. One Mbyte disks are not supported. If disk type is correct, re-attempt formatting. If second attempt fails, disk may be bad.</td>
</tr>
</tbody>
</table>
Appendix A2: Error Messages Issued by C Translator

If a spreadsheet program contains any of the following errors, the compilation will be interrupted and you will be returned to the Protocol Spreadsheet. A diagnostic message about the first error will be displayed at the top (second line) of the screen. To search for additional error messages, press F5.

<table>
<thead>
<tr>
<th>MESSAGE</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR &quot;C&quot; conditions text too long</td>
<td>A C region in a CONDITIONS block is more than 300 characters.</td>
</tr>
<tr>
<td>Bad format in object file</td>
<td>Unsuccessful attempt to access linkable–object file via the OBJECT block identifier. Use the Compile command on the File Maintenance screen to recreate the LOBJ file, and try again.</td>
</tr>
<tr>
<td>BIB value out of range</td>
<td>An SS#7 condition at Layer 2 specifies a BIB= value that is not zero or one.</td>
</tr>
<tr>
<td>Bit mask exceeds maximum length</td>
<td>A FLAG condition or FLAG name SET action includes a bit mask that exceeds 16 bits. In other uses, bit mask is typically eight bits.</td>
</tr>
<tr>
<td>Cannot find object file</td>
<td>Attempt to access a linkable–object file via the OBJECT block identifier. Either the file does not exist, or it resides in a directory not included in the search path.</td>
</tr>
<tr>
<td>Constant reference stack overflow</td>
<td>1) Attempt to nest constants more than eight deep, or 2) constants are defined circularly.</td>
</tr>
<tr>
<td>Constant value too long</td>
<td>Context in which constant is used determines what value is too long. A constant in a Layer 1 receive string condition, for example, when expanded, cannot exceed 32 characters.</td>
</tr>
<tr>
<td>Duplicate state name</td>
<td>Attempt to use state name twice in the same test.</td>
</tr>
<tr>
<td>Error Description</td>
<td>Error Description</td>
</tr>
<tr>
<td>-------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Duplicate test name</td>
<td>Attempt to use test name twice in the same task (layer).</td>
</tr>
<tr>
<td>Edit buffer full</td>
<td>Spreadsheet program is too large. Use include files.</td>
</tr>
<tr>
<td>Empty conditions section</td>
<td>There is no entry for a CONDITIONS block.</td>
</tr>
<tr>
<td>FIB value out of range</td>
<td>An SS#7 condition at Layer 2 specifies an FIB= value that is not zero or one.</td>
</tr>
<tr>
<td>Identifier exceeds maximum length</td>
<td>Message should not normally appear. It means, however, that an identifier is too long for the context in which it is being used.</td>
</tr>
<tr>
<td>Idle string must contain exactly one character</td>
<td>Layer 1 IDLE action includes a string with more than one character.</td>
</tr>
<tr>
<td>Illegal bit value</td>
<td>Bit has been assigned a value other than zero or one. In X.25 protocol, for example, the user supplies a value for the Q, D, or M bit at Layer 3.</td>
</tr>
<tr>
<td>Illegal cause value</td>
<td>An X.25 condition at Layer 3 specifies a numeric value for the CAUSE= selection which is outside the valid range. Select a value between hexadecimal 0 and FF.</td>
</tr>
<tr>
<td>Illegal CIC type for ISDN</td>
<td>An SS#7 condition at Layer 3 specifies a CIC= value for an ISDN header which is outside the valid range. Select a value between hexadecimal 0 and FFFF.</td>
</tr>
<tr>
<td>Illegal CIC type for TUP</td>
<td>An SS#7 condition at Layer 3 specifies a CIC= value for a TUP header which is outside the valid range. Select a value between hexadecimal 0 and FFF.</td>
</tr>
<tr>
<td>Illegal control byte</td>
<td>An X.25, LAPD, SDLC, or SNA condition at Layer 2 (as in the example which follows) specifies a value for the frame type which is outside the valid range: CONDITIONS: DTE OTHER 1FF. Select a value between hexadecimal 0 and FF.</td>
</tr>
<tr>
<td>Illegal diag value</td>
<td>An X.25 condition at Layer 3 specifies a value for the DIAG= selection which is outside the valid range. Select a value between hexadecimal 0 and FF.</td>
</tr>
<tr>
<td>Error Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Illegal DPC type</td>
<td>An SS#7 condition at Layer 3 specifies a value for the DPC= selection which is outside the valid range. For CCITT format, select a value between hexadecimal 0 and 3FFF. For ANSI format, select a value between hexadecimal 0 and FFFFFFF.</td>
</tr>
<tr>
<td>Illegal frame address</td>
<td>An X.25, SDLC, or SNA condition at Layer 2 (as in the example which follows) specifies a value for the frame address which is outside the valid range: CONDITIONS: DTE INFO ADDR= 1FF. Select a value between hexadecimal 0 and FF.</td>
</tr>
<tr>
<td>Illegal LCN value</td>
<td>An X.25 condition at Layer 3 specifies a value for the LCN= selection which is outside the valid range. Select a value between hexadecimal 0 and FFF.</td>
</tr>
<tr>
<td>Illegal OPC type</td>
<td>An SS#7 condition at Layer 3 specifies a value for the OPC= selection which is outside the valid range. For CCITT format, select a value between hexadecimal 0 and 3FFF. For ANSI format, select a value between hexadecimal 0 and FFFFFFF.</td>
</tr>
<tr>
<td>Illegal path number</td>
<td>An X.25 condition or action at Layer 3 specifies a value for the PATH= selection which is outside the valid range. Select a value between zero and eight.</td>
</tr>
<tr>
<td>Illegal P/F bit</td>
<td>An X.25, SDLC, or SNA condition or SEND action at Layer 2 specifies a value for the P/F= selection that is not zero or one.</td>
</tr>
<tr>
<td>Illegal PR value</td>
<td>An X.25 SEND action at Layer 3 specifies a value for the PR= selection which is outside the valid range. Select a value between zero and 127.</td>
</tr>
<tr>
<td>Illegal PS value</td>
<td>An X.25 SEND action at Layer 3 specifies a value for the PS= selection which is outside the valid range. Select a value between zero and 127.</td>
</tr>
<tr>
<td>Illegal receive count</td>
<td>In an X.25, SDLC, LAPD, or SNA Layer 2 SEND action, the value specified for N(R) is out of range. Select a value between zero and 127.</td>
</tr>
<tr>
<td>Illegal SAPI value</td>
<td>A LAPD condition or SEND action at Layer 2 specifies a value for the SAPI= selection which is outside the valid range. Select a value between hexadecimal 0 and 3F.</td>
</tr>
</tbody>
</table>
## Interview 7000 Series Advanced Programming: ATLC-107-951-108

<table>
<thead>
<tr>
<th>Error Description</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Illegal send count</td>
<td>In an X.25, SDLC, LAPD, or SNA Layer 2 SEND action, the value specified for N(S) is out of range. Select a value between zero and 127.</td>
</tr>
<tr>
<td>Illegal SI type</td>
<td>In an SS#7 Layer 3 OTHER condition, the value specified for Service Information is out of range. Select a value between hexadecimal 0 and FF.</td>
</tr>
<tr>
<td>Illegal SLS type</td>
<td>An SS#7 condition at Layer 3 specifies a value for the SLS= selection which is outside the valid range. For CCITT format, select a value between hexadecimal 0 and F. For ANSI format, select a value between hexadecimal 0 and 1F.</td>
</tr>
<tr>
<td>Illegal TEI value</td>
<td>A LAPD condition or SEND action at Layer 2 specifies a value for the TEI= selection which is outside the valid range. Select a value between hexadecimal 0 and 7F.</td>
</tr>
<tr>
<td>Incomplete EIA action</td>
<td>Required number of softkey selections have not been made for a Layer 1 EIA action.</td>
</tr>
<tr>
<td>Invalid constant reference</td>
<td>Valid characters for a constant name include 0–9, upper- and lower-case letters, and underscores. Name cannot begin with a number. The message also may indicate that a special “constant” of the form $(name{45})$ has been used, but the string using it is missing the enclosing quotation marks.</td>
</tr>
<tr>
<td>Invalid counter arguments</td>
<td>In a SEND action, the string to be sent contains a reference of the form $(counter[n])$. The value of $n$ is out of range. Select a value between zero and three.</td>
</tr>
<tr>
<td>Invalid counter value</td>
<td>COUNTER condition or a COUNTER name SET action specifies a value for the counter which is outside the valid range. Select a value between zero and 4,294,967,295.</td>
</tr>
<tr>
<td>Invalid day of month</td>
<td>TIME condition specifies a day of the month which is outside the valid range. Select a value between one and thirty-one.</td>
</tr>
<tr>
<td>Invalid flag arguments</td>
<td>In a SEND action, the string to be sent contains a reference of the form $(flag[n])$. The value of $n$ is out of range. Select either zero or one.</td>
</tr>
<tr>
<td>Error Message</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Invalid time value</td>
<td>TIME condition specifies a time which is outside the valid range for the 24-hour format.</td>
</tr>
<tr>
<td>Invalid time of day</td>
<td>TIME condition specifies a time which is outside the valid range for the 24-hour format.</td>
</tr>
<tr>
<td>Invalid timeout value</td>
<td>TIMEOUT name RESTART action specifies a value which is outside the valid range. Select a value between 0.001 and 65.535. Do not begin entry with decimal point.</td>
</tr>
<tr>
<td>Invalid trigger (lacks transitional)</td>
<td>Condition does not contain an event. At Layer 3 in X.25 for example, the status-only condition MORE_TO_RESEND is not combined with an event. Add an event such as PACKET_SENT to the condition.</td>
</tr>
<tr>
<td>Invalid trigger (multiple transitional–only)</td>
<td>Condition contains more than one event. Since no two events can come true at the same time, move one of the events to a separate CONDITIONS block.</td>
</tr>
<tr>
<td>Invalid character in constant name</td>
<td>Valid characters include 0–9, upper- and lower-case letters, and underscores. Name cannot begin with a number.</td>
</tr>
<tr>
<td>No closing »</td>
<td>Double parentheses delimit constants.</td>
</tr>
<tr>
<td>No closing ]</td>
<td>In a SEND action, the string contains a reference to a flag or counter which includes additional information inside brackets. The closing bracket is missing.</td>
</tr>
<tr>
<td>No more errors</td>
<td>There are no more errors to be displayed via GO_ERR. The next time you press @, the last error message will be displayed.</td>
</tr>
<tr>
<td>Not an object file</td>
<td>Attempt to access a file via the OBJECT block identifier that has a type other than LOBJ (linkable–object). Use the Compile command on the File Maintenance screen to create a linkable–object file from a source file containing standard C code.</td>
</tr>
<tr>
<td>Obsolete object file version</td>
<td>Attempt to access a linkable–object file via the OBJECT block identifier. Use the Compile command on the File Maintenance screen to recompile the source file, and try again.</td>
</tr>
</tbody>
</table>
### INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-10B

<table>
<thead>
<tr>
<th>Issue</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Obsolete package loaded</td>
<td>A layer personality package is loaded which came from an older version of the software. The package is attempting to use facilities which are not provided in the current version of the software.</td>
</tr>
<tr>
<td>Out of buffer space</td>
<td>The translator has run out of memory.</td>
</tr>
<tr>
<td>Out of memory</td>
<td>The translator has run out of memory.</td>
</tr>
<tr>
<td>Premature end of file</td>
<td>1) Required softkey selections for a condition have not been made. To send a string at Layer 1, for example, you must make a BCC selection, or 2) string does not contain closing quotation mark.</td>
</tr>
<tr>
<td>Receive string cannot be longer than 32 characters</td>
<td>RECEIVE STRING condition at Layer 1 contains more than 32 characters. Note that constants are expanded before they are counted.</td>
</tr>
<tr>
<td>Reference to undefined constant</td>
<td>Attempt to use a constant that has not been defined.</td>
</tr>
<tr>
<td>Reference to undefined state</td>
<td>State name referenced in NEXT_STATE does not exist.</td>
</tr>
<tr>
<td>Syntax error</td>
<td>Program may contain punctuation errors or incomplete softkey selections. This message often accompanies other errors messages.</td>
</tr>
<tr>
<td>There is no next state</td>
<td>Included NEXT_STATE: NEXT, but no state follows.</td>
</tr>
<tr>
<td>Unclosed AR&quot;C&quot; region</td>
<td>1) Unequal number of opening and closing curly braces, 2) unclosed quotation marks, or 3) unclosed parentheses.</td>
</tr>
<tr>
<td>Unclosed quoted string</td>
<td>Insert closing quotation marks at end of string.</td>
</tr>
<tr>
<td>Undefined name</td>
<td>The string in a SEND action contains a constant reference which does not refer to any defined flag, counter, constant, or special inter-layer data constant.</td>
</tr>
<tr>
<td>Unexpected character in constant name</td>
<td>Valid characters include 0–9, upper- and lower-case letters, and underscores. Name cannot begin with a number.</td>
</tr>
<tr>
<td>Unknown object file version</td>
<td>Should not normally appear. May be attempting to use version of a linkable-object file which is incompatible with older versions of software.</td>
</tr>
<tr>
<td>WAIT_EOF with ENTER_STATE</td>
<td>A trigger condition has both ENTER_STATE and a line condition including the WAIT_EOF option. This error is a special case of the &quot;Invalid trigger (multiple transition-only)&quot; error.</td>
</tr>
</tbody>
</table>
Appendix A3: Error Messages Issued by C Compiler

Most of the following messages report errors that interrupt the compilation of a spreadsheet program and return you to the Protocol Spreadsheet. A diagnostic message about the first error will be displayed at the top (second line) of the screen. Some messages also serve as warnings. Warnings do not cause compilation to be aborted, but they are displayed on the Protocol Spreadsheet with error messages. Suppress warning messages using the following #pragma:

```
#pragma nowarn
```

<table>
<thead>
<tr>
<th>Number</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>Only integral values may be added to pointers.</td>
</tr>
<tr>
<td>002</td>
<td>Within constant expressions, the operand of the unary <code>&amp;</code> operator must be an object of static storage class.</td>
</tr>
<tr>
<td>003</td>
<td>Only integers and pointers may be converted to pointers.</td>
</tr>
<tr>
<td>004</td>
<td>Attempt to create more than one instance of a task which waits for fast_event variable—task instance <code>(identifier)</code>.</td>
</tr>
<tr>
<td>005</td>
<td>Only numeric values may be converted to float.</td>
</tr>
<tr>
<td>006</td>
<td>Illegal operation on relocatable value in constant expression.</td>
</tr>
<tr>
<td>007</td>
<td>Illegal conversion from a structure or union type.</td>
</tr>
<tr>
<td>008</td>
<td>Operands of binary operator have incompatible types.</td>
</tr>
<tr>
<td>009</td>
<td>Illegal indirection through a non-pointer value.</td>
</tr>
<tr>
<td>010</td>
<td>An integral constant expression is required.</td>
</tr>
<tr>
<td>011</td>
<td>A scalar expression is required.</td>
</tr>
<tr>
<td>012</td>
<td>Bitfield values are not allowed in constant expressions.</td>
</tr>
<tr>
<td>013</td>
<td>Operands of <code>*</code>, <code>/</code>, and <code>%</code> must be numeric.</td>
</tr>
</tbody>
</table>

† Errors 001 – 699 are returned by the compiler. Errors 700 and higher are returned by the pre-processor.
<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>014</td>
<td>Operands of logical operators must be integers.</td>
</tr>
<tr>
<td>015</td>
<td>Pointer values being compared or subtracted in constant expressions must point to the same aggregate.</td>
</tr>
<tr>
<td>016</td>
<td>Assignment operators are invalid in constant expressions.</td>
</tr>
<tr>
<td>017</td>
<td>Operands of % operator may not be floating point.</td>
</tr>
<tr>
<td>018</td>
<td>The ++ and -- operators are invalid in constant expressions.</td>
</tr>
<tr>
<td>019</td>
<td>A non-relocatable constant expression is expected.</td>
</tr>
<tr>
<td>020</td>
<td>Relocatable quantities cannot be converted to float in constant expressions.</td>
</tr>
<tr>
<td>021</td>
<td>Void expressions are not permitted in constant expressions.</td>
</tr>
<tr>
<td>022</td>
<td>A structure or union is required for membership operators.</td>
</tr>
<tr>
<td>023</td>
<td>Attempt to apply a subscript to something other than an array or pointer.</td>
</tr>
<tr>
<td>024</td>
<td>Only integral values and pointers may be subtracted from pointers.</td>
</tr>
<tr>
<td>025</td>
<td>Pointers may only be subtracted from pointers of the same type.</td>
</tr>
</tbody>
</table>
| 026  | Undeclared variable `(identifier)`.
<p>| 027  | Constant expressions may not have type <code>void</code>. |
| 028  | Illegal implicit pointer-to-floating conversion. |
| 029  | Illegal implicit pointer-to-integer conversion. |
| 030  | Illegal implicit pointer-to-pointer conversion. |
| 031  | Illegal implicit integer-to-pointer conversion. |
| 032  | Illegal implicit floating-to-pointer conversion. |
| 033  | Illegal conversion. |
| 034  | Attempt to use an event variable in an arithmetic expression. |</p>
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>035</td>
<td>Parameter declarations are invalid with function prototypes.</td>
</tr>
<tr>
<td>036</td>
<td>Functions may not be initialized.</td>
</tr>
<tr>
<td>037</td>
<td>Task instances may not be initialized.</td>
</tr>
<tr>
<td>038</td>
<td>Typedefs may not be initialized.</td>
</tr>
<tr>
<td>039</td>
<td>Invalid initializer on function or task.</td>
</tr>
<tr>
<td>040</td>
<td>Array or structure initializers must be a list of constant expressions.</td>
</tr>
<tr>
<td>041</td>
<td>Attempt to initialize a bitfield with a relocatable value.</td>
</tr>
<tr>
<td>042</td>
<td>String is too long to fit into array.</td>
</tr>
<tr>
<td>043</td>
<td>Too many levels of braces in initializer.</td>
</tr>
<tr>
<td>044</td>
<td>Too many initializers.</td>
</tr>
<tr>
<td>045</td>
<td>Union (identifier) undefined.</td>
</tr>
<tr>
<td>046</td>
<td>Struct (identifier) undefined.</td>
</tr>
<tr>
<td>047</td>
<td>Task has more than one entrypoint.</td>
</tr>
<tr>
<td>048</td>
<td>File has more than one entrypoint.</td>
</tr>
<tr>
<td>049</td>
<td>A function exceeds 64K bytes in size.</td>
</tr>
<tr>
<td>050</td>
<td>Integral type expected.</td>
</tr>
<tr>
<td>051</td>
<td>Incompatible types.</td>
</tr>
<tr>
<td>052</td>
<td>Pointers must be of the same type.</td>
</tr>
<tr>
<td>053</td>
<td>Integral expression expected.</td>
</tr>
<tr>
<td>054</td>
<td>Illegal operands of minus.</td>
</tr>
<tr>
<td>055</td>
<td>Arithmetic types required.</td>
</tr>
</tbody>
</table>
Table A3-1 (Continued)

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>056</td>
<td>Division by zero.</td>
</tr>
<tr>
<td>057</td>
<td>Division by zero prohibited.</td>
</tr>
<tr>
<td>058</td>
<td>Illegal types.</td>
</tr>
<tr>
<td>059</td>
<td>Arithmetic types expected.</td>
</tr>
<tr>
<td>060</td>
<td>Integral types expected.</td>
</tr>
<tr>
<td>061</td>
<td>The operands of the (symbol) operator have incompatible types.</td>
</tr>
<tr>
<td>062</td>
<td>Operands of incompatible type to ‘(symbol)’ operator.</td>
</tr>
<tr>
<td>063</td>
<td>Branch condition must have scalar type.</td>
</tr>
<tr>
<td>064</td>
<td>Value of void function used.</td>
</tr>
<tr>
<td>065</td>
<td>Value of task invocation used.</td>
</tr>
<tr>
<td>066</td>
<td>Attempt to invoke an object which is not a function or task.</td>
</tr>
<tr>
<td>067</td>
<td>Argument must not be void.</td>
</tr>
<tr>
<td>068</td>
<td>Not enough arguments supplied in function call.</td>
</tr>
<tr>
<td>069</td>
<td>Too many arguments supplied in function call.</td>
</tr>
<tr>
<td>070</td>
<td>Unknown size.</td>
</tr>
<tr>
<td>071</td>
<td>Attempt to call bad function.</td>
</tr>
<tr>
<td>072</td>
<td>Extensive use of fast_event variables has caused a code segment to overflow its 64K byte limit.</td>
</tr>
<tr>
<td>073</td>
<td>(Identifier) undeclared.</td>
</tr>
<tr>
<td>074</td>
<td>The left operand of the DOT operator must be of structure or union type.</td>
</tr>
<tr>
<td>075</td>
<td>The left operand of the -&gt; operator must be either a pointer to a structure, or a pointer to a union.</td>
</tr>
</tbody>
</table>
Table A3-1 (Continued)

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>076</td>
<td>(Identifier) is an unknown member.</td>
</tr>
<tr>
<td>077</td>
<td>Illegal indirection or illegal subscript.</td>
</tr>
<tr>
<td>078</td>
<td>Illegal L-value.</td>
</tr>
<tr>
<td>079</td>
<td>Operand of prefix (symbol ++ or --) operator must be scalar.</td>
</tr>
<tr>
<td>080</td>
<td>Operand of postfix (symbol ++ or --) operator must be scalar.</td>
</tr>
<tr>
<td>081</td>
<td>Unary PLUS operator requires scalar operand.</td>
</tr>
<tr>
<td>082</td>
<td>Unary minus operator requires arithmetic operand.</td>
</tr>
<tr>
<td>083</td>
<td>Bitwise NOT operator requires integral operand.</td>
</tr>
<tr>
<td>084</td>
<td>DEFAULT not inside SWITCH.</td>
</tr>
<tr>
<td>085</td>
<td>Multiple DEFAULT's in switch.</td>
</tr>
<tr>
<td>086</td>
<td>Label '(identifier)' multiply defined.</td>
</tr>
<tr>
<td>087</td>
<td>BREAK outside of loop or switch.</td>
</tr>
<tr>
<td>088</td>
<td>CONTINUE outside of loop.</td>
</tr>
<tr>
<td>089</td>
<td>RETURN is invalid inside WAITFOR.</td>
</tr>
<tr>
<td>090</td>
<td>Void functions must not return a value.</td>
</tr>
<tr>
<td>095</td>
<td>Conflicting tag: (struct, union, enum, or task) (identifier) and (struct, union, enum, or task) (identifier).</td>
</tr>
<tr>
<td>091</td>
<td>Expression must have type 'label'.</td>
</tr>
<tr>
<td>092</td>
<td>Controlling expression must be integral.</td>
</tr>
<tr>
<td>093</td>
<td>CASE expression must be integral.</td>
</tr>
<tr>
<td>094</td>
<td>Duplicate CASE, value = (number).</td>
</tr>
<tr>
<td>Code</td>
<td>Message</td>
</tr>
<tr>
<td>------</td>
<td>---------</td>
</tr>
<tr>
<td>096</td>
<td>WAITFOR is invalid within another WAITFOR.</td>
</tr>
<tr>
<td>097</td>
<td>Attempt to wait for non-event variable.</td>
</tr>
<tr>
<td>098</td>
<td>Invalid L-value.</td>
</tr>
<tr>
<td>099</td>
<td>Attempt to modify CONST L-value.</td>
</tr>
<tr>
<td>100</td>
<td>Attempt to use an event value in an expression.</td>
</tr>
<tr>
<td>101</td>
<td>Attempt to use a label value in an expression.</td>
</tr>
<tr>
<td>102</td>
<td>Attempt to use a void value.</td>
</tr>
<tr>
<td>103</td>
<td>Arrays of functions or tasks are invalid.</td>
</tr>
<tr>
<td>104</td>
<td>Illegal storage class for function.</td>
</tr>
<tr>
<td>105</td>
<td>Illegal storage class for task instance.</td>
</tr>
<tr>
<td>106</td>
<td>Invalid storage class.</td>
</tr>
<tr>
<td>107</td>
<td>Extern variables may not be initialized within a function.</td>
</tr>
<tr>
<td>108</td>
<td>Function (identifier) redeclared.</td>
</tr>
<tr>
<td>109</td>
<td>Function (identifier) redefined.</td>
</tr>
<tr>
<td>110</td>
<td>(Identifier) redeclared.</td>
</tr>
<tr>
<td>111</td>
<td>(Identifier) redefined.</td>
</tr>
<tr>
<td>112</td>
<td>Typedef redefined.</td>
</tr>
<tr>
<td>113</td>
<td>Label '(identifier)' is undefined.</td>
</tr>
<tr>
<td>114</td>
<td>(1) Unknown size for (identifier).</td>
</tr>
<tr>
<td>115</td>
<td>(2) Unknown size for (identifier).</td>
</tr>
<tr>
<td>116</td>
<td>Enum (tag identifier) redeclared.</td>
</tr>
<tr>
<td>Number</td>
<td>Error Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------------</td>
</tr>
<tr>
<td>117</td>
<td>Newline in character constant.</td>
</tr>
<tr>
<td>118</td>
<td>Newline in string constant.</td>
</tr>
<tr>
<td>119</td>
<td>Unknown character.</td>
</tr>
<tr>
<td>120</td>
<td>Unexpected character.</td>
</tr>
<tr>
<td>121</td>
<td>Token type missing.</td>
</tr>
<tr>
<td>122</td>
<td>Wrong type of declarator for function definition.</td>
</tr>
<tr>
<td>123</td>
<td>Parameter # (number) has no identifier.</td>
</tr>
<tr>
<td>124</td>
<td>(Named item) is declared in the parameter declarations, but is not listed in the parameter list of the function.</td>
</tr>
<tr>
<td>125</td>
<td>Extra ; in function definition.</td>
</tr>
<tr>
<td>126</td>
<td>Syntax error in attribute.</td>
</tr>
<tr>
<td>127</td>
<td>Syntax error in declarator or initializer.</td>
</tr>
<tr>
<td>128</td>
<td>Syntax error in initializer.</td>
</tr>
<tr>
<td>129</td>
<td>Syntax error in parameter definition.</td>
</tr>
<tr>
<td>130</td>
<td>Attempt to initialize a formal parameter.</td>
</tr>
<tr>
<td>131</td>
<td>Syntax error in parameter.</td>
</tr>
<tr>
<td>132</td>
<td>Syntax error in struct/union declaration.</td>
</tr>
<tr>
<td>133</td>
<td>Syntax error in type specifier.</td>
</tr>
<tr>
<td>134</td>
<td>Syntax error in structure member.</td>
</tr>
<tr>
<td>135</td>
<td>Syntax error in enumerator list.</td>
</tr>
<tr>
<td>136</td>
<td>Syntax error in enumerator.</td>
</tr>
<tr>
<td>Line</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>137</td>
<td>Syntax error in task specifier.</td>
</tr>
<tr>
<td>138</td>
<td>Syntax error in array subscript.</td>
</tr>
<tr>
<td>139</td>
<td>Syntax error in parameter list.</td>
</tr>
<tr>
<td>140</td>
<td>Unresolved reference (identifier).</td>
</tr>
<tr>
<td>141</td>
<td>Syntax error in statement.</td>
</tr>
<tr>
<td>142</td>
<td>Syntax error in conditional expression.</td>
</tr>
<tr>
<td>143</td>
<td>Syntax error in DO statement.</td>
</tr>
<tr>
<td>144</td>
<td>Syntax error in condition.</td>
</tr>
<tr>
<td>145</td>
<td>Syntax error in BREAK statement.</td>
</tr>
<tr>
<td>146</td>
<td>Syntax error in CONTINUE statement.</td>
</tr>
<tr>
<td>147</td>
<td>Syntax error in RETURN statement.</td>
</tr>
<tr>
<td>148</td>
<td>Syntax error in GOTO expression.</td>
</tr>
<tr>
<td>149</td>
<td>Syntax error in TASK list.</td>
</tr>
<tr>
<td>150</td>
<td>Syntax error in FOR statement.</td>
</tr>
<tr>
<td>151</td>
<td>Syntax error in FOR initialization.</td>
</tr>
<tr>
<td>152</td>
<td>Syntax error in FOR condition.</td>
</tr>
<tr>
<td>153</td>
<td>Syntax error in FOR increment.</td>
</tr>
<tr>
<td>154</td>
<td>Syntax error in SWITCH condition.</td>
</tr>
<tr>
<td>155</td>
<td>Syntax error in CASE expression.</td>
</tr>
<tr>
<td>156</td>
<td>Syntax error in compound statement.</td>
</tr>
</tbody>
</table>
### Table A3-1 (Continued)

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>157</td>
<td>Syntax error in <code>sizeof</code> type name.</td>
</tr>
<tr>
<td>158</td>
<td>Syntax error in subscript.</td>
</tr>
<tr>
<td>159</td>
<td>Syntax error in function call.</td>
</tr>
<tr>
<td>160</td>
<td>Syntax error in expression.</td>
</tr>
<tr>
<td>161</td>
<td>Type clash.</td>
</tr>
<tr>
<td>162</td>
<td>More than one storage class.</td>
</tr>
<tr>
<td>163</td>
<td>Array of unknown size.</td>
</tr>
<tr>
<td>164</td>
<td>Cannot take size of function.</td>
</tr>
<tr>
<td>165</td>
<td>Structure or union of unknown size.</td>
</tr>
<tr>
<td>166</td>
<td>Circular definition of enumerated type.</td>
</tr>
<tr>
<td>167</td>
<td>Cannot take size of task.</td>
</tr>
<tr>
<td>168</td>
<td><code>Struct</code> (identifier) redeclared.</td>
</tr>
<tr>
<td>169</td>
<td>Functions and tasks may not be structure members.</td>
</tr>
<tr>
<td>170</td>
<td>Zero-width bitfields may not be named.</td>
</tr>
<tr>
<td>171</td>
<td>Structure member (identifier) multiply defined.</td>
</tr>
<tr>
<td>172</td>
<td>Invalid negative bit-field width.</td>
</tr>
<tr>
<td>173</td>
<td><code>(Struct, union, enum, or task)</code> (identifier) multiply defined.</td>
</tr>
<tr>
<td>174</td>
<td>Task (identifier) redeclared.</td>
</tr>
<tr>
<td>175</td>
<td>Union (identifier) redeclared.</td>
</tr>
<tr>
<td>176</td>
<td>Functions and tasks may not be union members.</td>
</tr>
<tr>
<td></td>
<td>Description</td>
</tr>
<tr>
<td>---</td>
<td>-------------</td>
</tr>
<tr>
<td>177</td>
<td>Invalid zero-bit member.</td>
</tr>
<tr>
<td>178</td>
<td>Union member (identifier) multiply defined.</td>
</tr>
<tr>
<td>179</td>
<td>No main routine supplied.</td>
</tr>
<tr>
<td>180</td>
<td>Arrow operator given structure, not a pointer.</td>
</tr>
<tr>
<td>181</td>
<td>DOT operator given a pointer to a structure, not a structure.</td>
</tr>
<tr>
<td>182</td>
<td>Address of array.</td>
</tr>
<tr>
<td>183</td>
<td>Address of function.</td>
</tr>
<tr>
<td>184</td>
<td>Address of register variable.</td>
</tr>
<tr>
<td>185</td>
<td>Address of bit-field.</td>
</tr>
<tr>
<td>186</td>
<td>Address of non-L-value.</td>
</tr>
<tr>
<td>187</td>
<td>Attempt to use a LABEL value in an expression.</td>
</tr>
<tr>
<td>188</td>
<td>Attempt to use an EVENT value in an expression.</td>
</tr>
<tr>
<td>189</td>
<td>Invalid zero or negative array dimension.</td>
</tr>
<tr>
<td>190</td>
<td>Only const or volatile allowed.</td>
</tr>
<tr>
<td>191</td>
<td>Maximum bit-field width is 16.</td>
</tr>
<tr>
<td>192</td>
<td>Illegal storage class for formal parameter.</td>
</tr>
<tr>
<td>193</td>
<td>Function parameters may not be functions.</td>
</tr>
<tr>
<td>194</td>
<td>Function parameters may not be tasks.</td>
</tr>
<tr>
<td>195</td>
<td>Bad parameter storage class.</td>
</tr>
<tr>
<td>196</td>
<td>Event expression required in waitfor clause.</td>
</tr>
<tr>
<td>Number</td>
<td>Error Message</td>
</tr>
<tr>
<td>--------</td>
<td>---------------</td>
</tr>
<tr>
<td>197</td>
<td>Scalars must be initialized with a single expression, optionally in braces.</td>
</tr>
<tr>
<td>198</td>
<td>Label undeclared.</td>
</tr>
<tr>
<td>199</td>
<td>Syntax error in declarator or initializer.</td>
</tr>
<tr>
<td>200</td>
<td>Pointers to different objects shouldn’t be subtracted.</td>
</tr>
<tr>
<td>201</td>
<td>Duplicate formal parameters of a function.</td>
</tr>
<tr>
<td>202</td>
<td>External variables may not be initialized inside of a function.</td>
</tr>
<tr>
<td>203</td>
<td>Formal parameters of functions may not be initialized.</td>
</tr>
<tr>
<td>204</td>
<td>Attempt to use labels outside a function.</td>
</tr>
<tr>
<td>205</td>
<td>Variable (identifier) undeclared.</td>
</tr>
<tr>
<td>206</td>
<td>Attempt to take the value of a typedef.</td>
</tr>
<tr>
<td>207</td>
<td>Function’s stack frame is too large.</td>
</tr>
<tr>
<td>208</td>
<td>Floating point has not yet been implemented.</td>
</tr>
<tr>
<td>209</td>
<td>Invalid conversion of relocatable quantity in constant expression.</td>
</tr>
<tr>
<td>210</td>
<td>Attempt to redefine the reserved name ‘(identifier)’.</td>
</tr>
<tr>
<td>211</td>
<td>CASE outside of switch statement.</td>
</tr>
<tr>
<td>212</td>
<td>Returned values of this size are not implemented.</td>
</tr>
<tr>
<td>213</td>
<td>Unrecoverable syntax error.</td>
</tr>
<tr>
<td>214</td>
<td>Parsing stack overflow.</td>
</tr>
<tr>
<td>215</td>
<td>Too many errors have been encountered during compilation.</td>
</tr>
<tr>
<td>216</td>
<td>Compiler aborted.</td>
</tr>
</tbody>
</table>
### Table A3-1 (Continued)

<table>
<thead>
<tr>
<th></th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>217</td>
<td>Register variable ‘(identifier)’ declared with non-scalar type.</td>
</tr>
<tr>
<td>218</td>
<td>Implicit declaration of function ‘(identifier)’.</td>
</tr>
<tr>
<td>219</td>
<td>Out of memory during compilation—program too big.</td>
</tr>
<tr>
<td>220</td>
<td>Internal software error in compiler (error number). Compilation aborted.</td>
</tr>
<tr>
<td>221</td>
<td>No T1 Mux Installed.</td>
</tr>
<tr>
<td>222</td>
<td>A waitfor statement has one or more condition clauses, none of which names an event variable. This is often caused by either misspelling an event variable, or by failing to declare an event variable.</td>
</tr>
<tr>
<td>223</td>
<td>The variable ‘(identifier)’ has been declared inside of a task with the “extern” attribute, but has never been defined within that task. In this context, the keyword “extern” may only be used to forward-declare an identifier which is fully defined later in the task body.</td>
</tr>
<tr>
<td>226</td>
<td>Invalid or Incompatible Data Acquisition Tracks on selected playback device.</td>
</tr>
<tr>
<td>227</td>
<td>No ISDN Mux Installed.</td>
</tr>
<tr>
<td>230</td>
<td>Object file (name) is in obsolete format. Fix by recompiling it.</td>
</tr>
<tr>
<td>231</td>
<td>Symbol (identifier) multiply defined by object file.</td>
</tr>
<tr>
<td>232</td>
<td>The symbol (identifier) has been used as an event variable in one module and as a function or variable in another.</td>
</tr>
<tr>
<td>233</td>
<td>The symbol (identifier) has type event in one module, but has type fast_event in another.</td>
</tr>
<tr>
<td>234</td>
<td>Different modules have used the symbol (identifier) inconsistently as code, data, or read-only data.</td>
</tr>
<tr>
<td>235</td>
<td>Bad format in object file (name).</td>
</tr>
<tr>
<td>236</td>
<td>Cannot find object file (name).</td>
</tr>
</tbody>
</table>
### Table A3-1 (Continued)

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>700</td>
<td>#else inside of #else clause.</td>
</tr>
<tr>
<td>701</td>
<td>#elif inside of #else clause.</td>
</tr>
<tr>
<td>702</td>
<td>Too many nested #if's.</td>
</tr>
<tr>
<td>703</td>
<td>Extra tokens at end of line.</td>
</tr>
<tr>
<td>704</td>
<td>Unexpected end of file.</td>
</tr>
<tr>
<td>705</td>
<td>Identifier missing from #ifdef directive.</td>
</tr>
<tr>
<td>706</td>
<td>Identifier missing from #ifndef directive.</td>
</tr>
<tr>
<td>707</td>
<td>#elif without matching #if.</td>
</tr>
<tr>
<td>708</td>
<td>#else without matching #if.</td>
</tr>
<tr>
<td>709</td>
<td>#endif without matching #if.</td>
</tr>
<tr>
<td>710</td>
<td>Syntax error.</td>
</tr>
<tr>
<td>711</td>
<td>Syntax error in constant expression.</td>
</tr>
<tr>
<td>712</td>
<td>String constant in constant expression.</td>
</tr>
<tr>
<td>713</td>
<td>Invalid character in constant expression.</td>
</tr>
<tr>
<td>714</td>
<td>Error in hex number.</td>
</tr>
<tr>
<td>715</td>
<td>End of file in char constant.</td>
</tr>
<tr>
<td>716</td>
<td>Newline in char constant.</td>
</tr>
<tr>
<td>717</td>
<td>End of file in string.</td>
</tr>
<tr>
<td>718</td>
<td>Newline inside string.</td>
</tr>
<tr>
<td>719</td>
<td>Attempt to divide by zero.</td>
</tr>
<tr>
<td>720</td>
<td>Unknown preprocessor command.</td>
</tr>
</tbody>
</table>
Table A3-1 (Continued)

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>721</td>
<td>Syntax error in formal parameters of macro.</td>
</tr>
<tr>
<td>722</td>
<td>Duplication of formal parameter (identifier) in macro definition.</td>
</tr>
<tr>
<td>723</td>
<td>No macro name given.</td>
</tr>
<tr>
<td>724</td>
<td>Macro redefined.</td>
</tr>
<tr>
<td>725</td>
<td>Syntax error in #line directive.</td>
</tr>
<tr>
<td>726</td>
<td>Unterminated string literal.</td>
</tr>
<tr>
<td>727</td>
<td>Cannot open include file (identifier).</td>
</tr>
<tr>
<td>728</td>
<td>Cannot find include file (identifier).</td>
</tr>
<tr>
<td>729</td>
<td>Identifier does not exist.</td>
</tr>
<tr>
<td>730</td>
<td>Syntax error in #include directive.</td>
</tr>
<tr>
<td>731</td>
<td>Include identifier is not defined.</td>
</tr>
<tr>
<td>732</td>
<td>Unterminated character constant.</td>
</tr>
<tr>
<td>733</td>
<td>End of file inside char constant.</td>
</tr>
<tr>
<td>734</td>
<td>End of file inside string.</td>
</tr>
<tr>
<td>735</td>
<td>End of file inside comment.</td>
</tr>
<tr>
<td>736</td>
<td>Argument list required.</td>
</tr>
<tr>
<td>737</td>
<td>Attempt to close bracket [ or ( with ).</td>
</tr>
<tr>
<td>738</td>
<td>Attempt to close arg list with }.</td>
</tr>
<tr>
<td>739</td>
<td>Attempt to close bracket [ or ( with }.</td>
</tr>
<tr>
<td>740</td>
<td>Attempt to close arg list with ].</td>
</tr>
<tr>
<td>Error Message</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| 741           | Attempt to close bracket ( or { with }.
| 742           | Incomplete argument list.
| 743           | No parameter after a # char.
| 744           | (User-generated error message.)
| 745           | File ends with \.
| 746           | Number of arguments does not match number of parameters.
| 749           | Identifier missing from #undef.
Appendix B: Glossary of Acronyms, Abbreviations and Mnemonics

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACK</td>
<td>Acknowledgment</td>
</tr>
<tr>
<td>ACTLU</td>
<td>Activate Logical Unit (SNA)</td>
</tr>
<tr>
<td>ADR</td>
<td>Address</td>
</tr>
<tr>
<td>AK</td>
<td>ACK: Acknowledgment</td>
</tr>
<tr>
<td>ANSI</td>
<td>American National Standards Institute</td>
</tr>
<tr>
<td>ASCII</td>
<td>American Standard Code for Information Interchange, standard code for digital communications</td>
</tr>
<tr>
<td>ASYNC</td>
<td>Asynchronous format (indicates START and STOP bits)</td>
</tr>
<tr>
<td>AUX</td>
<td>Auxiliary</td>
</tr>
<tr>
<td>BAUDOT</td>
<td>Five bit code for data transmission using one start and one stop element; used in some teletypewriter machines</td>
</tr>
<tr>
<td>BBI</td>
<td>Begin Bracket Indicator (SNA)</td>
</tr>
<tr>
<td>BCC</td>
<td>Block Check Calculation</td>
</tr>
<tr>
<td>BCN</td>
<td>Beacon (SDLC)</td>
</tr>
<tr>
<td>BDLCL</td>
<td>Burroughs Data Link Control</td>
</tr>
<tr>
<td>BEL</td>
<td>Bell</td>
</tr>
<tr>
<td>BERT</td>
<td>Bit Error Rate Testing</td>
</tr>
<tr>
<td>BIB</td>
<td>Backward Indicator Bit (SS#7 Layer 2)</td>
</tr>
<tr>
<td>BISYNC</td>
<td>Binary Synchronous Communications Protocol (IBM); also BSC</td>
</tr>
<tr>
<td>BITIM</td>
<td>Bit image</td>
</tr>
<tr>
<td>BL</td>
<td>BEL: Bell</td>
</tr>
<tr>
<td>BLI</td>
<td>Blink (CRT enhancement)</td>
</tr>
<tr>
<td>BM</td>
<td>Bit Mask</td>
</tr>
<tr>
<td>BNC</td>
<td>A highly reliable twist–lock connector used to carry wide–band video/digital signals: used with coaxial cable (G.703)</td>
</tr>
<tr>
<td>BOP</td>
<td>Bit–Oriented Protocol, e.g., SDLC</td>
</tr>
<tr>
<td>bps</td>
<td>Bits per second</td>
</tr>
<tr>
<td>BS</td>
<td>Backspace</td>
</tr>
<tr>
<td>BSC</td>
<td>BISYNC</td>
</tr>
<tr>
<td>Symbol</td>
<td>Definition</td>
</tr>
<tr>
<td>--------</td>
<td>------------</td>
</tr>
<tr>
<td>C</td>
<td>Control (X.21 signal)</td>
</tr>
<tr>
<td>CAN</td>
<td>Cancel</td>
</tr>
<tr>
<td>CAS</td>
<td>Channel Associated Signaling (G.703)</td>
</tr>
<tr>
<td>CCITT</td>
<td>Consultative Committee, International Telephone and Telegraph</td>
</tr>
<tr>
<td>CCS</td>
<td>Clear Channel Signaling (G.703)</td>
</tr>
<tr>
<td>CCSS#7</td>
<td>Common Channel Signaling System #7</td>
</tr>
<tr>
<td>CD</td>
<td>Carrier Detect (RS-232/V.24 and V.35 signal); same as RLSD</td>
</tr>
<tr>
<td>CDI</td>
<td>Change Direction Indicator (SNA)</td>
</tr>
<tr>
<td>CEDI</td>
<td>Conditional End Bracket Indicator (SNA)</td>
</tr>
<tr>
<td>CF</td>
<td>Command Format (SNA)</td>
</tr>
<tr>
<td>CHAR</td>
<td>Character</td>
</tr>
<tr>
<td>CHDAT</td>
<td>Character data</td>
</tr>
<tr>
<td>CIC</td>
<td>Circuit Identifier Code (SS#7 Layer 3)</td>
</tr>
<tr>
<td>CLR</td>
<td>Clear</td>
</tr>
<tr>
<td>CN</td>
<td>CAN: Cancel</td>
</tr>
<tr>
<td>CONF</td>
<td>Confirm</td>
</tr>
<tr>
<td>const</td>
<td>Constant, modifier to a declaration in C language</td>
</tr>
<tr>
<td>CPM</td>
<td>Central Processing Module</td>
</tr>
<tr>
<td>CR</td>
<td>Carriage Return</td>
</tr>
<tr>
<td>CRC</td>
<td>Cyclic Redundancy Check</td>
</tr>
<tr>
<td>CSI</td>
<td>Code Selection Indicator (SNA)</td>
</tr>
<tr>
<td>CSN</td>
<td>Command Sequence Number (SNA)</td>
</tr>
<tr>
<td>CTS</td>
<td>Clear To Send (RS-232/V.24 and V.35 signal)</td>
</tr>
<tr>
<td>D</td>
<td>D bit (Bit 7 in first octet of packet-level X.25)</td>
</tr>
<tr>
<td>DAF</td>
<td>Destination Address Field (SNA)</td>
</tr>
<tr>
<td>DAT</td>
<td>Data Acquisition Tracks</td>
</tr>
<tr>
<td>DB-25</td>
<td>25-Pin D connector (standard for RS-232/V.24)</td>
</tr>
<tr>
<td>DC1</td>
<td>Device Control 1</td>
</tr>
<tr>
<td>DC3</td>
<td>Device Control 3</td>
</tr>
<tr>
<td>DCE</td>
<td>Data Circuit-terminating Equipment (or Data Communications Equipment), typically a modem</td>
</tr>
<tr>
<td>DCF</td>
<td>Data Count Field (SNA)</td>
</tr>
<tr>
<td>DDCMP</td>
<td>Digital Data Communications Message Protocol</td>
</tr>
<tr>
<td>DEC</td>
<td>Decrement</td>
</tr>
<tr>
<td>DEF</td>
<td>Destination Element Field (SNA)</td>
</tr>
<tr>
<td>#define</td>
<td>Preprocessor directive, C language</td>
</tr>
<tr>
<td>DEL</td>
<td>Delete</td>
</tr>
<tr>
<td>DFC</td>
<td>Data Flow Control (SNA)</td>
</tr>
<tr>
<td>DIAG</td>
<td>Diagnostic (X.25 Layer 3)</td>
</tr>
<tr>
<td>DIR</td>
<td>Directory</td>
</tr>
<tr>
<td>DISC</td>
<td>Disconnect (SDLC, LAPD, X.25 Layer 2)</td>
</tr>
<tr>
<td>DL</td>
<td>DLE: Data Link Escape; also Data Link layer (OSI primitive)</td>
</tr>
<tr>
<td>DLC</td>
<td>Data Link Control</td>
</tr>
<tr>
<td>DLE</td>
<td>Data Link Escape (used principally in transparent BISYNC)</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>DM</td>
<td>Disconnected Mode (SDLC, LAPD, X.25 Layer 2)</td>
</tr>
<tr>
<td>DMA</td>
<td>Direct Memory Access</td>
</tr>
<tr>
<td>DPC</td>
<td>Destination Point Code (SS#7 Layer 2)</td>
</tr>
<tr>
<td>DRAM</td>
<td>Dynamic Random Access Memory; one Mbyte of memory space of each MPM, dedicated to storage or receive data</td>
</tr>
<tr>
<td>DRI</td>
<td>Direct Response Indicator (SNA)</td>
</tr>
<tr>
<td>DSAF</td>
<td>Destination Subarea Address Field (SNA)</td>
</tr>
<tr>
<td>DSK</td>
<td>Disk</td>
</tr>
<tr>
<td>DSR</td>
<td>Data Set Ready (RS-232/V.24 and V.35 control lead)</td>
</tr>
<tr>
<td>DTE</td>
<td>Data Terminal Equipment</td>
</tr>
<tr>
<td>DTR</td>
<td>Data Terminal Ready (RS-232/V.24 and V.35 control lead)</td>
</tr>
<tr>
<td>DUP</td>
<td>Duplicate</td>
</tr>
<tr>
<td>EB</td>
<td>ETB, EOB: End of Transmission Block</td>
</tr>
<tr>
<td>EBCD</td>
<td>Extended Binary Coded Decimal</td>
</tr>
<tr>
<td>EBCDIC</td>
<td>Extended Binary Coded Decimal Interchange Code</td>
</tr>
<tr>
<td>EBI</td>
<td>End Bracket Indicator (SNA)</td>
</tr>
<tr>
<td>EC</td>
<td>ESC: Escape</td>
</tr>
<tr>
<td>EIA</td>
<td>Electronic Industries Association</td>
</tr>
<tr>
<td>#elif</td>
<td>Else if, preprocessor directive, C language</td>
</tr>
<tr>
<td>EM</td>
<td>EOM: End of Message</td>
</tr>
<tr>
<td>#endif</td>
<td>Preprocessor directive, C language</td>
</tr>
<tr>
<td>ENQ</td>
<td>Enquiry</td>
</tr>
<tr>
<td>enum</td>
<td>Enumeration, set of integer constants, C language</td>
</tr>
<tr>
<td>EOB</td>
<td>End of Transmission Block</td>
</tr>
<tr>
<td>EOF</td>
<td>End of Frame</td>
</tr>
<tr>
<td>EOM</td>
<td>End of Message</td>
</tr>
<tr>
<td>EOT</td>
<td>End of Transmission</td>
</tr>
<tr>
<td>EPROM</td>
<td>PROM containing power-up software and initialization routines</td>
</tr>
<tr>
<td>EQ</td>
<td>Equal</td>
</tr>
<tr>
<td>EQ</td>
<td>ENQ: Enquiry</td>
</tr>
<tr>
<td>ERI</td>
<td>Exception Response Indicator</td>
</tr>
<tr>
<td>ERN</td>
<td>Explicit Route Number</td>
</tr>
<tr>
<td>ESC</td>
<td>Escape</td>
</tr>
<tr>
<td>ET</td>
<td>EOT: End of Transmission</td>
</tr>
<tr>
<td>ETB</td>
<td>End of Transmission Block</td>
</tr>
<tr>
<td>ETX</td>
<td>End of Text</td>
</tr>
<tr>
<td>evar</td>
<td>Event variable, pre-declared identifier, ARD extension to C language</td>
</tr>
<tr>
<td>EX</td>
<td>ETX: End of Text</td>
</tr>
<tr>
<td>extern</td>
<td>External, storage class specifier, C language</td>
</tr>
<tr>
<td>FAC</td>
<td>Facilities (X.25 Layer 3)</td>
</tr>
<tr>
<td>FAS</td>
<td>Frame Alignment Signal (G.703)</td>
</tr>
<tr>
<td>FCS</td>
<td>Frame Check Sequence (used in BOP)</td>
</tr>
<tr>
<td>FD1</td>
<td>Floppy-disk Drive 1</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Definition</td>
</tr>
<tr>
<td>--------------</td>
<td>------------</td>
</tr>
<tr>
<td>FD2</td>
<td>Floppy-disk Drive 2</td>
</tr>
<tr>
<td>FDX</td>
<td>Full duplex (permits simultaneous data in both directions)</td>
</tr>
<tr>
<td>FEB</td>
<td>Front End Buffer</td>
</tr>
<tr>
<td>fevar</td>
<td>Fast event variable, pre-declared identifier, ARD extension to C language</td>
</tr>
<tr>
<td>FF</td>
<td>Form Feed</td>
</tr>
<tr>
<td>FI</td>
<td>Format Indicator (SNA)</td>
</tr>
<tr>
<td>FIB</td>
<td>Forward Indicator Bit (SS#7 Layer 2)</td>
</tr>
<tr>
<td>FID</td>
<td>Format Identifier (SNA)</td>
</tr>
<tr>
<td>fifo</td>
<td>First in, first out; memory queue on boards</td>
</tr>
<tr>
<td>FMD</td>
<td>Function Management Data (SNA)</td>
</tr>
<tr>
<td>FRMR</td>
<td>Frame Reject (SDLC, LAPD, X.25 Layer 2)</td>
</tr>
<tr>
<td>FS</td>
<td>Field Separator</td>
</tr>
<tr>
<td>FSN</td>
<td>Forward Sequence Number (SS#7 Layer 2)</td>
</tr>
<tr>
<td>GBM</td>
<td>Global Bus Module</td>
</tr>
<tr>
<td>GE</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>GFI</td>
<td>Group Format Indicator (X.25 Layer 3)</td>
</tr>
<tr>
<td>goto</td>
<td>Jump statement, C language</td>
</tr>
<tr>
<td>GS</td>
<td>Group Separator</td>
</tr>
<tr>
<td>GT</td>
<td>Greater than</td>
</tr>
<tr>
<td>HDLC</td>
<td>High Level Data Link Control procedure</td>
</tr>
<tr>
<td>HDX</td>
<td>Half duplex (data cannot be transmitted in both directions simultaneously)</td>
</tr>
<tr>
<td>HEX</td>
<td>Hexadecimal number; also the hex key</td>
</tr>
<tr>
<td>HRD</td>
<td>Hard disk</td>
</tr>
<tr>
<td>Hz</td>
<td>Hertz</td>
</tr>
<tr>
<td>I</td>
<td>Indication (X.21 signal)</td>
</tr>
<tr>
<td>I</td>
<td>Information (SDLC, LAPD, X.25 Layer 2)</td>
</tr>
<tr>
<td>iAPX 286</td>
<td>Part number for Intel 80286 processor</td>
</tr>
<tr>
<td>IERN</td>
<td>Initial Explicit Route Number (SNA)</td>
</tr>
<tr>
<td>#if</td>
<td>Preprocessor directive, C language</td>
</tr>
<tr>
<td>#ifdef</td>
<td>If defined..., preprocessor directive, C language</td>
</tr>
<tr>
<td>#ifndef</td>
<td>If not defined..., preprocessor directive, C language</td>
</tr>
<tr>
<td>IL</td>
<td>Interlayer (message buffer)</td>
</tr>
<tr>
<td>INC</td>
<td>Increment</td>
</tr>
<tr>
<td>#include</td>
<td>Preprocessor directive, C language</td>
</tr>
<tr>
<td>IND</td>
<td>Indication</td>
</tr>
<tr>
<td>INFO</td>
<td>Information (SDLC, LAPD, X.25 Layer 2)</td>
</tr>
<tr>
<td>init</td>
<td>Initialize</td>
</tr>
<tr>
<td>int</td>
<td>Integer data type, C language</td>
</tr>
<tr>
<td>INT</td>
<td>Interrupt (X.25 Layer 3)</td>
</tr>
<tr>
<td>I/O</td>
<td>Input/Output</td>
</tr>
<tr>
<td>IPARS</td>
<td>International Passenger Airlines Reservation System</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Definition</td>
</tr>
<tr>
<td>--------------</td>
<td>------------</td>
</tr>
<tr>
<td>ISDN</td>
<td>Integrated Services Digital Network</td>
</tr>
<tr>
<td>ISO</td>
<td>International Standards Organization</td>
</tr>
<tr>
<td>ISOC</td>
<td>Isochronous</td>
</tr>
<tr>
<td>JIS</td>
<td>Japanese Industrial Standard</td>
</tr>
<tr>
<td>kana</td>
<td>Japanese syllabic alphabet</td>
</tr>
<tr>
<td>Kbits</td>
<td>Kilobits</td>
</tr>
<tr>
<td>Kbps</td>
<td>Kilobits per second</td>
</tr>
<tr>
<td>Kbyte</td>
<td>Kilobyte</td>
</tr>
<tr>
<td>LAF</td>
<td>Local Address Field (SNA)</td>
</tr>
<tr>
<td>LAPD</td>
<td>Link Access Procedure on the D-channel</td>
</tr>
<tr>
<td>LCG</td>
<td>Logical Channel Group (X.25 Layer 3)</td>
</tr>
<tr>
<td>LCN</td>
<td>Logical Channel Number (X.25 Layer 3)</td>
</tr>
<tr>
<td>LE</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>LED</td>
<td>Light Emitting Diode</td>
</tr>
<tr>
<td>LF</td>
<td>Line Feed</td>
</tr>
<tr>
<td>LI</td>
<td>Length Indicator (SS#7 Layer 2)</td>
</tr>
<tr>
<td>LOBJ</td>
<td>Linkable-object</td>
</tr>
<tr>
<td>LRC</td>
<td>Longitudinal Redundancy Check</td>
</tr>
<tr>
<td>LSU</td>
<td>Link Status Unit (SS#7 Layer 2)</td>
</tr>
<tr>
<td>LT</td>
<td>Less than</td>
</tr>
<tr>
<td>LTA</td>
<td>Link Test Acknowledge</td>
</tr>
<tr>
<td>LTM</td>
<td>Link Test Message</td>
</tr>
<tr>
<td>LU</td>
<td>Logical Unit (SNA)</td>
</tr>
<tr>
<td>M</td>
<td>M bit (X.25 Layer 3, Bit 4 of first octet)</td>
</tr>
<tr>
<td>macro</td>
<td>Macro replacement of text initiated by define pre-processor directive, C language</td>
</tr>
<tr>
<td>Mbyte</td>
<td>Megabyte</td>
</tr>
<tr>
<td>MOD</td>
<td>Modulus; maximum window size for frames or packets, 8 or 128</td>
</tr>
<tr>
<td>MPM</td>
<td>Main Processing Module</td>
</tr>
<tr>
<td>msec</td>
<td>Millisecond</td>
</tr>
<tr>
<td>MSU</td>
<td>Message Signal Unit (SS#7 Layer 2)</td>
</tr>
<tr>
<td>mux</td>
<td>Multiplexer</td>
</tr>
<tr>
<td>N</td>
<td>Network layer (OSI primitive)</td>
</tr>
<tr>
<td>NAK</td>
<td>Negative Acknowledgment</td>
</tr>
<tr>
<td>NC</td>
<td>Network Control (SNA)</td>
</tr>
<tr>
<td>NE</td>
<td>Not equal to</td>
</tr>
<tr>
<td>NETM</td>
<td>Network Management (SS#7 Layer 3)</td>
</tr>
<tr>
<td>NI</td>
<td>Network Indicator</td>
</tr>
<tr>
<td>NK</td>
<td>NAK: Negative Acknowledgment</td>
</tr>
<tr>
<td>NP</td>
<td>Network Priority (SNA)</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>Nr</td>
<td>Number (next) receive frame (SDLC, LAPD, X.25 Layer 2); also NR and N(R)</td>
</tr>
<tr>
<td>NRZI</td>
<td>Non-Return to Zero Inverted (used with SDLC and ASYNC modems—sometimes with clocked modems)</td>
</tr>
<tr>
<td>Ns</td>
<td>Number (frame) sent (SDLC, LAPD, X.25 Layer 2); also NS and N(S)</td>
</tr>
<tr>
<td>NT</td>
<td>Network Termination (ISDN)</td>
</tr>
<tr>
<td>NTR</td>
<td>Network Test Regular (SS#7 Layer 3)</td>
</tr>
<tr>
<td>NTS</td>
<td>Network Test Special (SS#7 Layer 3)</td>
</tr>
<tr>
<td>NU</td>
<td>NUL: Null</td>
</tr>
<tr>
<td>NUL</td>
<td>Null</td>
</tr>
<tr>
<td>OBJ</td>
<td>Object code</td>
</tr>
<tr>
<td>OEF</td>
<td>Origin Element Field (SNA)</td>
</tr>
<tr>
<td>OPC</td>
<td>Originating Point Code (SS#7 Layer 3)</td>
</tr>
<tr>
<td>OSAF</td>
<td>Origin Subarea Address Field (SNA)</td>
</tr>
<tr>
<td>OSI</td>
<td>Open Systems Interconnection</td>
</tr>
<tr>
<td>OUTSYNC</td>
<td>Out of synchronization</td>
</tr>
<tr>
<td>pad</td>
<td>DEL or idle line character</td>
</tr>
<tr>
<td>pal</td>
<td>Programmable array logic</td>
</tr>
<tr>
<td>paren</td>
<td>Parentheses</td>
</tr>
<tr>
<td>PCM</td>
<td>Peripheral Control Module</td>
</tr>
<tr>
<td>PDU</td>
<td>Primitive Data Unit</td>
</tr>
<tr>
<td>PERC</td>
<td>Percentage</td>
</tr>
<tr>
<td>P/F</td>
<td>Poll/Final bit used in control byte at frame level (SDLC, X.25)</td>
</tr>
<tr>
<td>PH</td>
<td>Physical layer (OSI primitive)</td>
</tr>
<tr>
<td>PI</td>
<td>Pacing Indicator (SNA)</td>
</tr>
<tr>
<td>PIU</td>
<td>Path Information Unit (SNA)</td>
</tr>
<tr>
<td>PLU</td>
<td>Primary Logical Unit (SNA)</td>
</tr>
<tr>
<td>pos</td>
<td>Position</td>
</tr>
<tr>
<td>Pr</td>
<td>Packet (next) receive sequence number (X.25 Layer 3); also PR and P(R)</td>
</tr>
<tr>
<td>#pragma</td>
<td>Preprocessor directive, C language</td>
</tr>
<tr>
<td>PRGM</td>
<td>Program</td>
</tr>
<tr>
<td>PROG TR</td>
<td>Program Trace Run-mode screen</td>
</tr>
<tr>
<td>PROM</td>
<td>Programmable Read-Only Memory</td>
</tr>
<tr>
<td>Ps</td>
<td>Packet send sequence number (X.25 Layer 3); also PS and P(S)</td>
</tr>
<tr>
<td>PU</td>
<td>Physical Unit (SNA)</td>
</tr>
<tr>
<td>Q</td>
<td>Q bit (Bit 8 of first octet in packet-level X.25)</td>
</tr>
<tr>
<td>QRI</td>
<td>Queued Response Indicator (SNA)</td>
</tr>
<tr>
<td>R</td>
<td>Receive (X.21 signal)</td>
</tr>
<tr>
<td>RAM</td>
<td>Random Access Memory</td>
</tr>
<tr>
<td>RD</td>
<td>Received Data (RS-232/V.24 and V.35 signal)</td>
</tr>
<tr>
<td>REG</td>
<td>Registration (X.25 Layer 3)</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>REJ</td>
<td>Reject (SDLC, LAPD, X.25 Layer 2)</td>
</tr>
<tr>
<td>REQ</td>
<td>Request</td>
</tr>
<tr>
<td>RESP</td>
<td>Response</td>
</tr>
<tr>
<td>Rev</td>
<td>Reverse</td>
</tr>
<tr>
<td>RGB</td>
<td>Red Green Blue (connector for color monitor)</td>
</tr>
<tr>
<td>RH</td>
<td>Request/Response Header (SNA)</td>
</tr>
<tr>
<td>RJ-11C</td>
<td>Standard for common telephone jack</td>
</tr>
<tr>
<td>RLSD</td>
<td>Received Line Signal Detect (RS-232/V.24 signal); same as CD: Carrier Detect</td>
</tr>
<tr>
<td>RNR</td>
<td>Receive Not-Ready (SDLC, LAPD, X.25 Layer 2, X.25 Layer 3)</td>
</tr>
<tr>
<td>ROM</td>
<td>Read-Only Memory (firmware/software storage)</td>
</tr>
<tr>
<td>RR</td>
<td>Receive Ready (SDLC, LAPD, X.25 Layer 2, X.25 Layer 3)</td>
</tr>
<tr>
<td>RS</td>
<td>Record Separator</td>
</tr>
<tr>
<td>RS-232/V.24</td>
<td>List of definitions for interchange circuit between data terminal equipment and data circuit termination equipment established by EIA</td>
</tr>
<tr>
<td>RS-449</td>
<td>EIA standard for 37-pin and 9-pin DTE-DCE interface</td>
</tr>
<tr>
<td>RTI</td>
<td>Response Type Indicator (SNA)</td>
</tr>
<tr>
<td>RTS</td>
<td>Request To Send (RS-232/V.24 and V.35 signal)</td>
</tr>
<tr>
<td>RU</td>
<td>Request/Response Unit (SNA)</td>
</tr>
<tr>
<td>SABM</td>
<td>Set Asynchronous Balanced Mode (LAPD, X.25 Layer 2)</td>
</tr>
<tr>
<td>SABME</td>
<td>Set Asynchronous Balanced Mode Extended (LAPD, X.25 Layer 2)</td>
</tr>
<tr>
<td>SAPI</td>
<td>Service Access Point Identifier (LAPD)</td>
</tr>
<tr>
<td>SB</td>
<td>SUB: Substitute</td>
</tr>
<tr>
<td>SC</td>
<td>Session Control (SNA)</td>
</tr>
<tr>
<td>SCCP</td>
<td>Signalling Connection Control Part (SS#7 Layer 3)</td>
</tr>
<tr>
<td>SCR</td>
<td>Signal Clock Receive (RS-232/V.24 and V.35 signal), used when DCE clock drives DTE</td>
</tr>
<tr>
<td>SCT</td>
<td>Signal Clock Transmit (RS-232/V.24 and V.35 signal), used when DCE clock drives DTE</td>
</tr>
<tr>
<td>SCTE</td>
<td>Signal Clock Transmit External (RS-232/V.24 and V.35 signal), used when DTE clock drives DCE</td>
</tr>
<tr>
<td>SDI</td>
<td>Sense Data Indicator (SNA)</td>
</tr>
<tr>
<td>SDLC</td>
<td>Synchronous Data Link Control (IBM)</td>
</tr>
<tr>
<td>SDU</td>
<td>Service Data Unit</td>
</tr>
<tr>
<td>SELECTRIC</td>
<td>IBM typewriter/printer code</td>
</tr>
<tr>
<td>SFO</td>
<td>Status Field Octet (SS#7 Layer 2)</td>
</tr>
<tr>
<td>SH</td>
<td>SOH: Start of Header</td>
</tr>
<tr>
<td>SI</td>
<td>Shift In</td>
</tr>
<tr>
<td>S10</td>
<td>Sequenced Information Frame 0 (LAPD)</td>
</tr>
<tr>
<td>S11</td>
<td>Sequenced Information Frame 1 (LAPD)</td>
</tr>
<tr>
<td>SIG</td>
<td>Signal</td>
</tr>
<tr>
<td>SIO</td>
<td>Service Information Octet (SS#7 Layer 3)</td>
</tr>
<tr>
<td>SLS</td>
<td>Signalling Link Selection (SS#7 Layer 3)</td>
</tr>
<tr>
<td>SLU</td>
<td>Secondary Logical Unit (SNA)</td>
</tr>
</tbody>
</table>
SMP  Sample
SNA  System Network Architecture (IBM)
SNAI SNA Indicator (SNA)
SNF  Sequence Number Field (SNA)
SNRM Set Normal Response Mode (SDLC)
SO   Shift Out
SOH  Start of Header
SRC  Source
SREJ Selective Reject (SDLC)
SSCP System Services Control Point (SNA)
SS#7 CCSS#7: Common Channel Signaling System #7
struct Structure, data type which consists of a group of variables referenced under the same name, C language
STR  String
STX  Start of Text
SUB  Substitute
SX   STX: Start of Text
SY   SYN: Synchronization character
SYN  Synchronization character
SYS  System file
sys  System directory
T    Transmit (X.21 signal)
T    Transport layer (OSI primitive)
TD   Transmitted Data (RS-232/V.24 and V.35 signal)
TE   Terminal Equipment (ISDN)
TEI  Terminal Endpoint Identifier (LAPD)
TGNFI Transmission Group Non-Fifo Indicator (SNA)
TGSI Transmission Group Sweep Indicator (SNA)
TGSNF Transmission Group Sequence Number Field (SNA)
TH   Transmission Header
TIM  Test Interface Module
TPF  Transmission Priority Field (SNA)
TS   Transmission Services (SNA)
TTL  Transistor-to-Transistor Logic
TUP  Telephone User Part (SS#7 Layer 3)
typedef Type definition, data type which creates new name for existing data type, C language
UA   Unnumbered Acknowledgment (SDLC, LAPD, X.25 Layer 2)
UI   Unnumbered Information (SDLC)
UL   Underwriters' Laboratory
#undef Undefined, preprocessor directive, C language
US   Unit Separator
usec Microsecond
<table>
<thead>
<tr>
<th>USER TR</th>
<th>Run-mode User Trace screen</th>
</tr>
</thead>
<tbody>
<tr>
<td>usr</td>
<td>User directory</td>
</tr>
<tr>
<td>VRCWI</td>
<td>Virtual Route Change Window Indicator (SNA)</td>
</tr>
<tr>
<td>VRCWRI</td>
<td>Virtual Route Change Window Reply Indicator (SNA)</td>
</tr>
<tr>
<td>VRID</td>
<td>Virtual Route Identifier (SNA)</td>
</tr>
<tr>
<td>VRN</td>
<td>Virtual Route Number (SNA)</td>
</tr>
<tr>
<td>VRPCI</td>
<td>Virtual Route Pacing Count Indicator (SNA)</td>
</tr>
<tr>
<td>VRPRQ</td>
<td>Virtual Route Pacing Request (SNA)</td>
</tr>
<tr>
<td>VRPRS</td>
<td>Virtual Route Pacing Response (SNA)</td>
</tr>
<tr>
<td>VRRWI</td>
<td>Virtual Route Reset Window Indicator (SNA)</td>
</tr>
<tr>
<td>VRSI</td>
<td>Virtual Route Support Indicator (SNA)</td>
</tr>
<tr>
<td>VRSSN</td>
<td>Virtual Route Send Sequence Number (SNA)</td>
</tr>
<tr>
<td>VRSTI</td>
<td>Virtual Route Sequence and Type Indicator (SNA)</td>
</tr>
<tr>
<td>VT</td>
<td>Vertical Tab</td>
</tr>
<tr>
<td>X.21</td>
<td>CCITT recommendation governing synchronous DTE-DCE operation on public data networks</td>
</tr>
<tr>
<td>X.25</td>
<td>CCITT recommendation governing the packet mode link connecting the user site with a public data network</td>
</tr>
<tr>
<td>XDRAM</td>
<td>Extended Dynamic Random Access Memory</td>
</tr>
<tr>
<td>XEQ</td>
<td>Execute</td>
</tr>
<tr>
<td>XFER</td>
<td>Transfer</td>
</tr>
<tr>
<td>XID</td>
<td>Exchange Identification (SDLC)</td>
</tr>
<tr>
<td>XMIT</td>
<td>Transmit, transmission</td>
</tr>
</tbody>
</table>
Appendix C: Selectable Data Speeds

There are four clock crystals installed in the INTERVIEW. These clocks provide the bits-per-second rates listed on the following pages. (An optional crystal is also available that may be factory-configured for speeds not listed here.)

These baud rates apply to all clock and data format selections, with one exception. If you are operating in Emulate DCE mode using internal clock and the data format is anything other than Async, you may enter clock speeds 16 times higher than those listed. The following formula allows you to determine whether a higher data speed is selectable in this special case.

The frequency of each standard clock crystal is first divided by 2 to derive four values of $X$:

$$X_1 = \frac{3686400}{2}$$
$$X_2 = \frac{4096000}{2}$$
$$X_3 = \frac{4608000}{2}$$
$$X_4 = \frac{5376000}{2}$$

Divide the desired bps rate into each of the values of $X$ to produce result $Y$.

$$Y_1 = \frac{X_1}{\text{bps}}$$
$$Y_2 = \frac{X_2}{\text{bps}}$$
$$Y_3 = \frac{X_3}{\text{bps}}$$
$$Y_4 = \frac{X_4}{\text{bps}}$$

Round each of the $Y$ values to the nearest whole number.

Next, divide each $Y$ value into the corresponding $X$ value, to produce four possible speeds:

$$X_1/Y_1 = \text{SPEED1}$$
$$X_2/Y_2 = \text{SPEED2}$$
$$X_3/Y_3 = \text{SPEED3}$$
$$X_4/Y_4 = \text{SPEED4}$$

The values resulting from this calculation are the data speeds which may be selected for the desired bits-per-second rate. Select the closest speed and use this as your entry on the Line Setup Screen as the Internal Clock speed.
<table>
<thead>
<tr>
<th>Line Setup</th>
<th>Clock</th>
<th>Speeds</th>
</tr>
</thead>
<tbody>
<tr>
<td>168000.0</td>
<td>12923.08</td>
<td>6582.9</td>
</tr>
<tr>
<td>144000.0</td>
<td>12800.0</td>
<td>6545.45</td>
</tr>
<tr>
<td>128000.0</td>
<td>12126.3</td>
<td>6461.54</td>
</tr>
<tr>
<td>115200.0</td>
<td>12000.0</td>
<td>6400.0</td>
</tr>
<tr>
<td>84000.0</td>
<td>11636.36</td>
<td>6260.87</td>
</tr>
<tr>
<td>76800.0</td>
<td>11520.0</td>
<td>6227.0</td>
</tr>
<tr>
<td>72000.0</td>
<td>11200.0</td>
<td>6222.22</td>
</tr>
<tr>
<td>64000.0</td>
<td>11076.92</td>
<td>6095.24</td>
</tr>
<tr>
<td>57600.0</td>
<td>10971.4</td>
<td>6063.16</td>
</tr>
<tr>
<td>56000.0</td>
<td>10666.67</td>
<td>6000.0</td>
</tr>
<tr>
<td>48000.0</td>
<td>10500.0</td>
<td>5907.7</td>
</tr>
<tr>
<td>46080.0</td>
<td>10472.73</td>
<td>5818.18</td>
</tr>
<tr>
<td>42666.67</td>
<td>10285.71</td>
<td>5793.10</td>
</tr>
<tr>
<td>42000.0</td>
<td>10017.4</td>
<td>5600.0</td>
</tr>
<tr>
<td>38400.0</td>
<td>9882.35</td>
<td>5619.5</td>
</tr>
<tr>
<td>36000.0</td>
<td>9846.15</td>
<td>5600.0</td>
</tr>
<tr>
<td>33600.0</td>
<td>9600.0</td>
<td>5565.22</td>
</tr>
<tr>
<td>32914.3</td>
<td>9333.33</td>
<td>5538.46</td>
</tr>
<tr>
<td>32000.0</td>
<td>9216.0</td>
<td>5485.71</td>
</tr>
<tr>
<td>28800.0</td>
<td>9142.86</td>
<td>5419.35</td>
</tr>
<tr>
<td>28000.0</td>
<td>9000.0</td>
<td>5358.1</td>
</tr>
<tr>
<td>25600.0</td>
<td>8861.54</td>
<td>5333.33</td>
</tr>
<tr>
<td>24000.0</td>
<td>8842.11</td>
<td>5250.0</td>
</tr>
<tr>
<td>23040.0</td>
<td>8533.33</td>
<td>5236.36</td>
</tr>
<tr>
<td>21333.33</td>
<td>8470.59</td>
<td>5142.86</td>
</tr>
<tr>
<td>21000.0</td>
<td>8400.0</td>
<td>5120.0</td>
</tr>
<tr>
<td>20945.5</td>
<td>8228.57</td>
<td>5090.91</td>
</tr>
<tr>
<td>20571.43</td>
<td>8000.0</td>
<td>5008.61</td>
</tr>
<tr>
<td>19200.0</td>
<td>7944.8</td>
<td>4965.52</td>
</tr>
<tr>
<td>18666.67</td>
<td>7680.0</td>
<td>4941.18</td>
</tr>
<tr>
<td>18285.71</td>
<td>7636.36</td>
<td>4923.08</td>
</tr>
<tr>
<td>18000.0</td>
<td>7578.95</td>
<td>4902.1</td>
</tr>
<tr>
<td>17723.1</td>
<td>7529.41</td>
<td>4800.0</td>
</tr>
<tr>
<td>16800.0</td>
<td>7432.3</td>
<td>4740.74</td>
</tr>
<tr>
<td>16457.14</td>
<td>7304.35</td>
<td>4702.0</td>
</tr>
<tr>
<td>16000.0</td>
<td>7200.0</td>
<td>4666.67</td>
</tr>
<tr>
<td>15360.0</td>
<td>7111.11</td>
<td>4645.16</td>
</tr>
<tr>
<td>15272.73</td>
<td>7000.0</td>
<td>4608.0</td>
</tr>
<tr>
<td>14400.0</td>
<td>6981.8</td>
<td>4571.43</td>
</tr>
<tr>
<td>14222.22</td>
<td>6857.14</td>
<td>4540.54</td>
</tr>
<tr>
<td>14000.0</td>
<td>6776.47</td>
<td>4517.6</td>
</tr>
<tr>
<td>13552.9</td>
<td>6736.84</td>
<td>4500.0</td>
</tr>
<tr>
<td>13090.91</td>
<td>6720.0</td>
<td>4430.77</td>
</tr>
</tbody>
</table>
### Table C-1 (continued)

| 1888.52 | 1655.17 | 1473.68 | 1331.8 | 1210.08 | 1112.58 | 1028.57 |
| 1887.64 | 1647.06 | 1471.26 | 1324.14 | 1208.63 | 1107.69 | 1024.39 |
| 1882.35 | 1645.71 | 1469.39 | 1322.83 | 1207.55 | 1105.26 | 1024.0 |
| 1873.2  | 1641.03 | 1467.5  | 1321.10 | 1206.3  | 1103.45 | 1021.28 |
| 1870.13 | 1636.36 | 1460.87 | 1319.59 | 1200.0  | 1102.4  | 1019.47 |
| 1866.67 | 1634.0  | 1458.23 | 1316.6  | 1196.26 | 1099.24 | 1018.18 |
| 1858.06 | 1631.07 | 1454.55 | 1312.30 | 1193.8  | 1098.04 | 1015.87 |
| 1855.07 | 1622.54 | 1449.1  | 1309.09 | 1191.49 | 1097.14 | 1015.0 |
| 1846.15 | 1620.25 | 1448.28 | 1306.12 | 1190.08 | 1094.02 | 1014.08 |
| 1843.2  | 1617.98 | 1440.0  | 1302.33 | 1187.63 | 1091.9  | 1012.05 |
| 1828.57 | 1615.38 | 1438.20 | 1301.7  | 1185.19 | 1090.91 | 1010.53 |
| 1826.09 | 1611.2  | 1435.810| 1297.210| 1183.010| 1086.79 | 1007.87 |
| 1822.78 | 1600.0  | 1431.1  | 1294.38 | 1181.5  | 1084.75 | 1006.99 |
| 1814.2  | 1589.0  | 1425.74 | 1292.93 | 1180.33 | 1083.87 | 1006.1 |
| 1806.45 | 1584.91 | 1423.73 | 1292.31 | 1175.51 | 1082.71 | 1005.99 |
| 1802.82 | 1582.42 | 1422.22 | 1287.2  | 1174.83 | 1081.7  | 1001.74 |
| 1800.0  | 1580.25 | 1413.5  | 1285.71 | 1174.31 | 1076.92 | 1000.0 |
| 1787.23 | 1578.08 | 1411.76 | 1282.44 | 1170.73 | 1076.64 | 997.4 |
| 1786.0  | 1570.09 | 1406.59 | 1280.0  | 1169.5  | 1075.63 | 994.08 |
| 1777.78 | 1567.3  | 1404.88 | 1274.34 | 1166.67 | 1074.63 | 993.10 |
| 1772.31 | 1565.22 | 1400.0  | 1272.9  | 1163.64 | 1071.6  | 992.25 |
| 1768.42 | 1560.98 | 1398.06 | 1272.73 | 1161.29 | 1070.06 | 988.8 |
| 1758.8  | 1556.76 | 1396.0  | 1267.33 | 1158.62 | 1066.67 | 988.24 |
| 1756.010| 1555.56 | 1391.30 | 1265.93 | 1157.8  | 1063.29 | 986.30 |
| 1753.42 | 1548.39 | 1388.43 | 1263.16 | 1153.15 | 1061.8  | 984.62 |
| 1750.0  | 1546.3  | 1388.0  | 1259.0  | 1152.0  | 1058.82 | 982.46 |
| 1745.45 | 1542.17 | 1387.95 | 1254.90 | 1150.68 | 1057.85 | 980.4 |
| 1734.94 | 1541.28 | 1384.62 | 1253.73 | 1146.3  | 1056.88 | 979.59 |
| 1732.3  | 1536.0  | 1379.6  | 1252.17 | 1142.86 | 1056.60 | 977.010|
| 1731.96 | 1531.91 | 1377.05 | 1245.4  | 1140.59 | 1052.1  | 976.74 |
| 1729.73 | 1527.27 | 1376.34 | 1244.44 | 1135.14 | 1051.09 | 976.27 |
| 1719.40 | 1525.8  | 1371.43 | 1242.72 | 1135.0  | 1050.0  | 972.97 |
| 1714.29 | 1523.81 | 1365.85 | 1241.38 | 1133.86 | 1049.18 | 972.2 |
| 1706.67 | 1515.79 | 1363.3  | 1238.71 | 1132.74 | 1047.27 | 971.010|
| 1696.97 | 1513.51 | 1361.70 | 1235.29 | 1129.41 | 1043.48 | 969.610|
| 1694.12 | 1505.88 | 1358.49 | 1232.1  | 1127.52 | 1042.5  | 968.07 |
| 1684.21 | 1500.0  | 1355.29 | 1230.77 | 1125.0  | 1040.65 | 966.44 |
| 1681.8  | 1496.10 | 1354.84 | 1226.28 | 1123.9  | 1037.84 | 965.52 |
| 1680.0  | 1488.37 | 1347.37 | 1225.53 | 1122.81 | 1037.04 | 964.0 |
| 1674.42 | 1486.73 | 1345.79 | 1220.34 | 1120.0  | 1035.97 | 962.41 |
| 1669.57 | 1486.5  | 1344.0  | 1219.05 | 1118.45 | 1033.2  | 960.0 |
| 1663.37 | 1484.54 | 1339.53 | 1217.39 | 1116.28 | 1032.26 | 956.0 |
| 1662.34 | 1476.92 | 1333.33 | 1212.63 | 1113.04 | 1030.67 | 955.22 |
### Table C-1 (continued)

| 954.55 | 893.02 | 835.82 | 786.89 | 743.23 | 702.93 | 669.32 |
| 953.64 | 889.6  | 834.78 | 786.3  | 742.27 | 702.44 | 667.8 |
| 952.07 | 888.89 | 832.37 | 785.28 | 740.8  | 700.3  | 666.67 |
| 949.15 | 886.15 | 831.8  | 785.05 | 740.09 | 700.0  | 665.9 |
| 948.15 | 884.21 | 831.68 | 783.67 | 739.88 | 699.45 | 665.810 |
| 947.37 | 883.44 | 831.17 | 782.61 | 738.46 | 699.03 | 664.03 |
| 944.26 | 882.76 | 828.78 | 781.310 | 736.84 | 698.18 | 663.59 |
| 943.82 | 879.58 | 827.59 | 781.0  | 736.1  | 697.010 | 663.21 |
| 941.18 | 879.39 | 825.81 | 780.49 | 735.63 | 696.1  | 662.07 |
| 940.4  | 878.05 | 823.53 | 778.38 | 734.69 | 695.65 | 661.42 |
| 938.55 | 876.71 | 822.86 | 777.78 | 733.76 | 694.21 | 660.55 |
| 936.59 | 876.0  | 820.51 | 775.76 | 733.62 | 694.0  | 660.2 |
| 935.06 | 875.0  | 819.9  | 774.19 | 731.43 | 693.98 | 659.79 |
| 934.31 | 872.73 | 819.51 | 773.15 | 730.96 | 692.31 | 658.82 |
| 933.33 | 870.75 | 818.18 | 771.08 | 730.43 | 691.89 | 658.29 |
| 932.8  | 870.47 | 817.02 | 770.64 | 729.11 | 691.36 | 657.53 |
| 929.03 | 869.4  | 815.53 | 770.05 | 727.27 | 689.82 | 656.41 |
| 928.18 | 867.47 | 815.29 | 768.0  | 726.8  | 688.910 | 656.25 |
| 927.54 | 866.17 | 814.1  | 767.12 | 724.53 | 688.52 | 654.55 |
| 925.3  | 865.98 | 813.56 | 766.47 | 724.14 | 688.17 | 653.610 |
| 923.08 | 864.86 | 811.59 | 765.96 | 723.62 | 687.8  | 653.06 |
| 921.60 | 862.9  | 811.27 | 765.4  | 723.16 | 685.71 | 652.7 |
| 920.86 | 862.28 | 810.13 | 763.64 | 722.3  | 684.49 | 651.58 |
| 918.03 | 861.54 | 808.99 | 762.91 | 721.03 | 683.7  | 651.16 |
| 917.9  | 859.70 | 808.4  | 761.90 | 720.0  | 682.93 | 650.85 |
| 917.110 | 859.06 | 807.69 | 760.4  | 719.10 | 682.46 | 649.75 |
| 914.29 | 857.14 | 805.59 | 760.18 | 717.95 | 681.66 | 649.0 |
| 913.04 | 856.5  | 805.03 | 757.89 | 717.8  | 680.85 | 648.65 |
| 911.39 | 853.33 | 804.47 | 757.310 | 716.42 | 680.16 | 647.19 |
| 910.7  | 852.79 | 803.83 | 756.76 | 715.53 | 679.6  | 646.46 |
| 908.11 | 852.07 | 802.8  | 755.4  | 715.08 | 679.25 | 646.15 |
| 907.80 | 850.2  | 800.0  | 753.93 | 714.89 | 677.65 | 645.74 |
| 907.09 | 848.48 | 797.2  | 753.36 | 713.3  | 677.42 | 645.4 |
| 905.66 | 847.68 | 796.21 | 752.94 | 712.87 | 677.25 | 643.68 |
| 903.5  | 847.06 | 795.58 | 750.5  | 711.86 | 676.06 | 643.58 |
| 903.23 | 844.22 | 795.03 | 750.0  | 711.11 | 675.7  | 643.22 |
| 901.41 | 844.0  | 794.48 | 748.54 | 709.36 | 674.610 | 642.86 |
| 900.0  | 842.11 | 792.43 | 748.05 | 708.86 | 673.68 | 641.8 |
| 898.310 | 840.88 | 791.8  | 746.67 | 707.18 | 672.810 | 641.22 |
| 896.5  | 840.0  | 791.21 | 746.11 | 706.75 | 672.0  | 640.0 |
| 895.10 | 837.8  | 790.12 | 745.6  | 705.88 | 671.7  | 638.78 |
| 894.41 | 837.21 | 789.04 | 744.19 | 704.6  | 670.16 | 638.2 |
| 893.62 | 836.60 | 788.73 | 743.36 | 703.210 | 669.77 | 637.17 |
### Table C-1 (continued)

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>636.82</td>
<td>608.610</td>
<td>581.82</td>
<td>559.22</td>
<td>538.32</td>
<td>517.8</td>
<td>498.52</td>
</tr>
<tr>
<td>636.46</td>
<td>607.9</td>
<td>581.31</td>
<td>558.95</td>
<td>537.82</td>
<td>516.92</td>
<td>498.27</td>
</tr>
<tr>
<td>636.36</td>
<td>607.59</td>
<td>580.65</td>
<td>558.14</td>
<td>537.31</td>
<td>516.59</td>
<td>498.05</td>
</tr>
<tr>
<td>634.7</td>
<td>606.64</td>
<td>580.4</td>
<td>557.9</td>
<td>537.1</td>
<td>516.13</td>
<td>497.6</td>
</tr>
<tr>
<td>634.36</td>
<td>606.410</td>
<td>579.31</td>
<td>556.52</td>
<td>536.74</td>
<td>515.4</td>
<td>497.04</td>
</tr>
<tr>
<td>633.96</td>
<td>606.32</td>
<td>579.19</td>
<td>556.29</td>
<td>535.81</td>
<td>515.34</td>
<td>496.55</td>
</tr>
<tr>
<td>633.66</td>
<td>605.04</td>
<td>578.89</td>
<td>555.98</td>
<td>535.56</td>
<td>514.29</td>
<td>496.6</td>
</tr>
<tr>
<td>633.15</td>
<td>604.31</td>
<td>578.31</td>
<td>555.2</td>
<td>535.32</td>
<td>514.06</td>
<td>496.12</td>
</tr>
<tr>
<td>632.47</td>
<td>604.32</td>
<td>577.4</td>
<td>554.46</td>
<td>534.4</td>
<td>513.76</td>
<td>495.58</td>
</tr>
<tr>
<td>631.58</td>
<td>603.77</td>
<td>577.32</td>
<td>554.11</td>
<td>534.6</td>
<td>513.1</td>
<td>495.5</td>
</tr>
<tr>
<td>631.12</td>
<td>603.14</td>
<td>576.58</td>
<td>553.85</td>
<td>533.33</td>
<td>512.46</td>
<td>494.85</td>
</tr>
<tr>
<td>630.54</td>
<td>602.51</td>
<td>576.0</td>
<td>552.63</td>
<td>532.1</td>
<td>512.110</td>
<td>494.42</td>
</tr>
<tr>
<td>629.51</td>
<td>602.15</td>
<td>575.34</td>
<td>552.5</td>
<td>531.65</td>
<td>512.0</td>
<td>494.21</td>
</tr>
<tr>
<td>629.21</td>
<td>601.6</td>
<td>574.6</td>
<td>554.72</td>
<td>531.37</td>
<td>510.9</td>
<td>494.12</td>
</tr>
<tr>
<td>628.82</td>
<td>600.94</td>
<td>573.99</td>
<td>551.2</td>
<td>531.12</td>
<td>510.64</td>
<td>494.3</td>
</tr>
<tr>
<td>627.8</td>
<td>600.0</td>
<td>573.71</td>
<td>551.110</td>
<td>530.88</td>
<td>509.96</td>
<td>493.15</td>
</tr>
<tr>
<td>627.45</td>
<td>598.4</td>
<td>573.38</td>
<td>550.82</td>
<td>529.97</td>
<td>509.73</td>
<td>492.67</td>
</tr>
<tr>
<td>626.87</td>
<td>598.13</td>
<td>573.13</td>
<td>549.9</td>
<td>529.7</td>
<td>509.09</td>
<td>492.31</td>
</tr>
<tr>
<td>626.09</td>
<td>597.86</td>
<td>571.7</td>
<td>549.62</td>
<td>529.41</td>
<td>508.83</td>
<td>491.47</td>
</tr>
<tr>
<td>624.54</td>
<td>597.51</td>
<td>571.43</td>
<td>549.36</td>
<td>528.93</td>
<td>508.6</td>
<td>491.3</td>
</tr>
<tr>
<td>624.39</td>
<td>596.89</td>
<td>570.3</td>
<td>549.02</td>
<td>528.44</td>
<td>507.94</td>
<td>491.23</td>
</tr>
<tr>
<td>623.38</td>
<td>595.74</td>
<td>570.210</td>
<td>548.57</td>
<td>528.30</td>
<td>507.55</td>
<td>490.42</td>
</tr>
<tr>
<td>622.70</td>
<td>595.35</td>
<td>569.49</td>
<td>547.53</td>
<td>527.47</td>
<td>507.49</td>
<td>490.21</td>
</tr>
<tr>
<td>622.22</td>
<td>595.04</td>
<td>569.17</td>
<td>547.3</td>
<td>527.2</td>
<td>507.04</td>
<td>489.710</td>
</tr>
<tr>
<td>621.36</td>
<td>593.81</td>
<td>568.89</td>
<td>547.23</td>
<td>526.75</td>
<td>506.4</td>
<td>489.2</td>
</tr>
<tr>
<td>621.0</td>
<td>593.64</td>
<td>567.57</td>
<td>547.01</td>
<td>526.65</td>
<td>506.02</td>
<td>488.55</td>
</tr>
<tr>
<td>620.69</td>
<td>592.59</td>
<td>567.49</td>
<td>546.0</td>
<td>526.03</td>
<td>505.93</td>
<td>488.37</td>
</tr>
<tr>
<td>619.93</td>
<td>592.3</td>
<td>566.93</td>
<td>545.97</td>
<td>525.55</td>
<td>505.26</td>
<td>488.14</td>
</tr>
<tr>
<td>619.35</td>
<td>591.55</td>
<td>566.37</td>
<td>545.45</td>
<td>525.0</td>
<td>504.50</td>
<td>486.96</td>
</tr>
<tr>
<td>617.7</td>
<td>590.77</td>
<td>566.1</td>
<td>544.68</td>
<td>524.8</td>
<td>504.2</td>
<td>486.69</td>
</tr>
<tr>
<td>618.36</td>
<td>590.16</td>
<td>565.66</td>
<td>543.69</td>
<td>524.59</td>
<td>503.94</td>
<td>486.49</td>
</tr>
<tr>
<td>618.03</td>
<td>589.86</td>
<td>564.71</td>
<td>543.4</td>
<td>523.64</td>
<td>503.410</td>
<td>486.08</td>
</tr>
<tr>
<td>617.65</td>
<td>589.47</td>
<td>563.88</td>
<td>543.310</td>
<td>523.36</td>
<td>503.1</td>
<td>485.55</td>
</tr>
<tr>
<td>616.04</td>
<td>589.3</td>
<td>563.76</td>
<td>542.37</td>
<td>522.45</td>
<td>503.06</td>
<td>484.85</td>
</tr>
<tr>
<td>615.38</td>
<td>587.76</td>
<td>563.3</td>
<td>542.1</td>
<td>521.74</td>
<td>502.99</td>
<td>484.15</td>
</tr>
<tr>
<td>614.4</td>
<td>587.41</td>
<td>562.5</td>
<td>541.94</td>
<td>521.27</td>
<td>502.0</td>
<td>484.03</td>
</tr>
<tr>
<td>613.14</td>
<td>587.16</td>
<td>562.0</td>
<td>541.35</td>
<td>520.33</td>
<td>501.96</td>
<td>483.22</td>
</tr>
<tr>
<td>612.77</td>
<td>586.3</td>
<td>561.95</td>
<td>540.85</td>
<td>520.12</td>
<td>501.74</td>
<td>483.02</td>
</tr>
<tr>
<td>612.44</td>
<td>585.37</td>
<td>561.87</td>
<td>540.19</td>
<td>519.86</td>
<td>501.49</td>
<td>482.76</td>
</tr>
<tr>
<td>611.1</td>
<td>584.77</td>
<td>561.40</td>
<td>540.08</td>
<td>518.92</td>
<td>500.87</td>
<td>482.01</td>
</tr>
<tr>
<td>610.91</td>
<td>584.47</td>
<td>560.31</td>
<td>539.6</td>
<td>518.52</td>
<td>500.0</td>
<td>481.61</td>
</tr>
<tr>
<td>610.17</td>
<td>583.33</td>
<td>560.6</td>
<td>539.33</td>
<td>518.22</td>
<td>499.8</td>
<td>481.38</td>
</tr>
<tr>
<td>609.52</td>
<td>582.910</td>
<td>560.0</td>
<td>538.46</td>
<td>517.99</td>
<td>498.70</td>
<td>481.20</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td></td>
<td></td>
</tr>
<tr>
<td>481.0</td>
<td>464.09</td>
<td>449.12</td>
<td>435.23</td>
<td>422.29</td>
<td>407.93</td>
<td>393.85</td>
</tr>
<tr>
<td>480.0</td>
<td>463.77</td>
<td>449.110</td>
<td>435.05</td>
<td>422.11</td>
<td>407.77</td>
<td>393.44</td>
</tr>
<tr>
<td>479.40</td>
<td>463.6</td>
<td>448.510</td>
<td>434.72</td>
<td>422.0</td>
<td>407.64</td>
<td>393.17</td>
</tr>
<tr>
<td>479.0</td>
<td>463.02</td>
<td>448.25</td>
<td>434.11</td>
<td>421.98</td>
<td>407.07</td>
<td>392.64</td>
</tr>
<tr>
<td>478.63</td>
<td>462.81</td>
<td>448.2</td>
<td>433.9</td>
<td>421.2</td>
<td>406.78</td>
<td>392.52</td>
</tr>
<tr>
<td>478.41</td>
<td>462.7</td>
<td>448.0</td>
<td>433.810</td>
<td>421.05</td>
<td>406.35</td>
<td>392.37</td>
</tr>
<tr>
<td>478.01</td>
<td>462.65</td>
<td>447.55</td>
<td>433.73</td>
<td>420.44</td>
<td>405.710</td>
<td>391.84</td>
</tr>
<tr>
<td>477.61</td>
<td>462.09</td>
<td>447.4</td>
<td>433.08</td>
<td>420.0</td>
<td>405.63</td>
<td>391.61</td>
</tr>
<tr>
<td>477.27</td>
<td>461.7</td>
<td>447.20</td>
<td>432.99</td>
<td>419.83</td>
<td>405.06</td>
<td>391.44</td>
</tr>
<tr>
<td>477.0</td>
<td>461.54</td>
<td>446.81</td>
<td>432.43</td>
<td>419.67</td>
<td>404.82</td>
<td>391.30</td>
</tr>
<tr>
<td>476.82</td>
<td>460.80</td>
<td>446.51</td>
<td>432.3</td>
<td>418.95</td>
<td>404.49</td>
<td>390.610</td>
</tr>
<tr>
<td>476.03</td>
<td>460.43</td>
<td>445.99</td>
<td>431.88</td>
<td>418.91</td>
<td>404.21</td>
<td>390.51</td>
</tr>
<tr>
<td>475.92</td>
<td>460.27</td>
<td>445.82</td>
<td>431.46</td>
<td>418.9</td>
<td>403.85</td>
<td>390.24</td>
</tr>
<tr>
<td>475.84</td>
<td>460.06</td>
<td>445.62</td>
<td>431.14</td>
<td>418.6</td>
<td>403.79</td>
<td>389.79</td>
</tr>
<tr>
<td>475.25</td>
<td>459.9</td>
<td>444.79</td>
<td>430.98</td>
<td>418.3</td>
<td>403.36</td>
<td>389.19</td>
</tr>
<tr>
<td>475.1</td>
<td>459.02</td>
<td>444.44</td>
<td>430.77</td>
<td>418.1</td>
<td>402.88</td>
<td>389.06</td>
</tr>
<tr>
<td>474.58</td>
<td>458.96</td>
<td>443.9</td>
<td>430.7</td>
<td>417.91</td>
<td>402.710</td>
<td>388.9</td>
</tr>
<tr>
<td>474.07</td>
<td>458.78</td>
<td>443.27</td>
<td>429.9</td>
<td>417.39</td>
<td>402.52</td>
<td>388.14</td>
</tr>
<tr>
<td>473.68</td>
<td>458.510</td>
<td>443.08</td>
<td>429.85</td>
<td>416.94</td>
<td>402.23</td>
<td>387.99</td>
</tr>
<tr>
<td>473.24</td>
<td>458.1</td>
<td>442.91</td>
<td>429.67</td>
<td>416.87</td>
<td>401.91</td>
<td>387.88</td>
</tr>
<tr>
<td>473.1</td>
<td>457.77</td>
<td>442.2</td>
<td>429.53</td>
<td>416.18</td>
<td>401.39</td>
<td>387.010</td>
</tr>
<tr>
<td>472.32</td>
<td>457.14</td>
<td>442.11</td>
<td>429.1</td>
<td>415.88</td>
<td>401.25</td>
<td>386.71</td>
</tr>
<tr>
<td>472.13</td>
<td>456.52</td>
<td>441.72</td>
<td>428.57</td>
<td>415.84</td>
<td>401.11</td>
<td>386.58</td>
</tr>
<tr>
<td>471.91</td>
<td>456.2</td>
<td>441.38</td>
<td>428.3</td>
<td>415.58</td>
<td>400.95</td>
<td>386.21</td>
</tr>
<tr>
<td>471.2</td>
<td>455.610</td>
<td>440.94</td>
<td>428.25</td>
<td>414.99</td>
<td>400.0</td>
<td>386.06</td>
</tr>
<tr>
<td>470.59</td>
<td>455.52</td>
<td>440.5</td>
<td>428.09</td>
<td>414.81</td>
<td>399.05</td>
<td>385.54</td>
</tr>
<tr>
<td>470.20</td>
<td>455.34</td>
<td>440.37</td>
<td>427.48</td>
<td>414.39</td>
<td>398.89</td>
<td>385.32</td>
</tr>
<tr>
<td>469.27</td>
<td>455.28</td>
<td>439.86</td>
<td>427.210</td>
<td>414.24</td>
<td>398.75</td>
<td>385.28</td>
</tr>
<tr>
<td>469.2</td>
<td>454.4</td>
<td>439.79</td>
<td>426.67</td>
<td>413.79</td>
<td>398.62</td>
<td>385.03</td>
</tr>
<tr>
<td>469.06</td>
<td>454.26</td>
<td>439.69</td>
<td>426.310</td>
<td>412.90</td>
<td>398.10</td>
<td>384.44</td>
</tr>
<tr>
<td>468.86</td>
<td>454.05</td>
<td>439.02</td>
<td>426.04</td>
<td>412.78</td>
<td>397.79</td>
<td>384.38</td>
</tr>
<tr>
<td>468.29</td>
<td>453.90</td>
<td>438.9</td>
<td>425.9</td>
<td>412.61</td>
<td>397.52</td>
<td>384.0</td>
</tr>
<tr>
<td>467.97</td>
<td>453.54</td>
<td>438.64</td>
<td>425.32</td>
<td>411.76</td>
<td>397.24</td>
<td>383.56</td>
</tr>
<tr>
<td>467.53</td>
<td>452.83</td>
<td>438.36</td>
<td>425.25</td>
<td>411.58</td>
<td>397.16</td>
<td>383.23</td>
</tr>
<tr>
<td>467.15</td>
<td>452.7</td>
<td>438.02</td>
<td>425.09</td>
<td>411.43</td>
<td>396.69</td>
<td>382.98</td>
</tr>
<tr>
<td>466.67</td>
<td>452.210</td>
<td>438.0</td>
<td>424.78</td>
<td>410.76</td>
<td>396.28</td>
<td>382.72</td>
</tr>
<tr>
<td>466.4</td>
<td>451.76</td>
<td>437.69</td>
<td>424.3</td>
<td>410.26</td>
<td>396.23</td>
<td>382.69</td>
</tr>
<tr>
<td>466.310</td>
<td>451.61</td>
<td>437.5</td>
<td>424.24</td>
<td>409.96</td>
<td>395.88</td>
<td>382.09</td>
</tr>
<tr>
<td>466.02</td>
<td>451.41</td>
<td>437.2</td>
<td>423.84</td>
<td>409.76</td>
<td>395.60</td>
<td>381.96</td>
</tr>
<tr>
<td>465.5</td>
<td>450.9</td>
<td>436.86</td>
<td>423.53</td>
<td>409.09</td>
<td>395.29</td>
<td>381.82</td>
</tr>
<tr>
<td>465.45</td>
<td>450.70</td>
<td>436.36</td>
<td>423.17</td>
<td>408.95</td>
<td>395.06</td>
<td>381.46</td>
</tr>
<tr>
<td>465.37</td>
<td>450.40</td>
<td>435.5</td>
<td>422.8</td>
<td>408.76</td>
<td>394.52</td>
<td>380.95</td>
</tr>
<tr>
<td>464.52</td>
<td>450.0</td>
<td>435.37</td>
<td>422.44</td>
<td>408.51</td>
<td>394.37</td>
<td>380.110</td>
</tr>
</tbody>
</table>
### Table C-1 (continued)

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>370.04</td>
<td>357.76</td>
<td>346.88</td>
<td>336.0</td>
<td>325.79</td>
<td>316.98</td>
<td>308.02</td>
</tr>
<tr>
<td>369.94</td>
<td>357.54</td>
<td>346.39</td>
<td>335.96</td>
<td>325.610</td>
<td>316.83</td>
<td>307.69</td>
</tr>
<tr>
<td>369.23</td>
<td>357.45</td>
<td>346.15</td>
<td>335.86</td>
<td>325.58</td>
<td>316.48</td>
<td>307.20</td>
</tr>
<tr>
<td>368.88</td>
<td>357.32</td>
<td>345.95</td>
<td>335.66</td>
<td>325.42</td>
<td>316.38</td>
<td>307.13</td>
</tr>
<tr>
<td>368.42</td>
<td>356.69</td>
<td>345.68</td>
<td>335.33</td>
<td>325.06</td>
<td>316.05</td>
<td>307.04</td>
</tr>
<tr>
<td>368.29</td>
<td>356.66</td>
<td>345.32</td>
<td>335.08</td>
<td>324.95</td>
<td>315.79</td>
<td>306.95</td>
</tr>
<tr>
<td>368.05</td>
<td>356.55</td>
<td>345.01</td>
<td>334.88</td>
<td>324.87</td>
<td>315.62</td>
<td>306.57</td>
</tr>
<tr>
<td>367.82</td>
<td>356.44</td>
<td>344.97</td>
<td>334.66</td>
<td>324.51</td>
<td>315.27</td>
<td>306.38</td>
</tr>
</tbody>
</table>

---

Appendix C  Selectable Data Speeds
<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>306.22</td>
<td>303.710</td>
<td>301.18</td>
<td>150.0</td>
<td>100.0</td>
<td>45.0</td>
<td>16.0</td>
<td></td>
</tr>
<tr>
<td>306.01</td>
<td>303.32</td>
<td>301.08</td>
<td>144.0</td>
<td>96.0</td>
<td>42.0</td>
<td>15.0</td>
<td></td>
</tr>
<tr>
<td>305.73</td>
<td>303.25</td>
<td>300.0</td>
<td>140.0</td>
<td>90.0</td>
<td>40.0</td>
<td>14.0</td>
<td></td>
</tr>
<tr>
<td>305.57</td>
<td>303.16</td>
<td>288.0</td>
<td>134.5</td>
<td>84.0</td>
<td>36.0</td>
<td>12.0</td>
<td></td>
</tr>
<tr>
<td>305.49</td>
<td>302.70</td>
<td>280.0</td>
<td>128.0</td>
<td>80.0</td>
<td>35.0</td>
<td>10.0</td>
<td></td>
</tr>
<tr>
<td>305.45</td>
<td>302.60</td>
<td>256.0</td>
<td>125.0</td>
<td>75.0</td>
<td>32.0</td>
<td>9.0</td>
<td></td>
</tr>
<tr>
<td>305.08</td>
<td>302.52</td>
<td>210.0</td>
<td>120.0</td>
<td>72.0</td>
<td>30.0</td>
<td>8.0</td>
<td></td>
</tr>
<tr>
<td>304.90</td>
<td>302.36</td>
<td>200.0</td>
<td>250.0</td>
<td>70.0</td>
<td>28.0</td>
<td>7.0</td>
<td></td>
</tr>
<tr>
<td>304.76</td>
<td>302.16</td>
<td>192.0</td>
<td>240.0</td>
<td>64.0</td>
<td>25.0</td>
<td>6.0</td>
<td></td>
</tr>
<tr>
<td>304.44</td>
<td>301.89</td>
<td>180.0</td>
<td>225.0</td>
<td>60.0</td>
<td>24.0</td>
<td>5.0</td>
<td></td>
</tr>
<tr>
<td>304.35</td>
<td>301.62</td>
<td>175.0</td>
<td>224.0</td>
<td>60.0</td>
<td>21.0</td>
<td>4.0</td>
<td></td>
</tr>
<tr>
<td>304.04</td>
<td>301.57</td>
<td>168.0</td>
<td>112.0</td>
<td>50.0</td>
<td>20.0</td>
<td>3.0</td>
<td></td>
</tr>
<tr>
<td>303.96</td>
<td>301.26</td>
<td>160.0</td>
<td>105.0</td>
<td>48.0</td>
<td>18.0</td>
<td>2.0</td>
<td></td>
</tr>
</tbody>
</table>
Appendix D1: Keyboard-to-Hex Translation

<table>
<thead>
<tr>
<th>Letter</th>
<th>Decimal</th>
<th>Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>65</td>
<td>41</td>
</tr>
<tr>
<td>B</td>
<td>66</td>
<td>42</td>
</tr>
<tr>
<td>C</td>
<td>67</td>
<td>43</td>
</tr>
<tr>
<td>D</td>
<td>68</td>
<td>44</td>
</tr>
<tr>
<td>E</td>
<td>69</td>
<td>45</td>
</tr>
<tr>
<td>F</td>
<td>70</td>
<td>46</td>
</tr>
<tr>
<td>G</td>
<td>71</td>
<td>47</td>
</tr>
<tr>
<td>H</td>
<td>72</td>
<td>48</td>
</tr>
<tr>
<td>I</td>
<td>73</td>
<td>49</td>
</tr>
<tr>
<td>J</td>
<td>74</td>
<td>4A</td>
</tr>
<tr>
<td>K</td>
<td>75</td>
<td>4B</td>
</tr>
<tr>
<td>L</td>
<td>76</td>
<td>4C</td>
</tr>
<tr>
<td>M</td>
<td>77</td>
<td>4D</td>
</tr>
<tr>
<td>N</td>
<td>78</td>
<td>4E</td>
</tr>
<tr>
<td>O</td>
<td>79</td>
<td>4F</td>
</tr>
<tr>
<td>P</td>
<td>80</td>
<td>50</td>
</tr>
<tr>
<td>Q</td>
<td>81</td>
<td>51</td>
</tr>
<tr>
<td>R</td>
<td>82</td>
<td>52</td>
</tr>
<tr>
<td>S</td>
<td>83</td>
<td>53</td>
</tr>
<tr>
<td>T</td>
<td>84</td>
<td>54</td>
</tr>
<tr>
<td>U</td>
<td>85</td>
<td>55</td>
</tr>
<tr>
<td>V</td>
<td>86</td>
<td>56</td>
</tr>
<tr>
<td>W</td>
<td>87</td>
<td>57</td>
</tr>
<tr>
<td>X</td>
<td>88</td>
<td>58</td>
</tr>
<tr>
<td>Y</td>
<td>89</td>
<td>59</td>
</tr>
<tr>
<td>Z</td>
<td>90</td>
<td>41</td>
</tr>
</tbody>
</table>
## Table D1-1
### Keyboard-to-EBCDIC

<table>
<thead>
<tr>
<th>KEY</th>
<th>UNSHIFTED</th>
<th>CONTROL</th>
<th>SHIFTED</th>
<th>SHIFTED</th>
<th>UNSHIFTED</th>
<th>CONTROL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CHAR¹</td>
<td>HEX²</td>
<td>CHAR¹</td>
<td>HEX²</td>
<td>CHAR¹</td>
<td>HEX²</td>
</tr>
<tr>
<td>A</td>
<td>a 81</td>
<td>A C1</td>
<td>SOH 01</td>
<td>B 82</td>
<td>B C2</td>
<td>STX 02</td>
</tr>
<tr>
<td>B</td>
<td>b 83</td>
<td>B C3</td>
<td>EOT 37</td>
<td>C 84</td>
<td>C C4</td>
<td>ETX 03</td>
</tr>
<tr>
<td>C</td>
<td>c 85</td>
<td>C C5</td>
<td>ENQ 2D</td>
<td>D 86</td>
<td>D C6</td>
<td>ACK 2E</td>
</tr>
<tr>
<td>D</td>
<td>d 87</td>
<td>D C7</td>
<td>BEL 2F</td>
<td>E 88</td>
<td>E C8</td>
<td>BS 16</td>
</tr>
<tr>
<td>E</td>
<td>e 89</td>
<td>E C9</td>
<td>HT 05</td>
<td>F 90</td>
<td>F D0</td>
<td>FF 0C</td>
</tr>
<tr>
<td>F</td>
<td>f 91</td>
<td>F D1</td>
<td>LF 25</td>
<td>G 92</td>
<td>G D2</td>
<td>VT 0B</td>
</tr>
<tr>
<td>G</td>
<td>g 93</td>
<td>G D3</td>
<td>DC1 11</td>
<td>H 94</td>
<td>H D4</td>
<td>DC2 12</td>
</tr>
<tr>
<td>H</td>
<td>h 95</td>
<td>H D5</td>
<td>DC3 13</td>
<td>I 96</td>
<td>I D6</td>
<td>DC4 3C</td>
</tr>
<tr>
<td>I</td>
<td>i 97</td>
<td>I D7</td>
<td>SYN 32</td>
<td>J 98</td>
<td>J D8</td>
<td>DLE 10</td>
</tr>
<tr>
<td>J</td>
<td>j 99</td>
<td>J D9</td>
<td>DBC 11</td>
<td>K 100</td>
<td>K D10</td>
<td>DC2 12</td>
</tr>
<tr>
<td>K</td>
<td>k 101</td>
<td>K D11</td>
<td>DC3 13</td>
<td>L 102</td>
<td>L D12</td>
<td>DC4 3C</td>
</tr>
<tr>
<td>L</td>
<td>l 103</td>
<td>L D13</td>
<td>DBC 11</td>
<td>M 104</td>
<td>M D14</td>
<td>DC2 12</td>
</tr>
<tr>
<td>M</td>
<td>m 105</td>
<td>M D15</td>
<td>DC3 13</td>
<td>N 106</td>
<td>N D16</td>
<td>DC4 3C</td>
</tr>
<tr>
<td>N</td>
<td>n 107</td>
<td>N D17</td>
<td>DBC 11</td>
<td>O 108</td>
<td>O D18</td>
<td>DC2 12</td>
</tr>
<tr>
<td>O</td>
<td>o 109</td>
<td>O D19</td>
<td>DC3 13</td>
<td>P 110</td>
<td>P D20</td>
<td>DC4 3C</td>
</tr>
<tr>
<td>P</td>
<td>p 111</td>
<td>P D21</td>
<td>DBC 11</td>
<td>Q 112</td>
<td>Q D22</td>
<td>DC2 12</td>
</tr>
<tr>
<td>Q</td>
<td>q 113</td>
<td>Q D23</td>
<td>DC3 13</td>
<td>R 114</td>
<td>R D24</td>
<td>DC4 3C</td>
</tr>
<tr>
<td>R</td>
<td>r 115</td>
<td>R D25</td>
<td>DBC 11</td>
<td>S 116</td>
<td>S D26</td>
<td>DC2 12</td>
</tr>
<tr>
<td>S</td>
<td>s 117</td>
<td>S D27</td>
<td>DC3 13</td>
<td>T 118</td>
<td>T D28</td>
<td>DC4 3C</td>
</tr>
<tr>
<td>T</td>
<td>t 119</td>
<td>T D29</td>
<td>DBC 11</td>
<td>U 120</td>
<td>U D30</td>
<td>DC2 12</td>
</tr>
<tr>
<td>U</td>
<td>u 121</td>
<td>U D31</td>
<td>DC3 13</td>
<td>V 122</td>
<td>V D32</td>
<td>DC4 3C</td>
</tr>
<tr>
<td>V</td>
<td>v 123</td>
<td>V D33</td>
<td>DBC 11</td>
<td>W 124</td>
<td>W D34</td>
<td>DC2 12</td>
</tr>
<tr>
<td>W</td>
<td>w 125</td>
<td>W D35</td>
<td>DC3 13</td>
<td>X 126</td>
<td>X D36</td>
<td>DC4 3C</td>
</tr>
<tr>
<td>X</td>
<td>x 127</td>
<td>X D37</td>
<td>DBC 11</td>
<td>Y 128</td>
<td>Y D38</td>
<td>DC2 12</td>
</tr>
<tr>
<td>Y</td>
<td>y 129</td>
<td>Y D39</td>
<td>DC3 13</td>
<td>Z 130</td>
<td>Z D40</td>
<td>DC4 3C</td>
</tr>
<tr>
<td>Z</td>
<td>z 131</td>
<td>Z D41</td>
<td>DBC 11</td>
<td>0 132</td>
<td>0 D42</td>
<td>DC2 12</td>
</tr>
<tr>
<td>0</td>
<td>0 133</td>
<td>0 D43</td>
<td>DC3 13</td>
<td>1 134</td>
<td>1 D44</td>
<td>DC4 3C</td>
</tr>
<tr>
<td>1</td>
<td>1 135</td>
<td>1 D45</td>
<td>DBC 11</td>
<td>2 136</td>
<td>2 D46</td>
<td>DC2 12</td>
</tr>
<tr>
<td>2</td>
<td>2 137</td>
<td>2 D47</td>
<td>DC3 13</td>
<td>3 138</td>
<td>3 D48</td>
<td>DC4 3C</td>
</tr>
<tr>
<td>3</td>
<td>3 139</td>
<td>3 D49</td>
<td>DBC 11</td>
<td>4 140</td>
<td>4 D50</td>
<td>DC2 12</td>
</tr>
<tr>
<td>4</td>
<td>4 141</td>
<td>4 D51</td>
<td>DC3 13</td>
<td>5 142</td>
<td>5 D52</td>
<td>DC4 3C</td>
</tr>
<tr>
<td>5</td>
<td>5 143</td>
<td>5 D53</td>
<td>DBC 11</td>
<td>6 144</td>
<td>6 D54</td>
<td>DC2 12</td>
</tr>
<tr>
<td>6</td>
<td>6 145</td>
<td>6 D55</td>
<td>DC3 13</td>
<td>7 146</td>
<td>7 D56</td>
<td>DC4 3C</td>
</tr>
<tr>
<td>7</td>
<td>7 147</td>
<td>7 D57</td>
<td>DBC 11</td>
<td>8 148</td>
<td>8 D58</td>
<td>DC2 12</td>
</tr>
<tr>
<td>8</td>
<td>8 149</td>
<td>8 D59</td>
<td>DC3 13</td>
<td>9 150</td>
<td>9 D60</td>
<td>DC4 3C</td>
</tr>
<tr>
<td>9</td>
<td>9 151</td>
<td>9 D61</td>
<td>DBC 11</td>
<td>0 152</td>
<td>0 D62</td>
<td>DC2 12</td>
</tr>
<tr>
<td></td>
<td>dash</td>
<td>6A</td>
<td>-</td>
<td>4D</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>7E</td>
<td>+</td>
<td>4E</td>
<td></td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>\</td>
<td>7E</td>
<td>+</td>
<td>6E</td>
<td></td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>]</td>
<td>6B</td>
<td>&lt;</td>
<td>4C</td>
<td></td>
<td>&lt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6B</td>
<td>&gt;</td>
<td>4C</td>
<td></td>
<td>&gt;</td>
</tr>
<tr>
<td></td>
<td>/</td>
<td>6B</td>
<td>?</td>
<td>4C</td>
<td></td>
<td>?</td>
</tr>
<tr>
<td>Space</td>
<td>Space</td>
<td>Space</td>
<td>Space</td>
<td>Space</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Untranslatable characters ("-" in the above table) that are entered in transmit strings will be replaced by NULL (hex 00) during transmission.

1 CHAR displayed in Run mode
2 HEX byte trapped/transmitted
3 Enter the hex value for the \ character.
### Table D1-2

#### Keyboard-to-ASCII

<table>
<thead>
<tr>
<th>KEY</th>
<th>UNSHIFTED CHAR¹</th>
<th>HEX ²</th>
<th>SHIFTED CHAR¹</th>
<th>HEX ²</th>
<th>CONTROL CHAR¹</th>
<th>HEX ²</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>a</td>
<td>61</td>
<td>A</td>
<td>41</td>
<td>SOH</td>
<td>01</td>
</tr>
<tr>
<td>B</td>
<td>b</td>
<td>62</td>
<td>B</td>
<td>42</td>
<td>STX</td>
<td>02</td>
</tr>
<tr>
<td>C</td>
<td>c</td>
<td>63</td>
<td>C</td>
<td>43</td>
<td>ETX</td>
<td>03</td>
</tr>
<tr>
<td>D</td>
<td>d</td>
<td>64</td>
<td>D</td>
<td>44</td>
<td>EOT</td>
<td>04</td>
</tr>
<tr>
<td>E</td>
<td>e</td>
<td>65</td>
<td>E</td>
<td>45</td>
<td>ENQ</td>
<td>05</td>
</tr>
<tr>
<td>F</td>
<td>f</td>
<td>66</td>
<td>F</td>
<td>46</td>
<td>ACK</td>
<td>06</td>
</tr>
<tr>
<td>G</td>
<td>g</td>
<td>67</td>
<td>G</td>
<td>47</td>
<td>BEL</td>
<td>07</td>
</tr>
<tr>
<td>H</td>
<td>h</td>
<td>68</td>
<td>H</td>
<td>48</td>
<td>BS</td>
<td>08</td>
</tr>
<tr>
<td>I</td>
<td>i</td>
<td>69</td>
<td>I</td>
<td>49</td>
<td>HT</td>
<td>09</td>
</tr>
<tr>
<td>J</td>
<td>j</td>
<td>6A</td>
<td>J</td>
<td>4A</td>
<td>LF</td>
<td>0A</td>
</tr>
<tr>
<td>K</td>
<td>k</td>
<td>6B</td>
<td>K</td>
<td>4B</td>
<td>VT</td>
<td>0B</td>
</tr>
<tr>
<td>L</td>
<td>l</td>
<td>6C</td>
<td>L</td>
<td>4C</td>
<td>FF</td>
<td>0C</td>
</tr>
<tr>
<td>M</td>
<td>m</td>
<td>6D</td>
<td>M</td>
<td>4D</td>
<td>CR</td>
<td>0D</td>
</tr>
<tr>
<td>N</td>
<td>n</td>
<td>6E</td>
<td>N</td>
<td>4E</td>
<td>SO</td>
<td>0E</td>
</tr>
<tr>
<td>O</td>
<td>o</td>
<td>6F</td>
<td>O</td>
<td>4F</td>
<td>SI</td>
<td>0F</td>
</tr>
<tr>
<td>P</td>
<td>p</td>
<td>70</td>
<td>P</td>
<td>50</td>
<td>DLE</td>
<td>10</td>
</tr>
<tr>
<td>Q</td>
<td>q</td>
<td>71</td>
<td>Q</td>
<td>51</td>
<td>DC1</td>
<td>11</td>
</tr>
<tr>
<td>R</td>
<td>r</td>
<td>72</td>
<td>R</td>
<td>52</td>
<td>DC2</td>
<td>12</td>
</tr>
<tr>
<td>S</td>
<td>s</td>
<td>73</td>
<td>S</td>
<td>53</td>
<td>DC3</td>
<td>13</td>
</tr>
<tr>
<td>T</td>
<td>t</td>
<td>74</td>
<td>T</td>
<td>54</td>
<td>DC4</td>
<td>14</td>
</tr>
<tr>
<td>U</td>
<td>u</td>
<td>75</td>
<td>U</td>
<td>55</td>
<td>NAK</td>
<td>15</td>
</tr>
<tr>
<td>V</td>
<td>v</td>
<td>76</td>
<td>V</td>
<td>56</td>
<td>SYN</td>
<td>16</td>
</tr>
<tr>
<td>W</td>
<td>w</td>
<td>77</td>
<td>W</td>
<td>57</td>
<td>ETB</td>
<td>17</td>
</tr>
<tr>
<td>X</td>
<td>x</td>
<td>78</td>
<td>X</td>
<td>58</td>
<td>CAN</td>
<td>18</td>
</tr>
<tr>
<td>Y</td>
<td>y</td>
<td>79</td>
<td>Y</td>
<td>59</td>
<td>EM</td>
<td>19</td>
</tr>
<tr>
<td>Z</td>
<td>z</td>
<td>7A</td>
<td>Z</td>
<td>5A</td>
<td>SUB</td>
<td>1A</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>30</td>
<td></td>
<td>29</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>31</td>
<td></td>
<td>21</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>32</td>
<td>@</td>
<td>40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>33</td>
<td>#</td>
<td>23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>34</td>
<td>$</td>
<td>24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>35</td>
<td>%</td>
<td>25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>36</td>
<td>&amp;</td>
<td>26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>37</td>
<td>*</td>
<td>2A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>38</td>
<td>(</td>
<td>28</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>39</td>
<td>)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dash</td>
<td>dash</td>
<td>2D</td>
<td>underline</td>
<td>5F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>\</td>
<td>=</td>
<td>3D</td>
<td>+</td>
<td>2B</td>
<td>.</td>
<td>7E</td>
</tr>
<tr>
<td></td>
<td></td>
<td>\</td>
<td>5C</td>
<td>7</td>
<td>7C</td>
<td>DEL</td>
</tr>
<tr>
<td>]</td>
<td>)</td>
<td>5B</td>
<td>7B</td>
<td>7B</td>
<td>ESC</td>
<td>1B</td>
</tr>
<tr>
<td>::</td>
<td>::</td>
<td>5D</td>
<td>7D</td>
<td>7D</td>
<td>NUL</td>
<td>00</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>3B</td>
<td>.</td>
<td>3A</td>
<td>GS</td>
<td>1D</td>
</tr>
<tr>
<td>\</td>
<td>\</td>
<td>2C</td>
<td>&lt;</td>
<td>3C</td>
<td>RS</td>
<td>1E</td>
</tr>
<tr>
<td>/</td>
<td>/</td>
<td>2E</td>
<td>&gt;</td>
<td>3E</td>
<td>US</td>
<td>1F</td>
</tr>
<tr>
<td>Space</td>
<td>Space</td>
<td>20</td>
<td>Space</td>
<td>20</td>
<td>Space</td>
<td>20</td>
</tr>
</tbody>
</table>

¹CHAR displayed in Run mode  
²HEX byte trapped/transmitted (space parity)  
³Enter the hex value for the \ character.
### Table D1-3
### Keyboard-to-EBCD

<table>
<thead>
<tr>
<th>KEY</th>
<th>LOWER()&lt;sup&gt;1&lt;/sup&gt;</th>
<th>UPPER()&lt;sup&gt;2&lt;/sup&gt;</th>
<th>HEX&lt;sup&gt;3&lt;/sup&gt;</th>
<th>LOWER()&lt;sup&gt;1&lt;/sup&gt;</th>
<th>UPPER()&lt;sup&gt;2&lt;/sup&gt;</th>
<th>HEX&lt;sup&gt;3&lt;/sup&gt;</th>
<th>CHAR&lt;sup&gt;4&lt;/sup&gt;</th>
<th>HEX&lt;sup&gt;3&lt;/sup&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>a</td>
<td>A</td>
<td>23</td>
<td>a</td>
<td>A</td>
<td>23</td>
<td>SOH</td>
<td>3E</td>
</tr>
<tr>
<td>B</td>
<td>b</td>
<td>B</td>
<td>13</td>
<td>b</td>
<td>B</td>
<td>13</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>C</td>
<td>c</td>
<td>C</td>
<td>73</td>
<td>c</td>
<td>C</td>
<td>73</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>D</td>
<td>d</td>
<td>D</td>
<td>0B</td>
<td>d</td>
<td>D</td>
<td>0B</td>
<td>EOT</td>
<td>7C</td>
</tr>
<tr>
<td>E</td>
<td>e</td>
<td>E</td>
<td>6B</td>
<td>e</td>
<td>E</td>
<td>6B</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>F</td>
<td>f</td>
<td>F</td>
<td>5B</td>
<td>f</td>
<td>F</td>
<td>5B</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>G</td>
<td>g</td>
<td>G</td>
<td>3B</td>
<td>g</td>
<td>G</td>
<td>3B</td>
<td>BS</td>
<td>5D</td>
</tr>
<tr>
<td>H</td>
<td>h</td>
<td>H</td>
<td>07</td>
<td>h</td>
<td>H</td>
<td>07</td>
<td>HT</td>
<td>2F</td>
</tr>
<tr>
<td>I</td>
<td>i</td>
<td>I</td>
<td>67</td>
<td>i</td>
<td>I</td>
<td>67</td>
<td>LF</td>
<td>6E</td>
</tr>
<tr>
<td>J</td>
<td>j</td>
<td>J</td>
<td>61</td>
<td>j</td>
<td>J</td>
<td>61</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>K</td>
<td>k</td>
<td>K</td>
<td>51</td>
<td>k</td>
<td>K</td>
<td>51</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>M</td>
<td>m</td>
<td>M</td>
<td>49</td>
<td>m</td>
<td>M</td>
<td>49</td>
<td>CR</td>
<td>6D</td>
</tr>
<tr>
<td>N</td>
<td>n</td>
<td>N</td>
<td>29</td>
<td>n</td>
<td>N</td>
<td>29</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>O</td>
<td>o</td>
<td>O</td>
<td>19</td>
<td>o</td>
<td>O</td>
<td>19</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>P</td>
<td>p</td>
<td>P</td>
<td>79</td>
<td>p</td>
<td>P</td>
<td>79</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Q</td>
<td>q</td>
<td>Q</td>
<td>45</td>
<td>q</td>
<td>Q</td>
<td>45</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>R</td>
<td>r</td>
<td>R</td>
<td>25</td>
<td>r</td>
<td>R</td>
<td>25</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>S</td>
<td>s</td>
<td>S</td>
<td>52</td>
<td>s</td>
<td>S</td>
<td>52</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>T</td>
<td>t</td>
<td>T</td>
<td>32</td>
<td>t</td>
<td>T</td>
<td>32</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>U</td>
<td>u</td>
<td>U</td>
<td>4A</td>
<td>u</td>
<td>U</td>
<td>4A</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>V</td>
<td>v</td>
<td>V</td>
<td>2A</td>
<td>v</td>
<td>V</td>
<td>2A</td>
<td>SYN</td>
<td>3D</td>
</tr>
<tr>
<td>W</td>
<td>w</td>
<td>W</td>
<td>1A</td>
<td>w</td>
<td>W</td>
<td>1A</td>
<td>ETB</td>
<td>5E</td>
</tr>
<tr>
<td>X</td>
<td>x</td>
<td>X</td>
<td>7A</td>
<td>x</td>
<td>X</td>
<td>7A</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Z</td>
<td>z</td>
<td>Z</td>
<td>46</td>
<td>z</td>
<td>Z</td>
<td>46</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>&gt;</td>
<td>54</td>
<td>0</td>
<td>&gt;</td>
<td>54</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>=</td>
<td>20</td>
<td>1</td>
<td>=</td>
<td>20</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>&lt;</td>
<td>10</td>
<td>2</td>
<td>&lt;</td>
<td>10</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>@</td>
<td>02</td>
<td>3</td>
<td>@</td>
<td>02</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>#</td>
<td>34</td>
<td>4</td>
<td>#</td>
<td>34</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>$</td>
<td>75</td>
<td>5</td>
<td>$</td>
<td>75</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>%</td>
<td>68</td>
<td>6</td>
<td>%</td>
<td>68</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>&amp;</td>
<td>5B</td>
<td>7</td>
<td>&amp;</td>
<td>5B</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>*</td>
<td>43</td>
<td>8</td>
<td>*</td>
<td>43</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>(</td>
<td>64</td>
<td>9</td>
<td>(</td>
<td>64</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>dash</td>
<td>dash</td>
<td>dash</td>
<td>01</td>
<td>dash</td>
<td>dash</td>
<td>01</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>=</td>
<td>1</td>
<td>\</td>
<td>20</td>
<td>\</td>
<td>\</td>
<td>20</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>\</td>
<td>1\</td>
<td>F</td>
<td>1F</td>
<td>\</td>
<td>F</td>
<td>1F</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>l</td>
<td>3</td>
<td>#</td>
<td>4</td>
<td>3</td>
<td>#</td>
<td>4</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>;</td>
<td>6</td>
<td>:</td>
<td>8</td>
<td>;</td>
<td>:</td>
<td>8</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>.</td>
<td>6</td>
<td>?</td>
<td>34</td>
<td>.</td>
<td>?</td>
<td>34</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/</td>
<td>76</td>
<td>&gt;</td>
<td>38</td>
<td>/</td>
<td>&gt;</td>
<td>38</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Space</td>
<td>Space</td>
<td>Space</td>
<td>40</td>
<td>Space</td>
<td>Space</td>
<td>40</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Untranslatable characters ("--" in the above table) that are entered n transmit strings will be replaced by SPACE (hex 40) during transmission.

<sup>1</sup>CHAR displayed in Run mode if latest case-control character was lower
<sup>2</sup>CHAR displayed in Run mode if latest case-control character was upper
<sup>3</sup>HEX byte trapped/transmitted (odd parity)
<sup>4</sup>CHAR displayed in Run mode
<sup>5</sup>Enter the hex value for the \ character.
## Table D1-4

**Keyboard-to-XS-3 (SYN=35; EOM=55)**

<table>
<thead>
<tr>
<th>KEY</th>
<th>UNSHIFTED CHAR¹</th>
<th>UNSHIFTED HEX²</th>
<th>SHIFTED CHAR¹</th>
<th>SHIFTED HEX²</th>
<th>CONTROL CHAR¹</th>
<th>CONTROL HEX²</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
<td>54</td>
<td>A</td>
<td>54</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>B</td>
<td>15</td>
<td>B</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>C</td>
<td>16</td>
<td>C</td>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>D</td>
<td>57</td>
<td>D</td>
<td>57</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>E</td>
<td>58</td>
<td>E</td>
<td>58</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>19</td>
<td>F</td>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>G</td>
<td>1A</td>
<td>G</td>
<td>1A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>68</td>
<td>H</td>
<td>68</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>I</td>
<td>64</td>
<td>I</td>
<td>64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J</td>
<td>J</td>
<td>25</td>
<td>J</td>
<td>25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>K</td>
<td>67</td>
<td>K</td>
<td>26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>68</td>
<td>L</td>
<td>68</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>M</td>
<td>29</td>
<td>M</td>
<td>29</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>N</td>
<td>6B</td>
<td>N</td>
<td>6B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td>O</td>
<td>2C</td>
<td>O</td>
<td>2C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>P</td>
<td>75</td>
<td>P</td>
<td>75</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td>Q</td>
<td>76</td>
<td>Q</td>
<td>76</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>R</td>
<td>37</td>
<td>R</td>
<td>37</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>S</td>
<td>38</td>
<td>S</td>
<td>38</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>79</td>
<td>T</td>
<td>79</td>
<td></td>
<td></td>
</tr>
<tr>
<td>U</td>
<td>U</td>
<td>7A</td>
<td>U</td>
<td>7A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>V</td>
<td>38</td>
<td>V</td>
<td>38</td>
<td></td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>W</td>
<td>7B</td>
<td>W</td>
<td>7B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>7C</td>
<td>X</td>
<td>7C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>Y</td>
<td>43</td>
<td>Y</td>
<td>43</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td>Z</td>
<td>04</td>
<td>Z</td>
<td>04</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>45</td>
<td>0</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>46</td>
<td>1</td>
<td>1F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>07</td>
<td>2</td>
<td>62</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>08</td>
<td>3</td>
<td>6D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>49</td>
<td>4</td>
<td>2F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>4A</td>
<td>5</td>
<td>73</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>0B</td>
<td>6</td>
<td>61</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>4C</td>
<td>7</td>
<td>31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>02</td>
<td>8</td>
<td>70</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>Underline</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dash</td>
<td>Dash</td>
<td>0D</td>
<td>Dash</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>=</td>
<td>=</td>
<td>0D</td>
<td>=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>\</td>
<td>\</td>
<td>0D</td>
<td>\</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4F</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>01</td>
<td>:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>05</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>6E</td>
<td>.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>,</td>
<td>,</td>
<td>32</td>
<td>,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/</td>
<td>/</td>
<td>52</td>
<td>/</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Space</td>
<td>Space</td>
<td>40</td>
<td>Space</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Untranslatable characters ("--" in the above tables) that are entered in transmit strings will be replaced by NULL (hex 00) during transmission.

1. CHAR displayed in Run mode
2. HEX byte trapped/transmitted (odd parity)
3. Enter the hex value for the \ character.

D1-5
### Table D1-5
**Keyboard-to-IPARS**

<table>
<thead>
<tr>
<th>KEY</th>
<th>UNSHIFTED</th>
<th>SHIFTED</th>
<th>CONTROL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CHAR¹</td>
<td>HEX²</td>
<td>CHAR¹</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>31</td>
<td>-</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
<td>32</td>
<td>-</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
<td>33</td>
<td>-</td>
</tr>
<tr>
<td>D</td>
<td>D</td>
<td>34</td>
<td>-</td>
</tr>
<tr>
<td>E</td>
<td>E</td>
<td>35</td>
<td>-</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>36</td>
<td>-</td>
</tr>
<tr>
<td>G</td>
<td>G</td>
<td>37</td>
<td>-</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>38</td>
<td>-</td>
</tr>
<tr>
<td>I</td>
<td>I</td>
<td>39</td>
<td>-</td>
</tr>
<tr>
<td>J</td>
<td>J</td>
<td>1</td>
<td>CR</td>
</tr>
<tr>
<td>K</td>
<td>K</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
<td>4</td>
<td>CR</td>
</tr>
<tr>
<td>N</td>
<td>N</td>
<td>5</td>
<td>-</td>
</tr>
<tr>
<td>O</td>
<td>O</td>
<td>6</td>
<td>-</td>
</tr>
<tr>
<td>P</td>
<td>P</td>
<td>7</td>
<td>-</td>
</tr>
<tr>
<td>Q</td>
<td>Q</td>
<td>8</td>
<td>-</td>
</tr>
<tr>
<td>R</td>
<td>R</td>
<td>9</td>
<td>-</td>
</tr>
<tr>
<td>S</td>
<td>S</td>
<td>A</td>
<td>-</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>B</td>
<td>-</td>
</tr>
<tr>
<td>U</td>
<td>U</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>V</td>
<td>V</td>
<td>D</td>
<td>-</td>
</tr>
<tr>
<td>W</td>
<td>W</td>
<td>E</td>
<td>-</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>F</td>
<td>-</td>
</tr>
<tr>
<td>Y</td>
<td>Y</td>
<td>G</td>
<td>-</td>
</tr>
<tr>
<td>Z</td>
<td>Z</td>
<td>H</td>
<td>-</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>I</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>J</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>K</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>L</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>M</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>N</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>O</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>P</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>Q</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>R</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>S</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>U</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>V</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>W</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Y</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Z</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>?</td>
<td>-</td>
</tr>
</tbody>
</table>

1CHAR displayed in Run mode
2HEX byte trapped/transmitted

D1-6
### Table D1-6
**Keyboard-to-REVERSE EBCD**

<table>
<thead>
<tr>
<th>KEY</th>
<th>LOWER(1)</th>
<th>UPPER(2)</th>
<th>DEC</th>
<th>LOWER(1)</th>
<th>UPPER(2)</th>
<th>DEC</th>
<th>LOWER(1)</th>
<th>UPPER(2)</th>
<th>DEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>a</td>
<td>A</td>
<td>01</td>
<td>a</td>
<td>A</td>
<td>01</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>b</td>
<td>B</td>
<td>02</td>
<td>b</td>
<td>B</td>
<td>02</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>c</td>
<td>C</td>
<td>03</td>
<td>c</td>
<td>C</td>
<td>03</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>D</td>
<td>D</td>
<td>04</td>
<td>D</td>
<td>D</td>
<td>04</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>E</td>
<td>E</td>
<td>05</td>
<td>E</td>
<td>E</td>
<td>05</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>06</td>
<td>F</td>
<td>F</td>
<td>06</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>G</td>
<td>G</td>
<td>07</td>
<td>G</td>
<td>G</td>
<td>07</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>H</td>
<td>08</td>
<td>H</td>
<td>H</td>
<td>08</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>I</td>
<td>I</td>
<td>09</td>
<td>I</td>
<td>I</td>
<td>09</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J</td>
<td>J</td>
<td>J</td>
<td>10</td>
<td>J</td>
<td>J</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>K</td>
<td>K</td>
<td>11</td>
<td>K</td>
<td>K</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>L</td>
<td>12</td>
<td>L</td>
<td>L</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>M</td>
<td>M</td>
<td>13</td>
<td>M</td>
<td>M</td>
<td>13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>N</td>
<td>N</td>
<td>14</td>
<td>N</td>
<td>N</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td>O</td>
<td>O</td>
<td>15</td>
<td>O</td>
<td>O</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>P</td>
<td>P</td>
<td>16</td>
<td>P</td>
<td>P</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td>Q</td>
<td>Q</td>
<td>17</td>
<td>Q</td>
<td>Q</td>
<td>17</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>R</td>
<td>R</td>
<td>18</td>
<td>R</td>
<td>R</td>
<td>18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>S</td>
<td>S</td>
<td>19</td>
<td>S</td>
<td>S</td>
<td>19</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>20</td>
<td>T</td>
<td>T</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>U</td>
<td>U</td>
<td>U</td>
<td>21</td>
<td>U</td>
<td>U</td>
<td>21</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>V</td>
<td>V</td>
<td>22</td>
<td>V</td>
<td>V</td>
<td>22</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>W</td>
<td>W</td>
<td>23</td>
<td>W</td>
<td>W</td>
<td>23</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>24</td>
<td>X</td>
<td>X</td>
<td>24</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>25</td>
<td>Y</td>
<td>Y</td>
<td>25</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td>Z</td>
<td>Z</td>
<td>26</td>
<td>Z</td>
<td>Z</td>
<td>26</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Unshifted**

<table>
<thead>
<tr>
<th>KEY</th>
<th>LOWER(1)</th>
<th>UPPER(2)</th>
<th>DEC</th>
<th>LOWER(1)</th>
<th>UPPER(2)</th>
<th>DEC</th>
<th>LOWER(1)</th>
<th>UPPER(2)</th>
<th>DEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>@</td>
<td>#</td>
<td>30</td>
<td>$</td>
<td>%</td>
<td>31</td>
<td>$</td>
<td>%</td>
<td>32</td>
</tr>
<tr>
<td>1</td>
<td>$</td>
<td>!</td>
<td>33</td>
<td>&amp;</td>
<td>+</td>
<td>34</td>
<td>&amp;</td>
<td>+</td>
<td>35</td>
</tr>
<tr>
<td>2</td>
<td>&lt;</td>
<td>&gt;</td>
<td>36</td>
<td>&lt;</td>
<td>&gt;</td>
<td>37</td>
<td>&lt;</td>
<td>&gt;</td>
<td>38</td>
</tr>
<tr>
<td>3</td>
<td>&gt;</td>
<td>&lt;</td>
<td>39</td>
<td>&gt;</td>
<td>&lt;</td>
<td>40</td>
<td>&gt;</td>
<td>&lt;</td>
<td>41</td>
</tr>
<tr>
<td>5</td>
<td>#</td>
<td>#</td>
<td>45</td>
<td>#</td>
<td>#</td>
<td>46</td>
<td>#</td>
<td>#</td>
<td>47</td>
</tr>
<tr>
<td>6</td>
<td>!</td>
<td>!</td>
<td>48</td>
<td>!</td>
<td>!</td>
<td>49</td>
<td>!</td>
<td>!</td>
<td>50</td>
</tr>
<tr>
<td>7</td>
<td>+</td>
<td>+</td>
<td>51</td>
<td>+</td>
<td>+</td>
<td>52</td>
<td>+</td>
<td>+</td>
<td>53</td>
</tr>
<tr>
<td>8</td>
<td>-</td>
<td>-</td>
<td>54</td>
<td>-</td>
<td>-</td>
<td>55</td>
<td>-</td>
<td>-</td>
<td>56</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>1</td>
<td>57</td>
<td>0</td>
<td>1</td>
<td>58</td>
<td>0</td>
<td>1</td>
<td>59</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>60</td>
<td>1</td>
<td>0</td>
<td>61</td>
<td>1</td>
<td>0</td>
<td>62</td>
</tr>
<tr>
<td>\</td>
<td>/</td>
<td>/</td>
<td>63</td>
<td>/</td>
<td>/</td>
<td>64</td>
<td>/</td>
<td>/</td>
<td>65</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>66</td>
<td></td>
<td></td>
<td>67</td>
<td></td>
<td></td>
<td>68</td>
</tr>
</tbody>
</table>

Untranslatable characters ("--" in the above table) that are entered in transmit strings will be replaced by SPACE (hex 40) during transmission.

1CHAR displayed in Run mode if latest case-control character was lower
2CHAR displayed in Run mode if latest case-control character was upper
3HEX byte trapped/transmitted (odd parity)
4Enter the hex value for the ` character.
### Table D1-7
**Keyboard-to-SELECTRIC**

<table>
<thead>
<tr>
<th>KEY</th>
<th>LOWER (')</th>
<th>UPPER (')</th>
<th>HEX</th>
<th>LOWER (')</th>
<th>UPPER (')</th>
<th>HEX</th>
<th>CHAR</th>
<th>HEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>a</td>
<td>A</td>
<td>79</td>
<td>a</td>
<td>A</td>
<td>79</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>B</td>
<td>b</td>
<td>B</td>
<td>76</td>
<td>b</td>
<td>B</td>
<td>76</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>C</td>
<td>c</td>
<td>C</td>
<td>7A</td>
<td>c</td>
<td>C</td>
<td>7A</td>
<td>EOT</td>
<td>7C</td>
</tr>
<tr>
<td>D</td>
<td>d</td>
<td>D</td>
<td>2A</td>
<td>d</td>
<td>D</td>
<td>2A</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>E</td>
<td>e</td>
<td>E</td>
<td>4A</td>
<td>e</td>
<td>E</td>
<td>4A</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>F</td>
<td>f</td>
<td>F</td>
<td>73</td>
<td>f</td>
<td>F</td>
<td>73</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>G</td>
<td>g</td>
<td>G</td>
<td>23</td>
<td>g</td>
<td>G</td>
<td>23</td>
<td>BS</td>
<td>5D</td>
</tr>
<tr>
<td>H</td>
<td>h</td>
<td>H</td>
<td>1A</td>
<td>h</td>
<td>H</td>
<td>19</td>
<td>HT</td>
<td>2F</td>
</tr>
<tr>
<td>I</td>
<td>i</td>
<td>I</td>
<td>43</td>
<td>i</td>
<td>I</td>
<td>43</td>
<td>LF</td>
<td>6E</td>
</tr>
<tr>
<td>J</td>
<td>j</td>
<td>J</td>
<td>46</td>
<td>j</td>
<td>J</td>
<td>46</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>K</td>
<td>k</td>
<td>K</td>
<td>61</td>
<td>k</td>
<td>K</td>
<td>61</td>
<td>CR</td>
<td>6D</td>
</tr>
<tr>
<td>L</td>
<td>l</td>
<td>L</td>
<td>52</td>
<td>l</td>
<td>L</td>
<td>52</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>M</td>
<td>m</td>
<td>M</td>
<td>61</td>
<td>m</td>
<td>M</td>
<td>61</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>N</td>
<td>n</td>
<td>N</td>
<td>45</td>
<td>n</td>
<td>N</td>
<td>45</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>O</td>
<td>o</td>
<td>O</td>
<td>5B</td>
<td>o</td>
<td>O</td>
<td>5B</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>P</td>
<td>p</td>
<td>P</td>
<td>08</td>
<td>p</td>
<td>P</td>
<td>08</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Q</td>
<td>q</td>
<td>Q</td>
<td>29</td>
<td>q</td>
<td>Q</td>
<td>29</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>R</td>
<td>r</td>
<td>R</td>
<td>25</td>
<td>r</td>
<td>R</td>
<td>25</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>S</td>
<td>s</td>
<td>S</td>
<td>02</td>
<td>s</td>
<td>S</td>
<td>02</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>T</td>
<td>t</td>
<td>T</td>
<td>32</td>
<td>t</td>
<td>T</td>
<td>32</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>U</td>
<td>u</td>
<td>U</td>
<td>31</td>
<td>u</td>
<td>U</td>
<td>31</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>V</td>
<td>v</td>
<td>V</td>
<td>75</td>
<td>v</td>
<td>V</td>
<td>75</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>W</td>
<td>w</td>
<td>W</td>
<td>62</td>
<td>w</td>
<td>W</td>
<td>62</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>X</td>
<td>x</td>
<td>X</td>
<td>67</td>
<td>x</td>
<td>X</td>
<td>67</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Y</td>
<td>y</td>
<td>Y</td>
<td>54</td>
<td>y</td>
<td>Y</td>
<td>54</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Z</td>
<td>z</td>
<td>Z</td>
<td>64</td>
<td>z</td>
<td>Z</td>
<td>64</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>01</td>
<td>0</td>
<td>01</td>
<td>01</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>10</td>
<td>1</td>
<td>10</td>
<td>10</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>70</td>
<td>3</td>
<td>70</td>
<td>70</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>04</td>
<td>4</td>
<td>04</td>
<td>04</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>08</td>
<td>5</td>
<td>08</td>
<td>-</td>
<td>1C</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
<td>58</td>
<td>6</td>
<td>58</td>
<td>68</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>7</td>
<td>68</td>
<td>7</td>
<td>68</td>
<td>68</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>8</td>
<td>38</td>
<td>8</td>
<td>38</td>
<td>38</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>9</td>
<td>34</td>
<td>9</td>
<td>34</td>
<td>34</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>dash</td>
<td>dash</td>
<td>underline</td>
<td>37</td>
<td>dash</td>
<td>underline</td>
<td>37</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>=</td>
<td>+</td>
<td></td>
<td>13</td>
<td>=</td>
<td>+</td>
<td>13</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>\</td>
<td>\</td>
<td>1</td>
<td>20</td>
<td>\</td>
<td>-</td>
<td>-</td>
<td>DEL</td>
<td>7F</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>6B</td>
<td>-</td>
<td>:</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>49</td>
<td>6B</td>
<td>.</td>
<td>6B</td>
<td>49</td>
<td>RS</td>
<td>2C</td>
</tr>
<tr>
<td>/</td>
<td>/</td>
<td>3B</td>
<td>-</td>
<td>/</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>?</td>
<td>51</td>
<td>-</td>
<td>?</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Space</td>
<td>Space</td>
<td>Space</td>
<td>40</td>
<td>Space</td>
<td>Space</td>
<td>40</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Untranslatable characters ("-" in the above table) that are entered in transmit strings will be replaced by SPACE (hex 40) during transmission.

1. CHAR displayed in Run mode if latest case-control character was lower
2. CHAR displayed in Run mode if latest case-control character was upper
3. HEX byte trapped/transmitted (odd parity)
4. CHAR displayed in Run mode
5. Enter the hex value for the \ character.
### Appendix D1 Code Charts: Keyboard-to-Hex-Translation

#### Table D1-8
Keyboard-to-BAUDOT

<table>
<thead>
<tr>
<th>UNSHIFTED</th>
<th>SHIFTED</th>
<th>CONTROL</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY</td>
<td>LOWER()</td>
<td>UPPER()</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>dash</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
<td>?</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
<td>0E</td>
</tr>
<tr>
<td>D</td>
<td>D</td>
<td>09</td>
</tr>
<tr>
<td>E</td>
<td>E</td>
<td>01</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>0D</td>
</tr>
<tr>
<td>G</td>
<td>G &amp;</td>
<td>1A</td>
</tr>
<tr>
<td>H</td>
<td>H #</td>
<td>14</td>
</tr>
<tr>
<td>I</td>
<td>I 06</td>
<td>06</td>
</tr>
<tr>
<td>J</td>
<td>J 0F</td>
<td>0F</td>
</tr>
<tr>
<td>K</td>
<td>K 0C</td>
<td>0C</td>
</tr>
<tr>
<td>L</td>
<td>L 1D</td>
<td>1D</td>
</tr>
<tr>
<td>M</td>
<td>M 11</td>
<td>11</td>
</tr>
<tr>
<td>N</td>
<td>N 10</td>
<td>10</td>
</tr>
<tr>
<td>O</td>
<td>O 05</td>
<td>05</td>
</tr>
<tr>
<td>P</td>
<td>P 04</td>
<td>04</td>
</tr>
<tr>
<td>Q</td>
<td>Q 03</td>
<td>03</td>
</tr>
<tr>
<td>R</td>
<td>R 02</td>
<td>02</td>
</tr>
<tr>
<td>S</td>
<td>S 01</td>
<td>01</td>
</tr>
<tr>
<td>T</td>
<td>T 00</td>
<td>00</td>
</tr>
<tr>
<td>U</td>
<td>U 09</td>
<td>09</td>
</tr>
<tr>
<td>V</td>
<td>V 08</td>
<td>08</td>
</tr>
<tr>
<td>W</td>
<td>W 07</td>
<td>07</td>
</tr>
<tr>
<td>X</td>
<td>X 06</td>
<td>06</td>
</tr>
<tr>
<td>Y</td>
<td>Y 05</td>
<td>05</td>
</tr>
<tr>
<td>Z</td>
<td>Z 04</td>
<td>04</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>03</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>02</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>01</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>00</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0F</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>0E</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>0D</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>0C</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>0B</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>0A</td>
</tr>
<tr>
<td>dash</td>
<td>dash</td>
<td>03</td>
</tr>
<tr>
<td>\</td>
<td>\</td>
<td>1F</td>
</tr>
<tr>
<td>/</td>
<td>/</td>
<td>1E</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>0B</td>
</tr>
<tr>
<td>;</td>
<td>;</td>
<td>0C</td>
</tr>
<tr>
<td>_</td>
<td>_</td>
<td>0D</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0A</td>
</tr>
<tr>
<td>Space</td>
<td>Space</td>
<td>Space</td>
</tr>
</tbody>
</table>

Untranslatable characters ("-" in the above table) that are entered in transmit strings will be replaced by NULL (hex 00) during transmission.

1CHAR displayed in Run mode if latest case-control character was letter
2CHAR displayed in Run mode if latest case-control character was figure
3HEX byte searched for/transmitted
4Enter the hex value for the \ character.
## Table D1-9

### Keyboard-to-JIS7

<table>
<thead>
<tr>
<th>KEY</th>
<th>LOWER ($\text{a}_2$)$^1$</th>
<th>UPPER ($\text{b}_2$)$^2$</th>
<th>HEX$^3$</th>
<th>LOWER ($\text{a}_2$)$^1$</th>
<th>UPPER ($\text{b}_2$)$^2$</th>
<th>HEX$^3$</th>
<th>LOWER ($\text{a}_2$)$^1$</th>
<th>UPPER ($\text{b}_2$)$^2$</th>
<th>HEX$^3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>a 61 61</td>
<td>A ędzi</td>
<td>41</td>
<td>SOH SOH</td>
<td>01</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>b 62 62</td>
<td>B ęż</td>
<td>42</td>
<td>STX STX</td>
<td>02</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>c 63 63</td>
<td>C ę</td>
<td>43</td>
<td>ETX ETX</td>
<td>03</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>d 64 64</td>
<td>D ę</td>
<td>44</td>
<td>EOT EOT</td>
<td>04</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>e 65 65</td>
<td>E ę</td>
<td>45</td>
<td>ENQ ENQ</td>
<td>05</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>f 66 66</td>
<td>F ę</td>
<td>46</td>
<td>ACK ACK</td>
<td>06</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>g 67 67</td>
<td>G ę</td>
<td>47</td>
<td>BEL BEL</td>
<td>07</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>h 68 68</td>
<td>H ę</td>
<td>48</td>
<td>BS BS</td>
<td>08</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>l 69 69</td>
<td>I ę</td>
<td>49</td>
<td>HT HT</td>
<td>09</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J</td>
<td>j 6A 6A</td>
<td>J ę</td>
<td>4A</td>
<td>LF LF</td>
<td>0A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>k 6B 6B</td>
<td>K ę</td>
<td>4B</td>
<td>VT VT</td>
<td>0B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>l 6C 6C</td>
<td>L ę</td>
<td>4C</td>
<td>FF FF</td>
<td>0C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>m 6D 6D</td>
<td>M ę</td>
<td>4D</td>
<td>CR CR</td>
<td>0D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>n 6E 6E</td>
<td>N ę</td>
<td>4E</td>
<td>SO SO</td>
<td>0E</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td>o 6F 6F</td>
<td>O ę</td>
<td>4F</td>
<td>SI SI</td>
<td>0F</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>p 70 70</td>
<td>P ę</td>
<td>50</td>
<td>DLE DLE</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td>q 71 71</td>
<td>Q ę</td>
<td>51</td>
<td>DC1 DC1</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>r 72 72</td>
<td>R ę</td>
<td>52</td>
<td>DC2 DC2</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>s 73 73</td>
<td>S ę</td>
<td>53</td>
<td>DC3 DC3</td>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>t 74 74</td>
<td>T ę</td>
<td>54</td>
<td>DC4 DC4</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>U</td>
<td>u 75 75</td>
<td>U ę</td>
<td>55</td>
<td>NAK NAK</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>v 76 76</td>
<td>V ę</td>
<td>56</td>
<td>SYN SYN</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>w 77 77</td>
<td>W ę</td>
<td>57</td>
<td>ETB ETB</td>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>x 78 78</td>
<td>X ę</td>
<td>58</td>
<td>CAN CAN</td>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>y 79 79</td>
<td>Y ę</td>
<td>59</td>
<td>EM EM</td>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$^1$CHAR displayed in Run mode if latest case--control character was Shift In ($\text{a}_2$).

$^2$CHAR displayed in Run mode if latest case--control character was Shift Out ($\text{b}_2$).

$^3$HEX byte trapped/transmitted (space parity)
Table D1-9 (continued)

<table>
<thead>
<tr>
<th>UNSHIFTED</th>
<th>SHIFTED</th>
<th>CONTROL</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY</td>
<td>LOWER (3)</td>
<td>UPPER (3)</td>
</tr>
<tr>
<td>Z</td>
<td>z</td>
<td>7A</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>ア</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>イ</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>ウ</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>エ</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>オ</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>カ</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>キ</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>グ</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>ケ</td>
</tr>
<tr>
<td>dash</td>
<td>dash</td>
<td>ユ</td>
</tr>
<tr>
<td>=</td>
<td>=</td>
<td>ス</td>
</tr>
<tr>
<td>\</td>
<td>¥</td>
<td>ヨ</td>
</tr>
<tr>
<td>[</td>
<td>[</td>
<td>ロ</td>
</tr>
<tr>
<td>]</td>
<td>]</td>
<td>ロ</td>
</tr>
<tr>
<td>;</td>
<td>;</td>
<td>サ</td>
</tr>
<tr>
<td>'</td>
<td>'</td>
<td>ア</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>キ</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>ヨ</td>
</tr>
<tr>
<td>/</td>
<td>/</td>
<td>ツ</td>
</tr>
<tr>
<td>Space</td>
<td>Space</td>
<td>Space</td>
</tr>
</tbody>
</table>

1CHAR displayed in Run mode if latest case-control character was Shift In (\(\text{3}\)).
2CHAR displayed in Run mode if latest case-control character was Shift Out (\(\text{5}\)).
3HEX byte trapped/transmitted (space parity)
*Enter the hex value for the ¥ and ¥ characters.
Table D1-10
Keyboard-to-JIS8 (space parity)^1

<table>
<thead>
<tr>
<th>KEY</th>
<th>UNSHIFTED</th>
<th>SHIFTED</th>
<th>CONTROL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CHAR^2</td>
<td>HEX^3</td>
<td>CHAR^2</td>
</tr>
<tr>
<td>A</td>
<td>a 61</td>
<td>A 41</td>
<td>SOH</td>
</tr>
<tr>
<td>B</td>
<td>b 62</td>
<td>B 42</td>
<td>STX</td>
</tr>
<tr>
<td>C</td>
<td>c 63</td>
<td>C 43</td>
<td>ETX</td>
</tr>
<tr>
<td>D</td>
<td>d 64</td>
<td>D 44</td>
<td>EOT</td>
</tr>
<tr>
<td>E</td>
<td>e 65</td>
<td>E 45</td>
<td>ENQ</td>
</tr>
<tr>
<td>F</td>
<td>f 66</td>
<td>F 46</td>
<td>ACK</td>
</tr>
<tr>
<td>G</td>
<td>g 67</td>
<td>G 47</td>
<td>BEL</td>
</tr>
<tr>
<td>H</td>
<td>h 68</td>
<td>H 48</td>
<td>BS</td>
</tr>
<tr>
<td>I</td>
<td>i 69</td>
<td>I 49</td>
<td>HT</td>
</tr>
<tr>
<td>J</td>
<td>j 6A</td>
<td>J 4A</td>
<td>LF</td>
</tr>
<tr>
<td>K</td>
<td>k 6B</td>
<td>K 4B</td>
<td>VT</td>
</tr>
<tr>
<td>L</td>
<td>l 6C</td>
<td>L 4C</td>
<td>FF</td>
</tr>
<tr>
<td>M</td>
<td>m 6D</td>
<td>M 4D</td>
<td>CR</td>
</tr>
<tr>
<td>N</td>
<td>n 6E</td>
<td>N 4E</td>
<td>SO</td>
</tr>
<tr>
<td>O</td>
<td>o 6F</td>
<td>O 4F</td>
<td>SI</td>
</tr>
<tr>
<td>P</td>
<td>p 70</td>
<td>P 50</td>
<td>DLE</td>
</tr>
<tr>
<td>Q</td>
<td>q 71</td>
<td>Q 51</td>
<td>DC1</td>
</tr>
<tr>
<td>R</td>
<td>r 72</td>
<td>R 52</td>
<td>DC2</td>
</tr>
<tr>
<td>S</td>
<td>s 73</td>
<td>S 53</td>
<td>DC3</td>
</tr>
<tr>
<td>T</td>
<td>t 74</td>
<td>T 54</td>
<td>DC4</td>
</tr>
<tr>
<td>U</td>
<td>u 75</td>
<td>U 55</td>
<td>NAK</td>
</tr>
<tr>
<td>V</td>
<td>v 76</td>
<td>V 56</td>
<td>SYN</td>
</tr>
<tr>
<td>W</td>
<td>w 77</td>
<td>W 57</td>
<td>ETO</td>
</tr>
<tr>
<td>X</td>
<td>x 78</td>
<td>X 58</td>
<td>CAN</td>
</tr>
<tr>
<td>Y</td>
<td>y 79</td>
<td>Y 59</td>
<td>EM</td>
</tr>
<tr>
<td>Z</td>
<td>z 7A</td>
<td>Z 5A</td>
<td>SUB</td>
</tr>
<tr>
<td>0</td>
<td>0 30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1 31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2 32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3 33</td>
<td>@</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4 34</td>
<td>#</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5 35</td>
<td>$</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6 36</td>
<td>%</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>7 37</td>
<td>&amp;</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>8 38</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>9 39</td>
<td>(</td>
<td></td>
</tr>
<tr>
<td>dash</td>
<td>dash</td>
<td>2D</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>underline</td>
<td>5F</td>
</tr>
<tr>
<td>\</td>
<td>\</td>
<td>3D</td>
<td>+</td>
</tr>
<tr>
<td>$</td>
<td>$</td>
<td>5C</td>
<td>7C</td>
</tr>
<tr>
<td>(</td>
<td>(</td>
<td>5B</td>
<td>7B</td>
</tr>
<tr>
<td>)</td>
<td>)</td>
<td>5D</td>
<td>7D</td>
</tr>
<tr>
<td>;</td>
<td>;</td>
<td>3B</td>
<td>3A</td>
</tr>
<tr>
<td>&lt;</td>
<td>&lt;</td>
<td>2C</td>
<td>3C</td>
</tr>
<tr>
<td>&gt;</td>
<td>&gt;</td>
<td>2E</td>
<td>3E</td>
</tr>
<tr>
<td>?</td>
<td>?</td>
<td>2F</td>
<td>3F</td>
</tr>
<tr>
<td>Space</td>
<td>Space</td>
<td>20</td>
<td>Space</td>
</tr>
</tbody>
</table>

^1Hex data—entry will override parity
^2CHAR displayed in Run mode
^3HEX byte trapped/transmitted
*Enter the hex value for the * character.
### Table D1-11
Keyboard-to-JIS8 (mark parity)

<table>
<thead>
<tr>
<th>KEY</th>
<th>UNSHIFTED</th>
<th>SHIFTED</th>
<th>CONTROL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CHAR²</td>
<td>HEX²</td>
<td>CHAR²</td>
</tr>
<tr>
<td>A</td>
<td>E1</td>
<td>C1</td>
<td>81</td>
</tr>
<tr>
<td>B</td>
<td>E2</td>
<td>C2</td>
<td>82</td>
</tr>
<tr>
<td>C</td>
<td>E3</td>
<td>C3</td>
<td>83</td>
</tr>
<tr>
<td>D</td>
<td>E4</td>
<td>C4</td>
<td>84</td>
</tr>
<tr>
<td>E</td>
<td>E5</td>
<td>C5</td>
<td>85</td>
</tr>
<tr>
<td>F</td>
<td>E6</td>
<td>C6</td>
<td>86</td>
</tr>
<tr>
<td>G</td>
<td>E7</td>
<td>C7</td>
<td>87</td>
</tr>
<tr>
<td>H</td>
<td>E8</td>
<td>C8</td>
<td>88</td>
</tr>
<tr>
<td>I</td>
<td>E9</td>
<td>C9</td>
<td>89</td>
</tr>
<tr>
<td>J</td>
<td>EA</td>
<td>CA</td>
<td>8A</td>
</tr>
<tr>
<td>K</td>
<td>EB</td>
<td>CB</td>
<td>8B</td>
</tr>
<tr>
<td>L</td>
<td>EC</td>
<td>CC</td>
<td>8C</td>
</tr>
<tr>
<td>M</td>
<td>ED</td>
<td>CD</td>
<td>8D</td>
</tr>
<tr>
<td>N</td>
<td>EE</td>
<td>CE</td>
<td>8E</td>
</tr>
<tr>
<td>O</td>
<td>EF</td>
<td>CF</td>
<td>8F</td>
</tr>
<tr>
<td>P</td>
<td>F0</td>
<td>D0</td>
<td>90</td>
</tr>
<tr>
<td>Q</td>
<td>F1</td>
<td>D1</td>
<td>91</td>
</tr>
<tr>
<td>R</td>
<td>F2</td>
<td>D2</td>
<td>92</td>
</tr>
<tr>
<td>S</td>
<td>F3</td>
<td>D3</td>
<td>93</td>
</tr>
<tr>
<td>T</td>
<td>F4</td>
<td>D4</td>
<td>94</td>
</tr>
<tr>
<td>U</td>
<td>F5</td>
<td>D5</td>
<td>95</td>
</tr>
<tr>
<td>V</td>
<td>F6</td>
<td>D6</td>
<td>96</td>
</tr>
<tr>
<td>W</td>
<td>F7</td>
<td>D7</td>
<td>97</td>
</tr>
<tr>
<td>X</td>
<td>F8</td>
<td>D8</td>
<td>98</td>
</tr>
<tr>
<td>Y</td>
<td>F9</td>
<td>D9</td>
<td>99</td>
</tr>
<tr>
<td>Z</td>
<td>FA</td>
<td>DA</td>
<td>9A</td>
</tr>
<tr>
<td>0</td>
<td>B0</td>
<td>A9</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>B1</td>
<td>A1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>B2</td>
<td>C0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>B3</td>
<td>A3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>B4</td>
<td>A4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>B5</td>
<td>A5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>B6</td>
<td>DE</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>B7</td>
<td>A6</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>B8</td>
<td>AA</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>B9</td>
<td>A8</td>
<td></td>
</tr>
<tr>
<td>dash</td>
<td>AD</td>
<td>DF</td>
<td></td>
</tr>
</tbody>
</table>

1^Hex data-entry will override parity
2^CHAR displayed in Run mode
3^HEX byte trapped/transmitted
Table D1-11 (continued)

| KEY | UNSHIFTED |  |  |  | SHIFTED |  |  |  | CONTROL |  |  |  |
|-----|-----------|  |  |  | CHAR |  |  |  | HEX |  |  |  |
| =   | ス | BD |  |  | オ | AB |  |  | FE |  |  |  |
| \* | ワ | DC |  |  | FC | FC |  |  | E0 |  |  |  |
| [   | タ | DB |  |  | FB | FB |  |  | FF |  |  |  |
| ]   | サ | DD |  |  | FD | FD |  |  | 9B |  |  |  |
| ：  | タ | BB |  |  | BA | BA |  |  | 80 |  |  |  |
| '   | タ | A7 |  |  | ' | A2 |  |  | 9D |  |  |  |
| ,   | イ | AC |  |  | シ | BC |  |  | 9E |  |  |  |
| .   | エ | AE |  |  | セ | BE |  |  | 9F |  |  |  |
| /   | ツ | AF |  |  | ソ | BF |  |  | 9C |  |  |  |
| Space | A0 | A0 |  |  | A0 | A0 |  |  | A0 |  |  |  |

*Enter the hex value for the ツ character.
Appendix D2: Hex-to-Display Translation

The left-hand column in the following table (labeled "INPUT HEX") is the hex value presented on the Run-mode data screen when HEX display is turned on.

The remaining columns show the character that is presented for each hex value in each of the available code sets when hex display is turned off. Where no character in the code set corresponds to the hex value received, hex display is always used.

The bit in the “input hex” value that was received first by the INTERVIEW’s receivers will vary with the code. In the column heading for each code we have placed a small arrow next to the rightmost or leftmost bit to indicate which was the first bit received. In IPARS, for example, the leftmost bit is the first bit received.

We have tried also to indicate the significance of each bit. In EBCD, the third bit from the left in the hex value is the least significant (=1) bit, while the rightmost bit is the most significant (=32). This means that the first ten hex values in this code set are not really 00 through 09. Rather they are 00, 20, 10, 30, 08, 28, 18, 38, 04, and 24—corresponding to the characters SPACE, 1, 2, 3, 4, 5, 6, 7, 8, and 9, and corresponding also to the following binary series, which increments from left to right:

```
00000000
00100000
00010000
00110000
00001000
00101000
00011000
00111000
00000100
00100100
etc.
```
<table>
<thead>
<tr>
<th>INPUT HEX</th>
<th>ASCll</th>
<th>EBCDIC</th>
<th>EBCD</th>
<th>X-S</th>
<th>IPARS</th>
<th>REV EBCD</th>
<th>BAUDOT</th>
<th>SELECTRIC</th>
<th>J1ST</th>
<th>J1ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 NU</td>
<td>NU</td>
<td>space</td>
<td>space</td>
<td>hex</td>
<td>space</td>
<td>hex</td>
<td>hex</td>
<td>SP</td>
<td>SP</td>
<td>NU</td>
</tr>
<tr>
<td>01 SH</td>
<td>SH</td>
<td>dān</td>
<td>dān</td>
<td>]</td>
<td>1</td>
<td>1</td>
<td>E</td>
<td>3</td>
<td>!</td>
<td>hex</td>
</tr>
<tr>
<td>02 SX</td>
<td>SX</td>
<td>@</td>
<td>hex</td>
<td>dān</td>
<td>2</td>
<td>2</td>
<td>&lt;</td>
<td>LF</td>
<td>LF</td>
<td>t</td>
</tr>
<tr>
<td>03 EX</td>
<td>EX</td>
<td>&amp;</td>
<td>+</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>:</td>
<td>A</td>
<td>-</td>
<td>J</td>
</tr>
<tr>
<td>04 ET</td>
<td>hex</td>
<td>8</td>
<td>*</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>:</td>
<td>space</td>
<td>hex</td>
<td>4</td>
</tr>
<tr>
<td>05 EQ</td>
<td>HT</td>
<td>q</td>
<td>Q</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>%</td>
<td>S</td>
<td>hex</td>
<td>O</td>
</tr>
<tr>
<td>06 AK</td>
<td>hex</td>
<td>y</td>
<td>Y</td>
<td>3</td>
<td>6</td>
<td>6</td>
<td>:</td>
<td>I</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>07 BL</td>
<td>pad</td>
<td>h</td>
<td>H</td>
<td>4</td>
<td>7</td>
<td>7</td>
<td>&gt;</td>
<td>U</td>
<td>7</td>
<td>-</td>
</tr>
<tr>
<td>08 BS</td>
<td>hex</td>
<td>4</td>
<td>:</td>
<td>5</td>
<td>8</td>
<td>8</td>
<td>*</td>
<td>CR</td>
<td>CR</td>
<td>5</td>
</tr>
<tr>
<td>09 HT</td>
<td>hex</td>
<td>m</td>
<td>M</td>
<td>6</td>
<td>9</td>
<td>9</td>
<td>(</td>
<td>D</td>
<td>$</td>
<td>'</td>
</tr>
<tr>
<td>0A LF</td>
<td>hex</td>
<td>u</td>
<td>U</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>)</td>
<td>R</td>
<td>4</td>
<td>e</td>
</tr>
<tr>
<td>0B VT</td>
<td>VT</td>
<td>d</td>
<td>D</td>
<td>8</td>
<td>*</td>
<td>#</td>
<td>*</td>
<td>J</td>
<td>'</td>
<td>p</td>
</tr>
<tr>
<td>0C FF</td>
<td>FF</td>
<td>D2</td>
<td>D2</td>
<td>9</td>
<td>CR</td>
<td>{</td>
<td></td>
<td></td>
<td>)</td>
<td>N</td>
</tr>
<tr>
<td>0D CR</td>
<td>CR</td>
<td>hex</td>
<td>hex</td>
<td>\</td>
<td>hex</td>
<td>FF</td>
<td>FF</td>
<td>FF</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>0E SO</td>
<td>SO</td>
<td>hex</td>
<td>hex</td>
<td>/</td>
<td>hex</td>
<td>C</td>
<td>hex</td>
<td>SO</td>
<td>SO</td>
<td>SO</td>
</tr>
<tr>
<td>0F SI</td>
<td>Sl</td>
<td>D4</td>
<td>D4</td>
<td>[</td>
<td>hex</td>
<td>ET</td>
<td>ET</td>
<td>K</td>
<td>hex</td>
<td>SI</td>
</tr>
</tbody>
</table>

Table D2-1
Hex-to-Display Translation

1Select Bit Order/Polarity: NORMAL
2Select Bit Order/Polarity: REV-INVERT
3Select Bit Order/Polarity: REVERSE-NORM

D2-3
<table>
<thead>
<tr>
<th>INPUT</th>
<th>ASCII</th>
<th>EBCDIC</th>
<th>EBCD</th>
<th>XS-3</th>
<th>IPARS</th>
<th>REV EBCD</th>
<th>BAUDOT</th>
<th>SELECTRIC</th>
<th>JIS7</th>
<th>JIS8</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEX</td>
<td>LOWER</td>
<td>UPPER</td>
<td>LOWER</td>
<td>UPPER</td>
<td>LETTERS</td>
<td>FIGURES</td>
<td>LOWER</td>
<td>UPPER</td>
<td>LOWER</td>
<td>UPPER</td>
</tr>
<tr>
<td>20</td>
<td>space</td>
<td>hex</td>
<td>1</td>
<td>=</td>
<td>@</td>
<td>@</td>
<td>dash</td>
<td>dash</td>
<td>hex</td>
<td>hex</td>
</tr>
<tr>
<td>21</td>
<td>hex</td>
<td>j</td>
<td>J</td>
<td>*</td>
<td>J</td>
<td>J</td>
<td>J</td>
<td>hex</td>
<td>hex</td>
<td>m</td>
</tr>
<tr>
<td>22</td>
<td>FS</td>
<td>/</td>
<td>?</td>
<td>$</td>
<td>K</td>
<td>K</td>
<td>K</td>
<td>hex</td>
<td>hex</td>
<td>x</td>
</tr>
<tr>
<td>23</td>
<td>hex</td>
<td>a</td>
<td>A</td>
<td>!</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>hex</td>
<td>hex</td>
<td>g</td>
</tr>
<tr>
<td>24</td>
<td>$</td>
<td>hex</td>
<td>9</td>
<td>G</td>
<td>J</td>
<td>M</td>
<td>m</td>
<td>m</td>
<td>M</td>
<td>hex</td>
</tr>
<tr>
<td>25</td>
<td>%</td>
<td>LF</td>
<td>r</td>
<td>R</td>
<td>K</td>
<td>n</td>
<td>N</td>
<td>N</td>
<td>hex</td>
<td>S</td>
</tr>
<tr>
<td>26</td>
<td>&amp;</td>
<td>EB</td>
<td>z</td>
<td>Z</td>
<td>L</td>
<td>O</td>
<td>o</td>
<td>O</td>
<td>hex</td>
<td>hex</td>
</tr>
<tr>
<td>27</td>
<td>EC</td>
<td>l</td>
<td>I</td>
<td>M</td>
<td>P</td>
<td>p</td>
<td>p</td>
<td>hex</td>
<td>hex</td>
<td>y</td>
</tr>
<tr>
<td>28</td>
<td>hex</td>
<td>5</td>
<td>%</td>
<td>N</td>
<td>Q</td>
<td>q</td>
<td>Q</td>
<td>hex</td>
<td>hex</td>
<td>7</td>
</tr>
<tr>
<td>29</td>
<td>hex</td>
<td>n</td>
<td>N</td>
<td>O</td>
<td>R</td>
<td>r</td>
<td>R</td>
<td>hex</td>
<td>hex</td>
<td>r</td>
</tr>
<tr>
<td>2A</td>
<td>hex</td>
<td>e</td>
<td>E</td>
<td>Q</td>
<td>&lt;</td>
<td>$</td>
<td>!</td>
<td>hex</td>
<td>hex</td>
<td>d</td>
</tr>
<tr>
<td>2B</td>
<td>+</td>
<td>hex</td>
<td>e</td>
<td>V</td>
<td>P</td>
<td>:</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
</tr>
<tr>
<td>2C</td>
<td>hex</td>
<td>RS</td>
<td>RS</td>
<td>R</td>
<td>+</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>RS</td>
</tr>
<tr>
<td>2D</td>
<td>EQ</td>
<td>CR</td>
<td>CR</td>
<td>%</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>CR</td>
<td>CR</td>
<td>d</td>
</tr>
<tr>
<td>2E</td>
<td>AK</td>
<td>LF</td>
<td>LF</td>
<td>)</td>
<td>BS</td>
<td>BS</td>
<td>hex</td>
<td>hex</td>
<td>LF</td>
<td>LF</td>
</tr>
<tr>
<td>2F</td>
<td>/</td>
<td>BL</td>
<td>HT</td>
<td>HT</td>
<td>/</td>
<td>pad</td>
<td>pad</td>
<td>pad</td>
<td>pad</td>
<td>pad</td>
</tr>
</tbody>
</table>

*SYNC = even parity S (35₁₆).*
### Table D2-1 (continued)

<table>
<thead>
<tr>
<th>INPUT ASCII</th>
<th>EBCDIC</th>
<th>EBCD LOWER</th>
<th>UPPER</th>
<th>XS-3</th>
<th>IPARS LOWER</th>
<th>UPPER</th>
<th>REV EBCD LETTERS</th>
<th>FIGURES LOWER</th>
<th>UPPER</th>
<th>BAUDOT</th>
<th>SELECTRIC</th>
<th>JIS7</th>
<th>JIS8</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>@</td>
<td>space</td>
<td>space</td>
<td>space</td>
<td>space</td>
<td>hex</td>
<td>space</td>
<td>space</td>
<td>hex</td>
<td>SP</td>
<td>space</td>
<td>@</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>A</td>
<td>hex</td>
<td>space</td>
<td></td>
<td>space</td>
<td>hex</td>
<td>space</td>
<td></td>
<td>hex</td>
<td>SP</td>
<td>space</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>B</td>
<td>hex</td>
<td>@</td>
<td>hex</td>
<td>dash</td>
<td>hex</td>
<td>2</td>
<td>&lt;</td>
<td>hex</td>
<td>t</td>
<td>B</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>C</td>
<td>hex</td>
<td>&amp;</td>
<td>+</td>
<td>0</td>
<td>hex</td>
<td>3</td>
<td>;</td>
<td>hex</td>
<td>j</td>
<td>C</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>D</td>
<td>hex</td>
<td>8</td>
<td>*</td>
<td>1</td>
<td>hex</td>
<td>4</td>
<td>4 $</td>
<td>$</td>
<td>D</td>
<td>D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>E</td>
<td>hex</td>
<td>q</td>
<td>Q</td>
<td>2</td>
<td>hex</td>
<td>5</td>
<td>%</td>
<td>hex</td>
<td>o</td>
<td>E</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>F</td>
<td>hex</td>
<td>y</td>
<td>Y</td>
<td>3</td>
<td>hex</td>
<td>6</td>
<td>,</td>
<td>hex</td>
<td>l</td>
<td>F</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>G</td>
<td>hex</td>
<td>h</td>
<td>H</td>
<td>4</td>
<td>hex</td>
<td>7</td>
<td>&gt;</td>
<td>hex</td>
<td>/</td>
<td>G</td>
<td>G</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>H</td>
<td>hex</td>
<td>4</td>
<td>:</td>
<td>5</td>
<td>hex</td>
<td>8</td>
<td>*</td>
<td>hex</td>
<td>5 %</td>
<td>H</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>I</td>
<td>hex</td>
<td>m</td>
<td>M</td>
<td>6</td>
<td>hex</td>
<td>9</td>
<td>(</td>
<td>hex</td>
<td>'</td>
<td>I</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>4A</td>
<td>J</td>
<td>hex</td>
<td>u</td>
<td>U</td>
<td>7</td>
<td>hex</td>
<td>0</td>
<td>)</td>
<td>hex</td>
<td>e</td>
<td>J</td>
<td>J</td>
<td></td>
</tr>
<tr>
<td>4B</td>
<td>K</td>
<td>hex</td>
<td>d</td>
<td>D</td>
<td>8</td>
<td>hex</td>
<td>#</td>
<td>'</td>
<td>hex</td>
<td>p</td>
<td>K</td>
<td>K</td>
<td></td>
</tr>
<tr>
<td>4C</td>
<td>L</td>
<td>&lt;</td>
<td>D4</td>
<td>D2</td>
<td>9</td>
<td>hex</td>
<td>[</td>
<td>]</td>
<td>hex</td>
<td>hex</td>
<td>L</td>
<td>L</td>
<td></td>
</tr>
<tr>
<td>4D</td>
<td>M</td>
<td>\</td>
<td>hex</td>
<td>hex</td>
<td>\</td>
<td>hex</td>
<td>FF</td>
<td>FF</td>
<td>hex</td>
<td>hex</td>
<td>M</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>4E</td>
<td>N</td>
<td>+</td>
<td>hex</td>
<td>hex</td>
<td>;</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>n</td>
<td>N</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>4F</td>
<td>O</td>
<td>hex</td>
<td>D4</td>
<td>D4</td>
<td>I</td>
<td>hex</td>
<td>ET</td>
<td>ET</td>
<td>hex</td>
<td>hex</td>
<td>O</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>P</td>
<td>&amp;</td>
<td>2</td>
<td>&lt;</td>
<td>+</td>
<td>hex</td>
<td>2</td>
<td>@</td>
<td>hex</td>
<td>2</td>
<td>P</td>
<td>P</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>Q</td>
<td>hex</td>
<td>k</td>
<td>K</td>
<td>?</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td>hex</td>
<td></td>
<td>Q</td>
<td>Q</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>R</td>
<td>hex</td>
<td>s</td>
<td>S</td>
<td>hex</td>
<td>s</td>
<td>hex</td>
<td>n</td>
<td>hex</td>
<td>N</td>
<td>R</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>S</td>
<td>hex</td>
<td>b</td>
<td>B</td>
<td>?</td>
<td>hex</td>
<td>t</td>
<td>T</td>
<td>hex</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>T</td>
<td>hex</td>
<td>0</td>
<td>)</td>
<td>A</td>
<td>hex</td>
<td>u</td>
<td>U</td>
<td>hex</td>
<td>z</td>
<td>T</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>U</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>B*</td>
<td>hex</td>
<td>v</td>
<td>V</td>
<td>hex</td>
<td>hex</td>
<td>U</td>
<td>U</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>V</td>
<td>hex</td>
<td>hex</td>
<td>C</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>V</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>W</td>
<td>hex</td>
<td>hex</td>
<td>D</td>
<td>hex</td>
<td>x</td>
<td>X</td>
<td>hex</td>
<td>hex</td>
<td>W</td>
<td>W</td>
<td>W</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>X</td>
<td>hex</td>
<td>6</td>
<td>'</td>
<td>E</td>
<td>hex</td>
<td>y</td>
<td>Y</td>
<td>hex</td>
<td>6</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>Y</td>
<td>hex</td>
<td>o</td>
<td>O</td>
<td>F</td>
<td>hex</td>
<td>z</td>
<td>Z</td>
<td>hex</td>
<td>I</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>5A</td>
<td>Z</td>
<td>!</td>
<td>w</td>
<td>W</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>k</td>
<td>Z</td>
<td>Z</td>
<td>Z</td>
<td></td>
</tr>
<tr>
<td>5B</td>
<td>[</td>
<td>$</td>
<td>f</td>
<td>F</td>
<td>H</td>
<td>hex</td>
<td>hex</td>
<td>q</td>
<td>Q</td>
<td>[</td>
<td>[</td>
<td>[</td>
<td></td>
</tr>
<tr>
<td>5C</td>
<td>\</td>
<td>,</td>
<td>\</td>
<td>\</td>
<td>I</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td>hex</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td></td>
</tr>
<tr>
<td>5D</td>
<td>]</td>
<td>)</td>
<td>BS</td>
<td>BS</td>
<td>=</td>
<td>hex</td>
<td>LF</td>
<td>LF</td>
<td>hex</td>
<td>BS</td>
<td>]</td>
<td>]</td>
<td></td>
</tr>
<tr>
<td>5E</td>
<td>,</td>
<td>EB</td>
<td>EB</td>
<td>&lt;</td>
<td>EB</td>
<td>hex</td>
<td>EB</td>
<td>EB</td>
<td>EB</td>
<td>EB</td>
<td>]</td>
<td>]</td>
<td></td>
</tr>
<tr>
<td>5F</td>
<td></td>
<td>hex</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>hex</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td></td>
</tr>
</tbody>
</table>

*EOM = even parity B (55 U).
<table>
<thead>
<tr>
<th>INPUT</th>
<th>ASCII</th>
<th>EBCDIC</th>
<th>EBCD</th>
<th>XS-3</th>
<th>IPARS</th>
<th>REV EBCD</th>
<th>BAUDOT</th>
<th>SELECTRIC</th>
<th>JIS7</th>
<th>JIS8</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>'</td>
<td>dășh</td>
<td>1 =</td>
<td>@</td>
<td>hex</td>
<td>dășh</td>
<td>dășh</td>
<td>hex</td>
<td>1 [</td>
<td>' hex</td>
</tr>
<tr>
<td>61</td>
<td>a</td>
<td>J</td>
<td>J *</td>
<td>hex</td>
<td>J</td>
<td>hex</td>
<td>hex</td>
<td>m M a hex</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>62</td>
<td>b</td>
<td>hex</td>
<td>I</td>
<td>$</td>
<td>hex</td>
<td>k K</td>
<td>hex</td>
<td>x X b hex</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>c</td>
<td>hex</td>
<td>a A</td>
<td>L</td>
<td>hex</td>
<td>I L</td>
<td>hex</td>
<td>g G c hex</td>
<td>c</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>d</td>
<td>hex</td>
<td>9</td>
<td>J</td>
<td>hex</td>
<td>m M</td>
<td>hex</td>
<td>0 j d hex</td>
<td>d</td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>e</td>
<td>hex</td>
<td>r R</td>
<td>K</td>
<td>hex</td>
<td>n N</td>
<td>hex</td>
<td>s S e hex</td>
<td>e</td>
<td></td>
</tr>
<tr>
<td>66</td>
<td>f</td>
<td>hex</td>
<td>z Z</td>
<td>L</td>
<td>hex</td>
<td>o O</td>
<td>hex</td>
<td>h H f hex</td>
<td>f</td>
<td></td>
</tr>
<tr>
<td>67</td>
<td>g</td>
<td>hex</td>
<td>l M</td>
<td>p</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>y Y g hex</td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>68</td>
<td>h</td>
<td>hex</td>
<td>5 %</td>
<td>N</td>
<td>hex</td>
<td>q Q</td>
<td>hex</td>
<td>7 &amp; h hex</td>
<td>h</td>
<td></td>
</tr>
<tr>
<td>69</td>
<td>i</td>
<td>hex</td>
<td>n N</td>
<td>O</td>
<td>hex</td>
<td>r R</td>
<td>hex</td>
<td>r Ri i hex</td>
<td>i</td>
<td></td>
</tr>
<tr>
<td>6A</td>
<td>j</td>
<td>;</td>
<td>v</td>
<td>P</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>d D j hex</td>
<td>j</td>
<td></td>
</tr>
<tr>
<td>6B</td>
<td>k</td>
<td>,</td>
<td>E</td>
<td>Q</td>
<td>hex</td>
<td>$ !</td>
<td>hex</td>
<td>; : k hex</td>
<td>k</td>
<td></td>
</tr>
<tr>
<td>6C</td>
<td>l</td>
<td>%</td>
<td>RS</td>
<td>RS</td>
<td>R</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>RS</td>
<td>RS</td>
</tr>
<tr>
<td>6D</td>
<td>m</td>
<td>CR</td>
<td>CR</td>
<td>%</td>
<td>hex</td>
<td>CR CR</td>
<td>hex</td>
<td>CR CR</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>6E</td>
<td>n</td>
<td>&gt;</td>
<td>LF</td>
<td>LF</td>
<td>'</td>
<td>hex</td>
<td>hex</td>
<td>BF BS hex</td>
<td>BF</td>
<td>BS</td>
</tr>
<tr>
<td>6F</td>
<td>o</td>
<td>?</td>
<td>HT</td>
<td>HT</td>
<td>'</td>
<td>hex</td>
<td>hex</td>
<td>hex HT</td>
<td>o</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p</td>
<td>hex</td>
<td>3 :</td>
<td>hex</td>
<td>&amp;</td>
<td>+</td>
<td>hex</td>
<td>hex</td>
<td>3 #</td>
<td>p hex</td>
</tr>
<tr>
<td>70</td>
<td>q</td>
<td>hex</td>
<td>1 L</td>
<td>(</td>
<td>hex</td>
<td>a A</td>
<td>hex</td>
<td>hex v V q</td>
<td>hex</td>
<td>q</td>
</tr>
<tr>
<td>71</td>
<td>r</td>
<td>hex</td>
<td>T</td>
<td>,</td>
<td>hex</td>
<td>b B</td>
<td>hex</td>
<td>hex u U r</td>
<td>hex</td>
<td>r</td>
</tr>
<tr>
<td>72</td>
<td>s</td>
<td>hex</td>
<td>C</td>
<td>&amp;</td>
<td>hex</td>
<td>c C</td>
<td>hex</td>
<td>f F s hex</td>
<td>s</td>
<td></td>
</tr>
<tr>
<td>73</td>
<td>t</td>
<td>hex</td>
<td>#</td>
<td>&quot;</td>
<td>hex</td>
<td>d D</td>
<td>hex</td>
<td>q ( t hex</td>
<td>t</td>
<td></td>
</tr>
<tr>
<td>74</td>
<td>u</td>
<td>hex</td>
<td>$</td>
<td>I</td>
<td>hex</td>
<td>S E</td>
<td>hex</td>
<td>w W u hex</td>
<td>u</td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>v</td>
<td>hex</td>
<td>.</td>
<td>hex</td>
<td>T</td>
<td>f F</td>
<td>hex</td>
<td>b B v hex</td>
<td>v</td>
<td></td>
</tr>
<tr>
<td>76</td>
<td>w</td>
<td>hex</td>
<td>&gt;</td>
<td>hex</td>
<td>U</td>
<td>g G</td>
<td>hex</td>
<td>hex dășh</td>
<td>underline</td>
<td>w x hex</td>
</tr>
<tr>
<td>77</td>
<td>x</td>
<td>hex</td>
<td>7 &gt;</td>
<td>V</td>
<td>hex</td>
<td>h H</td>
<td>hex</td>
<td>hex 8 * x</td>
<td>x</td>
<td>hex</td>
</tr>
<tr>
<td>78</td>
<td>y</td>
<td>hex</td>
<td>p</td>
<td>P</td>
<td>hex</td>
<td>l I</td>
<td>hex</td>
<td>hex a A y</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>79</td>
<td>z</td>
<td>;</td>
<td>x</td>
<td>X</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex c C z</td>
<td>hex</td>
<td></td>
</tr>
<tr>
<td>7A</td>
<td>#</td>
<td>g G</td>
<td>Y</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex , ( hex</td>
<td>{</td>
<td></td>
</tr>
<tr>
<td>7B</td>
<td>@</td>
<td>ET</td>
<td>ET</td>
<td>Z</td>
<td>hex</td>
<td>- '</td>
<td>hex</td>
<td>hex ET ET</td>
<td>;</td>
<td></td>
</tr>
<tr>
<td>7C</td>
<td>SY</td>
<td>SY</td>
<td>)</td>
<td>hex</td>
<td>HT</td>
<td>HT</td>
<td>hex</td>
<td>hex hex</td>
<td>)</td>
<td></td>
</tr>
<tr>
<td>7D</td>
<td>SH</td>
<td>SH</td>
<td>&gt;</td>
<td>hex</td>
<td>\ \</td>
<td>hex</td>
<td>hex</td>
<td>hex hex</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>7E</td>
<td>pad</td>
<td>pad</td>
<td>pad</td>
<td>pad</td>
<td>pad</td>
<td>pad</td>
<td>pad</td>
<td>pad pad pad</td>
<td>pad</td>
<td>pad</td>
</tr>
<tr>
<td>7F</td>
<td>pad</td>
<td>pad</td>
<td>pad</td>
<td>pad</td>
<td>pad</td>
<td>pad</td>
<td>pad</td>
<td>pad pad pad</td>
<td>pad</td>
<td>pad</td>
</tr>
<tr>
<td>INPUT HEX</td>
<td>ASCII</td>
<td>EBCDIC</td>
<td>EBCD</td>
<td>XS-3</td>
<td>IPARS</td>
<td>REV EBCD</td>
<td>BAUDOT</td>
<td>SELECTRIC</td>
<td>JIS7</td>
<td>JIS8</td>
</tr>
<tr>
<td>-----------</td>
<td>-------</td>
<td>--------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
<td>----------</td>
<td>--------</td>
<td>------------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>80</td>
<td>NU</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>NU</td>
<td>NU</td>
<td>NU</td>
<td>NU</td>
<td>NU</td>
<td>NU</td>
</tr>
<tr>
<td>81</td>
<td>SH</td>
<td>a</td>
<td>hex</td>
<td>hex</td>
<td>SH</td>
<td>SH</td>
<td>SH</td>
<td>SH</td>
<td>SH</td>
<td>SH</td>
</tr>
<tr>
<td>82</td>
<td>SX</td>
<td>b</td>
<td>hex</td>
<td>hex</td>
<td>SX</td>
<td>SX</td>
<td>SX</td>
<td>SX</td>
<td>SX</td>
<td>SX</td>
</tr>
<tr>
<td>83</td>
<td>EX</td>
<td>c</td>
<td>hex</td>
<td>hex</td>
<td>EX</td>
<td>EX</td>
<td>EX</td>
<td>EX</td>
<td>EX</td>
<td>EX</td>
</tr>
<tr>
<td>84</td>
<td>ET</td>
<td>d</td>
<td>hex</td>
<td>hex</td>
<td>ET</td>
<td>ET</td>
<td>ET</td>
<td>ET</td>
<td>ET</td>
<td>ET</td>
</tr>
<tr>
<td>85</td>
<td>EQ</td>
<td>e</td>
<td>hex</td>
<td>hex</td>
<td>EQ</td>
<td>EQ</td>
<td>EQ</td>
<td>EQ</td>
<td>EQ</td>
<td>EQ</td>
</tr>
<tr>
<td>86</td>
<td>AK</td>
<td>f</td>
<td>hex</td>
<td>hex</td>
<td>AK</td>
<td>AK</td>
<td>AK</td>
<td>AK</td>
<td>AK</td>
<td>AK</td>
</tr>
<tr>
<td>87</td>
<td>BL</td>
<td>g</td>
<td>hex</td>
<td>hex</td>
<td>BL</td>
<td>BL</td>
<td>BL</td>
<td>BL</td>
<td>BL</td>
<td>BL</td>
</tr>
<tr>
<td>88</td>
<td>BS</td>
<td>h</td>
<td>hex</td>
<td>hex</td>
<td>BS</td>
<td>BS</td>
<td>BS</td>
<td>BS</td>
<td>BS</td>
<td>BS</td>
</tr>
<tr>
<td>89</td>
<td>HT</td>
<td>I</td>
<td>hex</td>
<td>hex</td>
<td>HT</td>
<td>HT</td>
<td>HT</td>
<td>HT</td>
<td>HT</td>
<td>HT</td>
</tr>
<tr>
<td>8A</td>
<td>LF</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>LF</td>
<td>LF</td>
<td>LF</td>
<td>LF</td>
<td>LF</td>
<td>LF</td>
</tr>
<tr>
<td>8B</td>
<td>VT</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>VT</td>
<td>VT</td>
<td>VT</td>
<td>VT</td>
<td>VT</td>
<td>VT</td>
</tr>
<tr>
<td>8C</td>
<td>FF</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>FF</td>
<td>FF</td>
<td>FF</td>
<td>FF</td>
<td>FF</td>
<td>FF</td>
</tr>
<tr>
<td>8D</td>
<td>CR</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>CR</td>
<td>CR</td>
<td>CR</td>
<td>CR</td>
<td>CR</td>
<td>CR</td>
</tr>
<tr>
<td>8E</td>
<td>SO</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>SO</td>
<td>SO</td>
<td>SO</td>
<td>SO</td>
<td>SO</td>
<td>SO</td>
</tr>
<tr>
<td>8F</td>
<td>SI</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>SI</td>
<td>SI</td>
<td>SI</td>
<td>SI</td>
<td>SI</td>
<td>SI</td>
</tr>
<tr>
<td>90</td>
<td>DL</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>DL</td>
<td>DL</td>
<td>DL</td>
<td>DL</td>
<td>DL</td>
<td>DL</td>
</tr>
<tr>
<td>91</td>
<td>D1</td>
<td>j</td>
<td>hex</td>
<td>hex</td>
<td>D1</td>
<td>D1</td>
<td>D1</td>
<td>D1</td>
<td>D1</td>
<td>D1</td>
</tr>
<tr>
<td>92</td>
<td>D2</td>
<td>k</td>
<td>hex</td>
<td>hex</td>
<td>D2</td>
<td>D2</td>
<td>D2</td>
<td>D2</td>
<td>D2</td>
<td>D2</td>
</tr>
<tr>
<td>93</td>
<td>D3</td>
<td>l</td>
<td>hex</td>
<td>hex</td>
<td>D3</td>
<td>D3</td>
<td>D3</td>
<td>D3</td>
<td>D3</td>
<td>D3</td>
</tr>
<tr>
<td>94</td>
<td>D4</td>
<td>m</td>
<td>hex</td>
<td>hex</td>
<td>D4</td>
<td>D4</td>
<td>D4</td>
<td>D4</td>
<td>D4</td>
<td>D4</td>
</tr>
<tr>
<td>95</td>
<td>NK</td>
<td>n</td>
<td>hex</td>
<td>hex</td>
<td>NK</td>
<td>NK</td>
<td>NK</td>
<td>NK</td>
<td>NK</td>
<td>NK</td>
</tr>
<tr>
<td>96</td>
<td>SY</td>
<td>o</td>
<td>hex</td>
<td>hex</td>
<td>SY</td>
<td>SY</td>
<td>SY</td>
<td>SY</td>
<td>SY</td>
<td>SY</td>
</tr>
<tr>
<td>97</td>
<td>EB</td>
<td>p</td>
<td>hex</td>
<td>hex</td>
<td>EB</td>
<td>EB</td>
<td>EB</td>
<td>EB</td>
<td>EB</td>
<td>EB</td>
</tr>
<tr>
<td>98</td>
<td>CN</td>
<td>q</td>
<td>hex</td>
<td>hex</td>
<td>CN</td>
<td>CN</td>
<td>CN</td>
<td>CN</td>
<td>CN</td>
<td>CN</td>
</tr>
<tr>
<td>99</td>
<td>EM</td>
<td>r</td>
<td>hex</td>
<td>hex</td>
<td>EM</td>
<td>EM</td>
<td>EM</td>
<td>EM</td>
<td>EM</td>
<td>EM</td>
</tr>
<tr>
<td>9A</td>
<td>SB</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>SB</td>
<td>SB</td>
<td>SB</td>
<td>SB</td>
<td>SB</td>
<td>SB</td>
</tr>
<tr>
<td>9B</td>
<td>EC</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>EC</td>
<td>EC</td>
<td>EC</td>
<td>EC</td>
<td>EC</td>
<td>EC</td>
</tr>
<tr>
<td>9C</td>
<td>FS</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>FS</td>
<td>FS</td>
<td>FS</td>
<td>FS</td>
<td>FS</td>
<td>FS</td>
</tr>
<tr>
<td>9D</td>
<td>GS</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>GS</td>
<td>GS</td>
<td>GS</td>
<td>GS</td>
<td>GS</td>
<td>GS</td>
</tr>
<tr>
<td>9E</td>
<td>RS</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>RS</td>
<td>RS</td>
<td>RS</td>
<td>RS</td>
<td>RS</td>
<td>RS</td>
</tr>
<tr>
<td>INPUT HEX</td>
<td>ASCII</td>
<td>EBCDIC</td>
<td>EBCD</td>
<td>XS-3</td>
<td>IPARS</td>
<td>REV EBCD</td>
<td>BAUDOT</td>
<td>SELECTRIC</td>
<td>JIS7</td>
<td>JIS8</td>
</tr>
<tr>
<td>-----------</td>
<td>-------</td>
<td>--------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
<td>----------</td>
<td>--------</td>
<td>------------</td>
<td>-----</td>
<td>------</td>
</tr>
<tr>
<td>A0 space</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>space</td>
<td>space</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A1 !</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>!</td>
<td>!</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A2 *</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>*</td>
<td>*</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A3 #</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>#</td>
<td>#</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A4 $</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>$</td>
<td>$</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A5 %</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>%</td>
<td>%</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A6 &amp;</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>&amp;</td>
<td>&amp;</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A7 '</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>'</td>
<td>'</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A8 (</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>(</td>
<td>(</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A9 )</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>)</td>
<td>)</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AA *</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>*</td>
<td>*</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AB +</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>+</td>
<td>+</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AC -</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>-</td>
<td>-</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AD dash</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>dash</td>
<td>dash</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AE -</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>-</td>
<td>-</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AF /</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>/</td>
<td>/</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B0 0</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>0</td>
<td>0</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B1 1</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>1</td>
<td>1</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B2 2</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>2</td>
<td>2</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B3 3</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>3</td>
<td>3</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B4 4</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>4</td>
<td>4</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B5 5</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>5</td>
<td>5</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B6 6</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>6</td>
<td>6</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B7 7</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>7</td>
<td>7</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B8 8</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>8</td>
<td>8</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B9 9</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>9</td>
<td>9</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BA :</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>:</td>
<td>:</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BB ;</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>;</td>
<td>;</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BC &lt;</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>&lt;</td>
<td>&lt;</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BD =</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>=</td>
<td>=</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BE &gt;</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>&gt;</td>
<td>&gt;</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BF ?</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>?</td>
<td>?</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INPUT ASCII</td>
<td>EBCDIC</td>
<td>EBCD LOWER UPPER</td>
<td>XS-3</td>
<td>IPARS</td>
<td>REV EBCD LOWER UPPER</td>
<td>BAUDOT LETTERS FIGURES</td>
<td>SELECTRIC LOWER UPPER</td>
<td>JIS7</td>
<td>JIS8</td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td>--------</td>
<td>------------------</td>
<td>------</td>
<td>-------</td>
<td>----------------------</td>
<td>------------------------</td>
<td>-----------------------</td>
<td>------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>C0</td>
<td>@</td>
<td>(</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C1</td>
<td>A</td>
<td>A</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C2</td>
<td>B</td>
<td>B</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C3</td>
<td>C</td>
<td>C</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C4</td>
<td>D</td>
<td>D</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C5</td>
<td>E</td>
<td>E</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C6</td>
<td>F</td>
<td>F</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C7</td>
<td>G</td>
<td>G</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C8</td>
<td>H</td>
<td>H</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C9</td>
<td>I</td>
<td>I</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CA</td>
<td>J</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CB</td>
<td>K</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC</td>
<td>L</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CD</td>
<td>M</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CE</td>
<td>N</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CF</td>
<td>O</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D0</td>
<td>P</td>
<td>)</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D1</td>
<td>Q</td>
<td>J</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D2</td>
<td>R</td>
<td>K</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D3</td>
<td>S</td>
<td>L</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D4</td>
<td>T</td>
<td>M</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D5</td>
<td>U</td>
<td>N</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D6</td>
<td>V</td>
<td>O</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D7</td>
<td>W</td>
<td>P</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D8</td>
<td>X</td>
<td>Q</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D9</td>
<td>Y</td>
<td>R</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DA</td>
<td>Z</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB</td>
<td>I</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DC</td>
<td>\</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DD</td>
<td></td>
<td></td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DE</td>
<td>*</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DF</td>
<td>-</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INPUT HEX</td>
<td>ASCII</td>
<td>EBCDIC</td>
<td>EBCD</td>
<td>XS-3</td>
<td>IPARS</td>
<td>REV EBCD</td>
<td>BAUDOT</td>
<td>SELECTRIC</td>
<td>JIS7</td>
<td>JIS8</td>
</tr>
<tr>
<td>-----------</td>
<td>-------</td>
<td>--------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
<td>----------</td>
<td>--------</td>
<td>------------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>E0</td>
<td>'</td>
<td></td>
<td>hex</td>
<td>hex</td>
<td></td>
<td>a</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E1</td>
<td>a</td>
<td></td>
<td>hex</td>
<td>hex</td>
<td></td>
<td>b</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E2</td>
<td>b</td>
<td></td>
<td>hex</td>
<td>hex</td>
<td></td>
<td>c</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E3</td>
<td>c</td>
<td></td>
<td>hex</td>
<td>hex</td>
<td></td>
<td>d</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E4</td>
<td>d</td>
<td></td>
<td>hex</td>
<td>hex</td>
<td></td>
<td>e</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E5</td>
<td>e</td>
<td></td>
<td>hex</td>
<td>hex</td>
<td></td>
<td>f</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E6</td>
<td>f</td>
<td></td>
<td>hex</td>
<td>hex</td>
<td></td>
<td>g</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E7</td>
<td>g</td>
<td></td>
<td>hex</td>
<td>hex</td>
<td></td>
<td>h</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E8</td>
<td>h</td>
<td></td>
<td>hex</td>
<td>hex</td>
<td></td>
<td>i</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E9</td>
<td>i</td>
<td></td>
<td>hex</td>
<td>hex</td>
<td></td>
<td>j</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EA</td>
<td>j</td>
<td></td>
<td>hex</td>
<td>hex</td>
<td></td>
<td>k</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EB</td>
<td>k</td>
<td></td>
<td>hex</td>
<td>hex</td>
<td></td>
<td>l</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EC</td>
<td>l</td>
<td></td>
<td>hex</td>
<td>hex</td>
<td></td>
<td>m</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ED</td>
<td>m</td>
<td></td>
<td>hex</td>
<td>hex</td>
<td></td>
<td>n</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EE</td>
<td>n</td>
<td></td>
<td>hex</td>
<td>hex</td>
<td></td>
<td>o</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EF</td>
<td>o</td>
<td></td>
<td>hex</td>
<td>hex</td>
<td></td>
<td>p</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F0</td>
<td>p</td>
<td></td>
<td>hex</td>
<td>hex</td>
<td></td>
<td>q</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F1</td>
<td>q</td>
<td></td>
<td>hex</td>
<td>hex</td>
<td></td>
<td>r</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F2</td>
<td>r</td>
<td></td>
<td>hex</td>
<td>hex</td>
<td></td>
<td>s</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F3</td>
<td>s</td>
<td></td>
<td>hex</td>
<td>hex</td>
<td></td>
<td>t</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F4</td>
<td>t</td>
<td></td>
<td>hex</td>
<td>hex</td>
<td></td>
<td>u</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F5</td>
<td>u</td>
<td></td>
<td>hex</td>
<td>hex</td>
<td></td>
<td>v</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F6</td>
<td>v</td>
<td></td>
<td>hex</td>
<td>hex</td>
<td></td>
<td>w</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F7</td>
<td>w</td>
<td></td>
<td>hex</td>
<td>hex</td>
<td></td>
<td>x</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F8</td>
<td>x</td>
<td></td>
<td>hex</td>
<td>hex</td>
<td></td>
<td>y</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F9</td>
<td>y</td>
<td></td>
<td>hex</td>
<td>hex</td>
<td></td>
<td>z</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FA</td>
<td>z</td>
<td></td>
<td>hex</td>
<td>hex</td>
<td></td>
<td>{</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FB</td>
<td>{</td>
<td></td>
<td>hex</td>
<td>hex</td>
<td></td>
<td>;</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FC</td>
<td>;</td>
<td></td>
<td>hex</td>
<td>hex</td>
<td></td>
<td>}</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FD</td>
<td>}</td>
<td></td>
<td>hex</td>
<td>hex</td>
<td></td>
<td>-</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FE</td>
<td>-</td>
<td></td>
<td>hex</td>
<td>hex</td>
<td></td>
<td>pad</td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FF</td>
<td>pad</td>
<td></td>
<td>hex</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix D3: User-Defined Codes

The character set shown in Table D3-1 can be used to adapt existing code sets or to create customized codes. Follow the steps in the example below to create a new code set.

As an example, we’ll change the standard ASCII code set to one which includes the ¥ (yen) symbol.

1. **Determine hex values.** First, we will determine which hexadecimal value or values we want to have generate the ¥ symbol, one for space parity and one for mark parity. In our example, the values will be hexadecimal 5C and DC.

2. **Read existing code file to spreadsheet.** Whether adapting an existing code set or creating a new one, use an existing code file as a template. (Some files include shifted and unshifted coding.) Go to the Protocol Spreadsheet and press **[F2]**, **[F1]**, **[F7]**, **[F3]**. Enter the name of the file when prompted. The absolute pathname of the standard ASCII code file is `HRD/sys/codes/ASCII`. Press **[Esc]**. Do not use the Load command on the File Maintenance screen to access the file.

   The ASCII code set will be displayed on the Protocol Spreadsheet, as in Figure D3-1. Initially, the file is highlighted on the Protocol Spreadsheet in reverse video. You may clear the highlighting by pressing **[Esc]**, **[F3]**. Since you will be writing your revised code set back to a file, however, you may want to retain the highlighting. Then you will not have to identify the block again before writing.

3. **Locate position.** Positions in the code proceed sequentially, beginning with hexadecimal 00 and ending with FF. Each row in the code table contains eight elements. The first two rows, for example, correspond to hex 00 through 0F. The next two rows contain elements in positions 10 through 1F, and so on. Move the cursor to position 5C.
** Protocol Spreadsheet **

Name of file: HRD/sys/codes/ASCII

/  Code for ASCII
/

Version: 3;

Name: "ASCII";

To Graphic:

nu sh sx ex et eq ak bl
bs ht lf vt ff cr so si
d1 d1 d2 d3 d4 nk sy eb
cn em sb ec fs gs rs us
', ',', ',', '!', '#', '$', '%', '&', '!
'( ' ', '*', '+', ',', '-', '.', '/'

** Figure D3-1 ** When the standard ASCII code file is written to the Protocol Spreadsheet, a code table appears with 32 rows of eight elements per row, corresponding to 256 possible hex values.

4. ** Enter new code. ** Replace the entry with a new value. Refer to Table D3-1. All values under "Code-Table Entry" are three-digit hexadecimals. A leading zero identifies an entry as a numerical value and guarantees accurate translation. Notice in Figure D3-1 that there is special notation for ASCII control characters. A d1 entry, for example, translates as the ASCII control character \. Entered as 0d1 (or 0D1), the meaning is \. Values which begin with a digit in the range 0–9, 80 for example, do not strictly require the leading zero. Also notice in Figure D3-1 and Figure D3-2 that alphanumerics may be entered as character constants. A set of single quotation marks surrounds a character constant, an alternative way of entering ASCII keyboard characters.

In our example, replace the delimiter `\` with 080. Figure D3-2 shows the set after the first replacement. Next, locate and edit position DC.

D3-2
**Protocol Spreadsheet**

Name of file: HRD/sys/codes/ASCII

To Graphic:

<table>
<thead>
<tr>
<th>nu</th>
<th>sh</th>
<th>sx</th>
<th>ex</th>
<th>et</th>
<th>eq</th>
<th>ak</th>
<th>bl</th>
</tr>
</thead>
<tbody>
<tr>
<td>bs</td>
<td>ht</td>
<td>lf</td>
<td>vt</td>
<td>ff</td>
<td>cr</td>
<td>so</td>
<td>si</td>
</tr>
<tr>
<td>dl</td>
<td>d1</td>
<td>d2</td>
<td>d3</td>
<td>d4</td>
<td>nk</td>
<td>sy</td>
<td>eb</td>
</tr>
<tr>
<td>cn</td>
<td>em</td>
<td>sb</td>
<td>ec</td>
<td>fs</td>
<td>gs</td>
<td>rs</td>
<td>us</td>
</tr>
<tr>
<td>','</td>
<td>';'</td>
<td>','</td>
<td>'#'</td>
<td>'$'</td>
<td>'%'</td>
<td>'&amp;'</td>
<td>','</td>
</tr>
<tr>
<td>'('</td>
<td>','</td>
<td>'+'</td>
<td>'-'</td>
<td>','</td>
<td>','</td>
<td>','</td>
<td>','</td>
</tr>
<tr>
<td>'0'</td>
<td>'1'</td>
<td>'2'</td>
<td>'3'</td>
<td>'4'</td>
<td>'5'</td>
<td>'6'</td>
<td>'7'</td>
</tr>
<tr>
<td>'8'</td>
<td>'9'</td>
<td>':'</td>
<td>';'</td>
<td>'&lt;'</td>
<td>'='</td>
<td>'&gt;'</td>
<td>'?'</td>
</tr>
<tr>
<td>'@'</td>
<td>'A'</td>
<td>'B'</td>
<td>'C'</td>
<td>'D'</td>
<td>'E'</td>
<td>'F'</td>
<td>'G'</td>
</tr>
<tr>
<td>'H'</td>
<td>'I'</td>
<td>'J'</td>
<td>'K'</td>
<td>'L'</td>
<td>'M'</td>
<td>'N'</td>
<td>'O'</td>
</tr>
<tr>
<td>'P'</td>
<td>'Q'</td>
<td>'R'</td>
<td>'S'</td>
<td>'T'</td>
<td>'U'</td>
<td>'V'</td>
<td>'W'</td>
</tr>
<tr>
<td>'X'</td>
<td>'Y'</td>
<td>'Z'</td>
<td>'['</td>
<td>'080'</td>
<td>']'</td>
<td>'^'</td>
<td>'_'</td>
</tr>
</tbody>
</table>

Figure D3-2 On the bottom line of the spreadsheet, the entry 080 has replaced the previous entry. On Table D3-1, 080 corresponds to the yen symbol.

5. **Write file to disk.** If you cleared the highlighting, mark the file via the BLOCK, BEGIN, and END softkey selections. Use the BLOCK, IN/OUT, and WRITE/U commands to write your code to the disk. Give the file a different name to prevent an existing file from being overwritten.

6. **Reboot.** Turn the INTERVIEW off. When you turn the unit back on, it will reboot and automatically load in the new (or edited) code set. The first seven characters in the name of the code file will be displayed as a softkey selection for the Code field on the Line Setup screen.

**NOTE:** If your code contains an error—a hexadecimal value does not begin with a digit, for example—it will not be loaded into the INTERVIEW's memory, even if it appears as a Code selection on the Line Setup menu. Usually, the standard ASCII code will be used instead.
<table>
<thead>
<tr>
<th>Character</th>
<th>Code-Table Entry</th>
<th>Character</th>
<th>Code-Table Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>¢</td>
<td>000</td>
<td>£</td>
<td>01b</td>
</tr>
<tr>
<td>£</td>
<td>001</td>
<td>$</td>
<td>01c</td>
</tr>
<tr>
<td>¥</td>
<td>002</td>
<td>¥</td>
<td>01d</td>
</tr>
<tr>
<td>§</td>
<td>003</td>
<td>¥</td>
<td>01e</td>
</tr>
<tr>
<td>¶</td>
<td>004</td>
<td>¥</td>
<td>01f</td>
</tr>
<tr>
<td>·</td>
<td>005</td>
<td>(space)</td>
<td>020</td>
</tr>
<tr>
<td>;</td>
<td>006</td>
<td>!</td>
<td>021</td>
</tr>
<tr>
<td>'</td>
<td>007</td>
<td>&quot;</td>
<td>022</td>
</tr>
<tr>
<td>#</td>
<td>008</td>
<td>#</td>
<td>023</td>
</tr>
<tr>
<td>$</td>
<td>009</td>
<td>$</td>
<td>024</td>
</tr>
<tr>
<td>%</td>
<td>00a</td>
<td>%</td>
<td>025</td>
</tr>
<tr>
<td>&amp;</td>
<td>00b</td>
<td>&amp;</td>
<td>026</td>
</tr>
<tr>
<td>'</td>
<td>00c</td>
<td>`</td>
<td>027</td>
</tr>
<tr>
<td>(</td>
<td>00d</td>
<td>(</td>
<td>028</td>
</tr>
<tr>
<td>)</td>
<td>00e</td>
<td>)</td>
<td>029</td>
</tr>
<tr>
<td>*</td>
<td>00f</td>
<td>*</td>
<td>02a</td>
</tr>
<tr>
<td>+</td>
<td>010</td>
<td>+</td>
<td>02b</td>
</tr>
<tr>
<td>,</td>
<td>011</td>
<td>,</td>
<td>02c</td>
</tr>
<tr>
<td>-</td>
<td>012</td>
<td>-</td>
<td>02d</td>
</tr>
<tr>
<td>.</td>
<td>013</td>
<td>.</td>
<td>02e</td>
</tr>
<tr>
<td>/</td>
<td>014</td>
<td>/</td>
<td>02f</td>
</tr>
<tr>
<td>0</td>
<td>015</td>
<td>0</td>
<td>030</td>
</tr>
<tr>
<td>1</td>
<td>016</td>
<td>1</td>
<td>031</td>
</tr>
<tr>
<td>2</td>
<td>017</td>
<td>2</td>
<td>032</td>
</tr>
<tr>
<td>3</td>
<td>018</td>
<td>3</td>
<td>033</td>
</tr>
</tbody>
</table>

D3-4
### Table D3-1 (continued)

<table>
<thead>
<tr>
<th>Character</th>
<th>Code–Table Entry</th>
<th>Character</th>
<th>Code–Table Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>034</td>
<td>P</td>
<td>050</td>
</tr>
<tr>
<td>5</td>
<td>035</td>
<td>Q</td>
<td>051</td>
</tr>
<tr>
<td>6</td>
<td>036</td>
<td>R</td>
<td>052</td>
</tr>
<tr>
<td>7</td>
<td>037</td>
<td>S</td>
<td>053</td>
</tr>
<tr>
<td>8</td>
<td>038</td>
<td>T</td>
<td>054</td>
</tr>
<tr>
<td>9</td>
<td>039</td>
<td>U</td>
<td>055</td>
</tr>
<tr>
<td>:</td>
<td>03a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>;</td>
<td>03b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;</td>
<td>03c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>=</td>
<td>03d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt;</td>
<td>03e</td>
<td></td>
<td></td>
</tr>
<tr>
<td>?</td>
<td>03f</td>
<td></td>
<td></td>
</tr>
<tr>
<td>@</td>
<td>040</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>041</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>042</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>043</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>044</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>045</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>046</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>047</td>
<td></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>048</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>049</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J</td>
<td>04a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>04b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>04c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>04d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>04e</td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td>04f</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

D3-5
<table>
<thead>
<tr>
<th>Character</th>
<th>Code-Table Entry</th>
<th>Character</th>
<th>Code-Table Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>j</td>
<td>06a</td>
<td>.yml</td>
<td>086</td>
</tr>
<tr>
<td>k</td>
<td>06b</td>
<td>月下</td>
<td>087</td>
</tr>
<tr>
<td>l</td>
<td>06c</td>
<td>丫</td>
<td>088</td>
</tr>
<tr>
<td>m</td>
<td>06d</td>
<td>う</td>
<td>089</td>
</tr>
<tr>
<td>n</td>
<td>06e</td>
<td>え</td>
<td>08a</td>
</tr>
<tr>
<td>o</td>
<td>06f</td>
<td>オ</td>
<td>08b</td>
</tr>
<tr>
<td>p</td>
<td>070</td>
<td>が</td>
<td>08c</td>
</tr>
<tr>
<td>q</td>
<td>071</td>
<td>こ</td>
<td>08d</td>
</tr>
<tr>
<td>r</td>
<td>072</td>
<td>け</td>
<td>08e</td>
</tr>
<tr>
<td>s</td>
<td>073</td>
<td>せ</td>
<td>08f</td>
</tr>
<tr>
<td>t</td>
<td>074</td>
<td>っ</td>
<td>090</td>
</tr>
<tr>
<td>u</td>
<td>075</td>
<td>ー</td>
<td>091</td>
</tr>
<tr>
<td>v</td>
<td>076</td>
<td>ア</td>
<td>092</td>
</tr>
<tr>
<td>w</td>
<td>077</td>
<td>イ</td>
<td>093</td>
</tr>
<tr>
<td>x</td>
<td>078</td>
<td>ウ</td>
<td>094</td>
</tr>
<tr>
<td>y</td>
<td>079</td>
<td>エ</td>
<td>095</td>
</tr>
<tr>
<td>z</td>
<td>07a</td>
<td>オ</td>
<td>096</td>
</tr>
<tr>
<td>(</td>
<td>07b</td>
<td>カ</td>
<td>097</td>
</tr>
<tr>
<td>:</td>
<td>07c</td>
<td>キ</td>
<td>098</td>
</tr>
<tr>
<td>)</td>
<td>07d</td>
<td>リ</td>
<td>099</td>
</tr>
<tr>
<td>~</td>
<td>07e</td>
<td>ケ</td>
<td>09a</td>
</tr>
<tr>
<td>¥</td>
<td>07f</td>
<td>コ</td>
<td>09b</td>
</tr>
<tr>
<td>.</td>
<td>080</td>
<td>サ</td>
<td>09c</td>
</tr>
<tr>
<td>r</td>
<td>081</td>
<td>シ</td>
<td>09d</td>
</tr>
<tr>
<td>j</td>
<td>082</td>
<td>ス</td>
<td>09e</td>
</tr>
<tr>
<td>.</td>
<td>083</td>
<td>セ</td>
<td>09f</td>
</tr>
<tr>
<td>.</td>
<td>084</td>
<td>ソ</td>
<td></td>
</tr>
<tr>
<td>.</td>
<td>085</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table D3-1 (continued)

<table>
<thead>
<tr>
<th>Character</th>
<th>Code-Table Entry</th>
<th>Character</th>
<th>Code-Table Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>ハ</td>
<td>0aa</td>
<td>□</td>
<td>0bb</td>
</tr>
<tr>
<td>ツ</td>
<td>0a1</td>
<td>ワ</td>
<td>0bc</td>
</tr>
<tr>
<td>チ</td>
<td>0a2</td>
<td>ヌ</td>
<td>0bd</td>
</tr>
<tr>
<td>テ</td>
<td>0a3</td>
<td>ビ</td>
<td>0be</td>
</tr>
<tr>
<td>ツ</td>
<td>0a4</td>
<td>ノ</td>
<td>0bf</td>
</tr>
<tr>
<td>ナ</td>
<td>0a5</td>
<td>ツ</td>
<td>0c0</td>
</tr>
<tr>
<td>ニ</td>
<td>0a6</td>
<td>ナ</td>
<td>0c1</td>
</tr>
<tr>
<td>ヌ</td>
<td>0a7</td>
<td>エ</td>
<td>0c2</td>
</tr>
<tr>
<td>ネ</td>
<td>0a8</td>
<td>ア</td>
<td>0c3</td>
</tr>
<tr>
<td>ノ</td>
<td>0a9</td>
<td>ア</td>
<td>0c4</td>
</tr>
<tr>
<td>ハ</td>
<td>0aa</td>
<td>ア</td>
<td>0c5</td>
</tr>
<tr>
<td>ツ</td>
<td>0ab</td>
<td>ア</td>
<td>0c6</td>
</tr>
<tr>
<td>テ</td>
<td>0ac</td>
<td>イ</td>
<td>0c7</td>
</tr>
<tr>
<td>ツ</td>
<td>0ad</td>
<td>イ</td>
<td>0c8</td>
</tr>
<tr>
<td>ニ</td>
<td>0ae</td>
<td>イ</td>
<td>0c9</td>
</tr>
<tr>
<td>ヌ</td>
<td>0af</td>
<td>イ</td>
<td>0ca</td>
</tr>
<tr>
<td>ネ</td>
<td>0b0</td>
<td>イ</td>
<td>0cb</td>
</tr>
<tr>
<td>ネ</td>
<td>0b1</td>
<td>イ</td>
<td>0cc</td>
</tr>
<tr>
<td>ナ</td>
<td>0b2</td>
<td>イ</td>
<td>0cd</td>
</tr>
<tr>
<td>ニ</td>
<td>0b3</td>
<td>イ</td>
<td>0ce</td>
</tr>
<tr>
<td>ヌ</td>
<td>0b4</td>
<td>イ</td>
<td>0cf</td>
</tr>
<tr>
<td>ネ</td>
<td>0b5</td>
<td>イ</td>
<td>0d0</td>
</tr>
<tr>
<td>ネ</td>
<td>0b6</td>
<td>イ</td>
<td>0d1</td>
</tr>
<tr>
<td>ナ</td>
<td>0b7</td>
<td>イ</td>
<td>0d2</td>
</tr>
<tr>
<td>ニ</td>
<td>0b8</td>
<td>イ</td>
<td>0d3</td>
</tr>
<tr>
<td>ヌ</td>
<td>0b9</td>
<td>イ</td>
<td></td>
</tr>
<tr>
<td>ネ</td>
<td>0ba</td>
<td>イ</td>
<td></td>
</tr>
<tr>
<td>Character</td>
<td>Code-Table Entry</td>
<td>Character</td>
<td>Code-Table Entry</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------</td>
<td>-----------</td>
<td>-----------------</td>
</tr>
<tr>
<td>ö</td>
<td>0d4</td>
<td>õ</td>
<td>0e3</td>
</tr>
<tr>
<td>ó</td>
<td>0d5</td>
<td>ŋ</td>
<td>0e4</td>
</tr>
<tr>
<td>ũ</td>
<td>0d6</td>
<td>ñ</td>
<td>0e5</td>
</tr>
<tr>
<td>û</td>
<td>0d7</td>
<td>æ</td>
<td>0e6</td>
</tr>
<tr>
<td>ų</td>
<td>0d8</td>
<td>ò</td>
<td>0e7</td>
</tr>
<tr>
<td>ð</td>
<td>0d9</td>
<td>õ</td>
<td>0e8</td>
</tr>
<tr>
<td>ŵ</td>
<td>0da</td>
<td>—</td>
<td>0e9</td>
</tr>
<tr>
<td>ç</td>
<td>0db</td>
<td>—</td>
<td>0ea</td>
</tr>
<tr>
<td>£</td>
<td>0dc</td>
<td>′</td>
<td>0eb</td>
</tr>
<tr>
<td>ø</td>
<td>0dd</td>
<td>′</td>
<td>0ec</td>
</tr>
<tr>
<td>î</td>
<td>0de</td>
<td>i</td>
<td>0ed</td>
</tr>
<tr>
<td>ť</td>
<td>0df</td>
<td>..</td>
<td>0ee</td>
</tr>
<tr>
<td>á</td>
<td>0e0</td>
<td>5</td>
<td>0ef</td>
</tr>
<tr>
<td>í</td>
<td>0e1</td>
<td>*</td>
<td>0f0 †</td>
</tr>
<tr>
<td>ó</td>
<td>0e2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

† Values 0f1–0ff are undefined.
Appendix E: Communications with the AR Division Factory

All communications with the factory of the AR Division of Telenex Corporation begin with a call to Customer Service:

Customers outside the Washington D.C. 1-800-368-3261
Greater Metropolitan Area and Virginia

In Virginia 1-703-644-9190

Local customers 644-9190

If necessary, Customer Service will direct your call to the appropriate department.

E.1 Returning an INTERVIEW or Subassemblies for Repair

(A) Authorization

1. The first step is always to call AR Division Customer Service in Springfield, Virginia.

2. Customer Service will issue a RETURN AUTHORIZATION (RA) number. This number should be posted on the outside of the package of all equipment returned for repair. The RA number, as well as a description of the problem, should be cited in all documentation, written correspondence, or telephone conversations concerning the specific repair.

WARNING: Special RA numbers are issued for customers who have purchased a Maintenance Agreement plan (or plans) from AR Division. Since these numbers identify equipment under maintenance, you must post this RA number on the outside of the package in order for AR Division to honor the terms of the Maintenance Agreement.
3. Turnaround time for repairs is usually two weeks in addition to transportation time. Customer Service can arrange to furnish a rental unit if it is not practical for you to be without the equipment for that length of time. We can either include the rental fee on the repair bill or bill the rental fee separately.

NOTE: AR Division offers expedited service Maintenance Agreement plans. Under these plans, the customer chooses between expedited repair (72-hour factory turnaround) or a loaner unit for the duration of the repair. Contact Customer Service for complete details.

(B) Shipping

1. Always include with the shipment a detailed description of the problem to be corrected. Put the assigned RA number on this document.

2. If the item is out of warranty, you should either
   a. provide a purchase order for the repair, or
   b. request an estimate of the amount of the repair.

3. Select suitable packing materials for electronic equipment containing a cathode ray tube, and pack the INTERVIEW with care. If possible, the carton and foam packing material in which you received the equipment should be used for returning it for repairs.

4. Write the return authorization number on the outside of the shipment: “ATTN RA number.”

5. International customers should address the shipment to

   Telenex Corporation, AR Division  
   ATTN RA number  
   c/o Emery Customs Brokers  
   101A Executive Drive  
   Sterling, Virginia 22170  
   U.S.A.

NOTE: For customs purposes, international customers MUST identify the country of origin (usually the U.S.A.) for returned equipment on the pro forma invoice. When returning an individual part, use the country of origin listed on the part.
6. Domestic customers should address the shipment to

   Customer Service
   Telenex Corporation
   AR Division
   ATTN RA number
   7401 Boston Boulevard
   Springfield, Virginia  22153
   U.S.A.

7. Ship PREPAID even if you have a Maintenance Agreement with AR Division. No collect shipments will be accepted unless previously authorized by Customer Service.

8. Most repairs will be completed within two weeks, not including transportation time.

E.2 Ordering Replacement Parts or Assemblies

To obtain price quotations or to order spare or replacement parts, contact Customer Service. Customer Service will need to know the model designation of the unit, its serial number and software version, and what options are installed.

E.3 PC Board or Subassembly Exchanges

The AR Division's repair replacement policy applies to the exchange of PC Boards or Subassemblies that need repair. Please contact Customer Service.

E.4 For Analysis of Problems

For applications, troubleshooting, or repair problems requiring technical assistance, call Customer Service.

E.5 Warranties

There is a standard warranty on all AR Division equipment. This warranty is for 12 months.

Extended and/or Expedited Service Agreements are available for INTERVIEW 7000 Series equipment. Operating system software maintenance is also offered. Please contact Customer Service.

E.6 Loaner Units

Loaner units are available under some hardware Maintenance Agreement plans. Contact Customer Service for additional information.
Appendix F: Packing and Shipping Instructions

The INTERVIEW is usually shipped either as baggage or as freight. The basic difference, of course, is in quantity and quality of handling to which the unit is subjected. It follows that different packing methods are called for.

When a unit is shipped as baggage, it will probably be subjected to much less severe treatment than when it is shipped by freight. The AR Division of Telenex Corporation offers its INTERVIEW Soft Pack Travel Bag, Option No. OPT-951-99-1, for this purpose. This bag has two inches of high-density foam protecting all surfaces of the INTERVIEW. It is yellow for easy identification among other luggage. An identification card case, FRAGILE markings, and leather appointments are standard features. On the outside is a large pocket for carrying notes, manuals, and so forth.

Before packing the INTERVIEW in the carrying bag, remove any diskettes from the microdiskette drives. To protect the heads during transit, insert the two yellow plastic shipping diskettes that were delivered with the unit, one in each drive. The manual should go in the front (center) pocket of the travel bag. There is an inside pocket for the power cord and other cables.

Put the INTERVIEW in the bag with its handle up (as in Figure F-1). Then close and secure the bag cover with its velcro closing.

CAUTION: The bag is considered to be reasonable protection for the INTERVIEW when it is shipped as baggage. However, it should never be used for freight shipment. The AR Division of Telenex Corporation can assume no liability for damage to units shipped this way, owing to circumstances beyond our control.

For freight shipment, the INTERVIEW should be packed in molded polyurethane foam and a heavy-duty outer cardboard carton, as delivered by AR Division. All manuals and accessories should be packed in a separate box within the carton. This packing system has been designed to give maximum reasonable protection to the INTERVIEW and ensure its safe arrival. However, damages due to mishandling must be the responsibility of the carrier.
Figure F-1  Soft Pack Travel Bag, Option 99.
For freight shipment, we also recommend the hard-shell travel case (OPT-951-95-1). See Figure F-2. This is a wheeled suitcase made of high-impact plastic, steel and rubber. It is designed for use with all AR test equipment. Because it has built-in wheels and an extension handle, the hard-shell travel case is especially useful for frequent hand-toting of the instrument.

NOTE: Please do not return any unit to the AR Division without prior authorization (see Appendix E).
Appendix G: Rack Mount (OPT-951-98-1)
Figure G-1 Rack mount for INTERVIEW 7000/7500.
Appendix G: Rack Mount (OPT-951-98-1)

A Rack Mount Kit (OPT-951-98-1) allows the INTERVIEW to be installed in a standard 19-inch wide equipment rack.

G.1 General Description

The Kit will fit either standard vertical high-boy or sloped front-panel, low-boy racks. Please note that, for proper installation, the rack must be equipped with a horizontal writing shelf.

The Rack Mount Kit offers the user slide-in/out mounting with a sloped keyboard position.

Physical specifications are as follows:

- Height: 10.5 inches
- Width: 19 inches
- Depth: 18 inches
- Weight: approximately 5.5 pounds

G.2 Installation

1. Install the rack mount into the front of the cabinet directly above the writing shelf. Secure the rack mount with the eight sets of included black panel screws (ARD #33689) and nut clips (ARD #33686).

2. Slide the INTERVIEW about three-quarters of the way into the opening. **DO NOT SLIDE THE UNIT IN FURTHER AT THIS TIME.**

3. Open the front panel and rest the keyboard on the writing shelf by sliding back the top two blue latches. At this point the hooks of the latches are exposed out the front of the unit. Press down slightly on the recessed circle of these latches and continue to slide the latches inside the unit until they stop. The indented circle should be almost centered in the sliding area and the hooks of the latches are no longer visible from the front of the unit. **These latches must be properly placed or they will lock the keyboard shut if it is accidentally closed.**

4. Carefully slide the unit into the rack mount, with the keyboard lying open, until the front blue rubber bumpers on the right side of the unit are behind the face of the rack mount. You will have approximately one inch of the unit exposed out the front of the cabinet. The INTERVIEW is now in proper position for operation.
5. Notice the rack mount has two electrical wires connecting to a switch mounted on the right front of the rack. Plug the female connector of the top wire into the power connector, located at the bottom left of the rear panel of the unit. It is a standard three-wire grounded male connector.

6. The bottom wire of the rack mount is now the power connector for your unit. Plug this male connector into a standard outlet. Check the voltage selection; see Section 1.5(B). Turn on the power switch, located on the left side of the rear panel of the unit. This permits the ON/OFF switch on the rack mount to become the power switch for your unit.

7. To complete the connections on your unit, refer to Section 1, Hardware.
Appendix H: Optional Codes JIS7/JIS8

JIS7 and JIS8 Katakana character sets are contained in files named JIS7 and JIS8 in the /sys/codes directory of DSK-951-025-1, the floppy diskette that comprises software option OPT-951-22-1. The files should be copied into the /sys/codes directory on the boot-up disk. When the unit is rebooted, the new codes will be available as Code selections on the Line Setup menu.

H.1 Accessing the Directory Containing JIS7 and JIS8 Files

Insert the disk containing the optional codes into Floppy Drive 1 (FD1). With the unit powered on and booted, press FMAINT to access the File Maintenance screen. Press CHNGDIR and FLOPPY1, then enter the following pathname in the Name field: /sys/codes. The first two lines of your File Maintenance should look like the screen in Figure H-1.

Press to access the directory containing the JIS7 and JIS8 files.

![File Maintenance Screen](image)

Figure H-1  To see the JIS7 and JIS8 files in the File Maintenance listings, you must change to the FD1/sys/codes directory.
H.2 Copying JIS7 and JIS8 Files into /sys/codes Directory

Press COPY. Leave the source pathname on the From line blank: we will make the From selections via the 8 key in the body of the current directory listings themselves. Press the 8 key to move the cursor to the To field.

On the To line, select the boot disk-drive. This may be the hard (HRD) drive; or you may install the boot-up diskette in Floppy Drive 2 (FD2). If your unit has only a single disk drive, you will use Floppy Drive 1 (FD1) to house first the source disk and then the destination (boot-up) diskette. In that case, select To: NEW.

In the Name data-entry field, type /sys/codes. Be sure to type a slash (/) both before and after the sys entry.

Now move the cursor into the directory listings. With the blinking cursor positioned over the filename JIS7, press 8. Move the cursor down over JIS8 and mark this file as well.

Your screen should resemble the screen drawn in Figure H-2. Press 8 to copy the JIS7 and JIS8 files to the /sys/codes directory on the boot disk.

If you are using a single-drive unit, prompts will “walk” you through the exchange of disks in the single drive.

![** File Maintenance **](image)

Command: COPY Push XEQ To Perform Command
From: FD1 Name: /sys/codes
To: HRD Name: /sys/codes
Current Directory: FD1/sys/codes

** Figure H-2 You may use the MARK key to select both JIS files for copying into the /sys/codes directory on the boot disk. **
H.3 Selecting JIS7 or JIS8 Code

Once the JIS files are copied into the /sys/codes directory, reboot the unit as follows: turn the unit off, wait ten seconds, then turn the power switch on again.

![Figure H-3](image)

After bootup, press SETUP, LINE, to access the Line Setup menu. Move the cursor down to the Code field. Press [ or ] to rotate the selections in this field until you have verified that JIS7 and JIS8 are available as new code selections. Figure H-3 shows a line setup with JIS7 selected in the Code field.

H.4 Testing with JIS7/JIS8

In your line setup, be sure to change Mode: AUTOMON to MONITOR or to one of the emulate modes. The Automonitor sequence will not configure the unit to run with JIS7/JIS8 code, and it will usually change the code selection to ASCII.

Figure H-4 shows a screen display for JIS7, a shifted code. Note that the messages with Katakana text begin with Shift Out (% hex @).

When you type monitor/receive strings or transmit strings into your program, the characters displayed on the trigger menus or on the Protocol Spreadsheet will always be ASCII. Use the JIS7 and JIS8 charts in Appendix E to correlate your ASCII data-entries with the actual JIS7/JIS8 characters that will be searched for or transmitted.
**MON/DISK/FD1**

**JIS7/8/NONE/BOP**

% 0

, "Hello, this is an echo test from Washington DC."

%, "Hello, this is an echo test from Washington DC."

CENT

* Figure H-4: JIS7 is a shifted code, with an upshift character (SO) preceding Katakana conversion and a downshift character (SI) preceding ASCII conversion.
Appendix I: Interface Specifications
### Table I-1
Remote Connector

<table>
<thead>
<tr>
<th>Pin No.</th>
<th>Pin Name</th>
<th>Signal Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Frame Ground</td>
<td>Ground</td>
</tr>
<tr>
<td>2</td>
<td>TD</td>
<td>RS-232/V.24 Output</td>
</tr>
<tr>
<td>3</td>
<td>RD</td>
<td>RS-232/V.24 Input</td>
</tr>
<tr>
<td>4</td>
<td>RTS</td>
<td>RS-232/V.24 Output</td>
</tr>
<tr>
<td>5</td>
<td>CTS</td>
<td>RS-232/V.24 Input</td>
</tr>
<tr>
<td>6</td>
<td>Signal Ground</td>
<td>Ground</td>
</tr>
<tr>
<td>7</td>
<td>CD</td>
<td>RS-232/V.24 Input</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>SCT</td>
<td>RS-232/V.24 Input</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>SCR</td>
<td>RS-232/V.24 Input</td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>DTR</td>
<td>RS-232/V.24 Output</td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Table I-2
**Printer Connector**

![Diagram of DB-25, male connector](image)

<table>
<thead>
<tr>
<th>Pin No.</th>
<th>Pin Name</th>
<th>Signal Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Frame Ground</td>
<td>Ground</td>
</tr>
<tr>
<td>2</td>
<td>TD</td>
<td>RS-232/V.24 Input</td>
</tr>
<tr>
<td>3</td>
<td>RD</td>
<td>RS-232/V.24 Output</td>
</tr>
<tr>
<td>4</td>
<td>RTS</td>
<td>RS-232/V.24 Input</td>
</tr>
<tr>
<td>5</td>
<td>CTS</td>
<td>RS-232/V.24 Output</td>
</tr>
<tr>
<td>6</td>
<td>DSR</td>
<td>RS-232/V.24 Output</td>
</tr>
<tr>
<td>7</td>
<td>Signal Ground</td>
<td>Ground</td>
</tr>
<tr>
<td>8</td>
<td>CD</td>
<td>RS-232/V.24 Output</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>SCT</td>
<td>RS-232/V.24 Input</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>SCR</td>
<td>RS-232/V.24 Input</td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>DTR</td>
<td>RS-232/V.24 Input</td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table I-3
Auxiliary Connector

(16-pin Bi-directional TTL Input/Output, DB-25, female)

<table>
<thead>
<tr>
<th>Pin No.</th>
<th>Pin Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PA0</td>
</tr>
<tr>
<td>2</td>
<td>PB0</td>
</tr>
<tr>
<td>3</td>
<td>PA1</td>
</tr>
<tr>
<td>4</td>
<td>PB1</td>
</tr>
<tr>
<td>5</td>
<td>PA2</td>
</tr>
<tr>
<td>6</td>
<td>PB2</td>
</tr>
<tr>
<td>7</td>
<td>PA3</td>
</tr>
<tr>
<td>8</td>
<td>PB3</td>
</tr>
<tr>
<td>9</td>
<td>PA4</td>
</tr>
<tr>
<td>10</td>
<td>PB4</td>
</tr>
<tr>
<td>11</td>
<td>PA5</td>
</tr>
<tr>
<td>12</td>
<td>PB5</td>
</tr>
<tr>
<td>13</td>
<td>PA6</td>
</tr>
<tr>
<td>14</td>
<td>PB6</td>
</tr>
<tr>
<td>15</td>
<td>PA7</td>
</tr>
<tr>
<td>16</td>
<td>PB7</td>
</tr>
<tr>
<td>17</td>
<td>Signal Ground</td>
</tr>
<tr>
<td>18</td>
<td>Reserved</td>
</tr>
<tr>
<td>19</td>
<td>Signal Ground</td>
</tr>
<tr>
<td>20</td>
<td>Reserved</td>
</tr>
<tr>
<td>21</td>
<td>Signal Ground</td>
</tr>
<tr>
<td>22</td>
<td>Reserved</td>
</tr>
<tr>
<td>23</td>
<td>Signal Ground</td>
</tr>
<tr>
<td>24</td>
<td>Reserved</td>
</tr>
<tr>
<td>25</td>
<td>Signal Ground</td>
</tr>
</tbody>
</table>
Table 1-4
RGB Monitor

<table>
<thead>
<tr>
<th>Pin No.</th>
<th>Pin Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Signal Ground</td>
</tr>
<tr>
<td>2</td>
<td>Signal Ground</td>
</tr>
<tr>
<td>3</td>
<td>Red</td>
</tr>
<tr>
<td>4</td>
<td>Green</td>
</tr>
<tr>
<td>5</td>
<td>Blue</td>
</tr>
<tr>
<td>6</td>
<td>Brightness</td>
</tr>
<tr>
<td>7</td>
<td>Reserved</td>
</tr>
<tr>
<td>8</td>
<td>Horizontal Sync</td>
</tr>
<tr>
<td>9</td>
<td>Vertical Sync</td>
</tr>
</tbody>
</table>

(DB-9, female)
RS-232/V.24 INTERFACE MODULE

Figure I-1  RS-232/V.24 Interface Module.
## Table I-5
RS-232 Test Interface Module

<table>
<thead>
<tr>
<th>Pin No.</th>
<th>Pin Name</th>
<th>Monitor Mode</th>
<th>To DCE (Em DTE) (DB-25, male)</th>
<th>To DTE (Em DCE) (DB-25, female)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Frame Ground</td>
<td>Frame Ground</td>
<td>Frame Ground</td>
<td>Frame Ground</td>
</tr>
<tr>
<td>2</td>
<td>TD</td>
<td>High Impedance Input</td>
<td>RS-232/V.24 Output</td>
<td>RS-232/V.24 Output</td>
</tr>
<tr>
<td>3</td>
<td>RD</td>
<td>High Impedance Input</td>
<td>RS-232/V.24 Input</td>
<td>RS-232/V.24 Output</td>
</tr>
<tr>
<td>4</td>
<td>RTS</td>
<td>High Impedance Input</td>
<td>RS-232/V.24 Output</td>
<td>RS-232/V.24 Output</td>
</tr>
<tr>
<td>5</td>
<td>CTS</td>
<td>High Impedance Input</td>
<td>RS-232/V.24 Input</td>
<td>RS-232/V.24 Output</td>
</tr>
<tr>
<td>6</td>
<td>DSR</td>
<td>High Impedance Input</td>
<td>RS-232/V.24 Output</td>
<td>RS-232/V.24 Output</td>
</tr>
<tr>
<td>7</td>
<td>Signal Ground</td>
<td>Signal Ground</td>
<td>Signal Ground</td>
<td>Signal Ground</td>
</tr>
<tr>
<td>8</td>
<td>CD</td>
<td>High Impedance Input</td>
<td>RS-232/V.24 Input</td>
<td>RS-232/V.24 Output</td>
</tr>
<tr>
<td>9</td>
<td>—</td>
<td>Test Point</td>
<td>Test Point</td>
<td>Test Point</td>
</tr>
<tr>
<td>10</td>
<td>—</td>
<td>Test Point</td>
<td>Test Point</td>
<td>Test Point</td>
</tr>
<tr>
<td>11</td>
<td>—</td>
<td>Test Point</td>
<td>Test Point</td>
<td>Test Point</td>
</tr>
<tr>
<td>12</td>
<td>SCD</td>
<td>Test Point</td>
<td>Test Point</td>
<td>Test Point</td>
</tr>
<tr>
<td>13</td>
<td>SCTS</td>
<td>Test Point</td>
<td>Test Point</td>
<td>Test Point</td>
</tr>
<tr>
<td>14</td>
<td>STD</td>
<td>Test Point</td>
<td>Test Point</td>
<td>Test Point</td>
</tr>
<tr>
<td>15</td>
<td>SCT</td>
<td>High Impedance Input</td>
<td>RS-232/V.24 Input</td>
<td>RS-232/V.24 Output</td>
</tr>
<tr>
<td>16</td>
<td>SRD</td>
<td>Test Point</td>
<td>Test Point</td>
<td>Test Point</td>
</tr>
<tr>
<td>17</td>
<td>SCR</td>
<td>High Impedance Input</td>
<td>RS-232/V.24 Input</td>
<td>RS-232/V.24 Output</td>
</tr>
<tr>
<td>18</td>
<td>LL</td>
<td>Test Point</td>
<td>Test Point</td>
<td>Test Point</td>
</tr>
<tr>
<td>19</td>
<td>SRTS</td>
<td>Test Point</td>
<td>Test Point</td>
<td>Test Point</td>
</tr>
<tr>
<td>20</td>
<td>DTR</td>
<td>High Impedance Input</td>
<td>RS-232/V.24 Output</td>
<td>RS-232/V.24 Input</td>
</tr>
<tr>
<td>21</td>
<td>SQ</td>
<td>Test Point</td>
<td>Test Point</td>
<td>Test Point</td>
</tr>
<tr>
<td>22</td>
<td>RI</td>
<td>Test Point</td>
<td>Test Point</td>
<td>Test Point</td>
</tr>
<tr>
<td>23</td>
<td>DSRS</td>
<td>Test Point</td>
<td>Test Point</td>
<td>Test Point</td>
</tr>
<tr>
<td>24</td>
<td>SCTE</td>
<td>High Impedance Input</td>
<td>RS-232/V.24 Output</td>
<td>RS-232/V.24 Input</td>
</tr>
<tr>
<td>25</td>
<td>TI</td>
<td>Test Point</td>
<td>Test Point</td>
<td>Test Point</td>
</tr>
</tbody>
</table>
**Figure 1-2** V.35 Interface Module.
### Table 1-6

**V.35 Test Interface Module**

<table>
<thead>
<tr>
<th>Pin No.</th>
<th>Circuit</th>
<th>Signal</th>
<th>Monitor Mode</th>
<th>To DTE (Em DCE) (34-pin, female)</th>
<th>To DCE (Em DTE) (34-pin, female)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>101</td>
<td>Frame Ground</td>
<td>Frame Ground</td>
<td>Frame Ground</td>
<td>Frame Ground</td>
</tr>
<tr>
<td>B</td>
<td>102</td>
<td>Signal Ground</td>
<td>Signal Ground</td>
<td>Signal Ground</td>
<td>Signal Ground</td>
</tr>
<tr>
<td>C</td>
<td>105</td>
<td>RTS</td>
<td>High Impedance Input</td>
<td>V.35 Input</td>
<td>V.35 Output</td>
</tr>
<tr>
<td>D</td>
<td>106</td>
<td>CTS</td>
<td>High Impedance Input</td>
<td>V.35 Output</td>
<td>V.35 Input</td>
</tr>
<tr>
<td>E</td>
<td>107</td>
<td>DSR</td>
<td>High Impedance Input</td>
<td>V.35 Output</td>
<td>V.35 Input</td>
</tr>
<tr>
<td>F</td>
<td>109</td>
<td>CD</td>
<td>High Impedance Input</td>
<td>V.35 Output</td>
<td>V.35 Input</td>
</tr>
<tr>
<td>H</td>
<td>108</td>
<td>DTR</td>
<td>High Impedance Input</td>
<td>V.35 Input</td>
<td>V.35 Output</td>
</tr>
<tr>
<td>J</td>
<td>125</td>
<td>RI</td>
<td>Test Point</td>
<td>Test Point</td>
<td>Test Point</td>
</tr>
<tr>
<td>R</td>
<td>104A</td>
<td>RD</td>
<td>High Impedance Input</td>
<td>V.35 Output</td>
<td>V.35 Input</td>
</tr>
<tr>
<td>T</td>
<td>104B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>115A</td>
<td>SCR</td>
<td>High Impedance Input</td>
<td>V.35 Output</td>
<td>V.35 Input</td>
</tr>
<tr>
<td>X</td>
<td>115B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>114A</td>
<td>SCT</td>
<td>High Impedance Input</td>
<td>V.35 Output</td>
<td>V.35 Input</td>
</tr>
<tr>
<td>AA</td>
<td>114B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>103A</td>
<td>TD</td>
<td>High Impedance Input</td>
<td>V.35 Input</td>
<td>V.35 Output</td>
</tr>
<tr>
<td>S</td>
<td>103B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>U</td>
<td>113A</td>
<td>SCTE</td>
<td>High Impedance Input</td>
<td>V.35 Input</td>
<td>V.35 Output</td>
</tr>
<tr>
<td>W</td>
<td>113B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>F1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>F1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>F2</td>
<td>Test Point</td>
<td>Test Point</td>
<td>Test Point</td>
<td>Test Point</td>
</tr>
<tr>
<td>N</td>
<td>F2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td>F3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BB</td>
<td>F3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC</td>
<td>F4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EE</td>
<td>F4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DD</td>
<td>F5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FF</td>
<td>F5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HH</td>
<td>N1</td>
<td>{ Test Point</td>
<td>Test Point</td>
<td>Test Point</td>
<td>Test Point</td>
</tr>
<tr>
<td>KK</td>
<td>N1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JJ</td>
<td>N2</td>
<td>{ Test Point</td>
<td>Test Point</td>
<td>Test Point</td>
<td>Test Point</td>
</tr>
<tr>
<td>LL</td>
<td>N2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MM</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NN</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**X.21 INTERFACE MODULE**

![Diagram of X.21 Interface Module](image)

**Figure I-3** X.21 Interface Module.

---

<table>
<thead>
<tr>
<th><strong>X.21 Interface</strong></th>
<th><strong>C</strong></th>
<th><strong>I</strong></th>
<th><strong>T</strong></th>
<th><strong>R</strong></th>
<th><strong>S</strong></th>
<th><strong>B</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>INTERFACE</strong></td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>

**Figure I-4** X.21 LED Overlay.
### Appendix I Interface Specifications

#### Table I-7
X.21 Test Interface Module

<table>
<thead>
<tr>
<th>Pin No.</th>
<th>Circuit</th>
<th>Pin ID</th>
<th>Pin Name</th>
<th>Signal Description</th>
<th>Monitor Mode</th>
<th>To DCE</th>
<th>To DTE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Shield</td>
<td></td>
<td></td>
<td>(Em DTE)</td>
<td>(Em DCE)</td>
</tr>
<tr>
<td>1</td>
<td>—</td>
<td></td>
<td>Shield</td>
<td>Frame Ground</td>
<td></td>
<td>Frame Ground</td>
<td>Frame Ground</td>
</tr>
<tr>
<td>2,9</td>
<td>103</td>
<td>T</td>
<td>Transmit Data</td>
<td>High Impedance</td>
<td>X.21 Output</td>
<td>X.21 Input</td>
<td>X.21 Input</td>
</tr>
<tr>
<td>3,10</td>
<td>105</td>
<td>C</td>
<td>Control</td>
<td>High Impedance</td>
<td>X.21 Output</td>
<td>X.21 Input</td>
<td>X.21 Input</td>
</tr>
<tr>
<td>4,11</td>
<td>104</td>
<td>R</td>
<td>Receive Data</td>
<td>High Impedance</td>
<td>X.21 Input</td>
<td>X.21 Output</td>
<td>X.21 Output</td>
</tr>
<tr>
<td>5,12</td>
<td>109</td>
<td>I</td>
<td>Indicator</td>
<td>High Impedance</td>
<td>X.21 Input</td>
<td>X.21 Output</td>
<td>X.21 Output</td>
</tr>
<tr>
<td>6,13</td>
<td>114</td>
<td>S</td>
<td>Signal Timing</td>
<td>High Impedance</td>
<td>X.21 Input</td>
<td>X.21 Output</td>
<td>X.21 Output</td>
</tr>
<tr>
<td>7,14</td>
<td>—</td>
<td>B</td>
<td>Byte Strobe</td>
<td>High Impedance</td>
<td>X.21 Input</td>
<td>X.21 Output</td>
<td>X.21 Output</td>
</tr>
<tr>
<td>15</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>Test Point</td>
<td>Test Point</td>
<td>Test Point</td>
<td>Test Point</td>
</tr>
</tbody>
</table>

**Patch Panel:** U/A A,B*

<table>
<thead>
<tr>
<th>Voltage</th>
<th>Description</th>
<th>Signal Description</th>
<th>To DCE</th>
<th>To DTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>+5V</td>
<td>Output</td>
<td>High Impedance</td>
<td>Output</td>
<td>Output</td>
</tr>
<tr>
<td>-5V</td>
<td>Output</td>
<td>High Impedance</td>
<td>Output</td>
<td>Output</td>
</tr>
<tr>
<td>GND</td>
<td>Ground</td>
<td>High Impedance</td>
<td>Ground</td>
<td>Ground</td>
</tr>
</tbody>
</table>

* UA A and B can be used for balanced or unbalanced signals. (Do not connect B when you are looking at unbalanced signals.)
Figure 1-5 RS-485 Interface Module.
## Appendix I Interface Specifications

### Table I-8

RS-485 Test Interface Module

<table>
<thead>
<tr>
<th>Pin No.</th>
<th>Circuit</th>
<th>Pin ID</th>
<th>Monitor Mode</th>
<th>15 Pin, Male</th>
<th>15 Pin, Female</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>Shield</td>
<td>Frame Ground</td>
<td>Frame Ground</td>
<td>Frame Ground</td>
</tr>
<tr>
<td>2</td>
<td>103</td>
<td>A Bus +</td>
<td>High Impedance Input</td>
<td>RS-485 Output</td>
<td>RS-485 Input</td>
</tr>
<tr>
<td>9</td>
<td>103</td>
<td>A Bus -</td>
<td>High Impedance Input</td>
<td>RS-485 Output</td>
<td>RS-485 Input</td>
</tr>
<tr>
<td>3,10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>104</td>
<td>B Bus +</td>
<td>High Impedance Input</td>
<td>RS-485 Input</td>
<td>RS-485 Output</td>
</tr>
<tr>
<td>11</td>
<td>104</td>
<td>B Bus -</td>
<td>High Impedance Input</td>
<td>RS-485 Input</td>
<td>RS-485 Output</td>
</tr>
<tr>
<td>5,12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6,13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7,14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Figure I-6 RS-449/V.36/V.37 Interface Module.

Figure I-7 RS-449/V.36/V.37 LED Overlay.
<table>
<thead>
<tr>
<th>Pin No.</th>
<th>Pin ID</th>
<th>Pin Name</th>
<th>Monitor Mode</th>
<th>To DCE (Em DTE) (36 Pin, male)</th>
<th>To DTE (Em DCE) (36 Pin, female)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>SI</td>
<td>Signalling Rate Indicator</td>
<td>Test Point</td>
<td>Test Point</td>
<td>Test Point</td>
</tr>
<tr>
<td>4,22</td>
<td>SD</td>
<td>Send Data</td>
<td>High Impedance Input</td>
<td>RS-449 Output</td>
<td>RS-449 Input</td>
</tr>
<tr>
<td>5,23</td>
<td>ST</td>
<td>Send Timing</td>
<td>High Impedance Input</td>
<td>RS-449 Input</td>
<td>RS-449 Output</td>
</tr>
<tr>
<td>6,24</td>
<td>RD</td>
<td>Receive Data</td>
<td>High Impedance Input</td>
<td>RS-449 Input</td>
<td>RS-449 Output</td>
</tr>
<tr>
<td>7,25</td>
<td>RS</td>
<td>Request to Send</td>
<td>High Impedance Input</td>
<td>RS-449 Output</td>
<td>RS-449 Output</td>
</tr>
<tr>
<td>8,26</td>
<td>RT</td>
<td>Receive Timing</td>
<td>High Impedance Input</td>
<td>RS-449 Input</td>
<td>RS-449 Output</td>
</tr>
<tr>
<td>9,27</td>
<td>CS</td>
<td>Clear to Send</td>
<td>High Impedance Input</td>
<td>Test Point</td>
<td>Test Point</td>
</tr>
<tr>
<td>10</td>
<td>LL</td>
<td>Local Loopback</td>
<td>High Impedance Input</td>
<td>Test Point</td>
<td>Test Point</td>
</tr>
<tr>
<td>11,29</td>
<td>DM</td>
<td>Data Mode</td>
<td>High Impedance Input</td>
<td>Test Point</td>
<td>Test Point</td>
</tr>
<tr>
<td>12,30</td>
<td>TR</td>
<td>Terminal Ready</td>
<td>High Impedance Input</td>
<td>Test Point</td>
<td>Test Point</td>
</tr>
<tr>
<td>13,31</td>
<td>RR</td>
<td>Receiver Ready</td>
<td>High Impedance Input</td>
<td>Test Point</td>
<td>Test Point</td>
</tr>
<tr>
<td>14</td>
<td>RL</td>
<td>Remote Loopback</td>
<td>High Impedance Input</td>
<td>Test Point</td>
<td>Test Point</td>
</tr>
<tr>
<td>15</td>
<td>IC</td>
<td>Incoming Call</td>
<td>High Impedance Input</td>
<td>Test Point</td>
<td>Test Point</td>
</tr>
<tr>
<td>16</td>
<td>SF/SR</td>
<td>Select Frequency/</td>
<td>High Impedance Input</td>
<td>Test Point</td>
<td>Test Point</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Signalling Rate Selector</td>
<td>Test Point</td>
<td>Test Point</td>
<td>Test Point</td>
</tr>
<tr>
<td>17,35</td>
<td>TT</td>
<td>Terminal Timing</td>
<td>High Impedance Input</td>
<td>RS-449 Output</td>
<td>RS-449 Input</td>
</tr>
<tr>
<td>18</td>
<td>TM</td>
<td>Test Mode</td>
<td>High Impedance Input</td>
<td>High Impedance Input</td>
<td>High Impedance Input</td>
</tr>
<tr>
<td>19</td>
<td>SG</td>
<td>Signal Ground</td>
<td>Signal Ground</td>
<td>Signal Ground</td>
<td>Signal Ground</td>
</tr>
<tr>
<td>28</td>
<td>IS</td>
<td>In Service</td>
<td>Test Point</td>
<td>Test Point</td>
<td>Test Point</td>
</tr>
<tr>
<td>32</td>
<td>SS</td>
<td>Select Standby</td>
<td>High Impedance Input</td>
<td>Test Point</td>
<td>Test Point</td>
</tr>
<tr>
<td>33</td>
<td>SQ</td>
<td>Signal Quality</td>
<td>High Impedance Input</td>
<td>Test Point</td>
<td>Test Point</td>
</tr>
<tr>
<td>36</td>
<td>SB</td>
<td>Standby Indicator</td>
<td>High Impedance Input</td>
<td>Test Point</td>
<td>Test Point</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auxilary Patch Panel</td>
<td>High Impedance Input</td>
<td>Output</td>
<td>Output</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Output</td>
<td>Output</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Output</td>
<td>Output</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Output</td>
<td>Output</td>
</tr>
<tr>
<td>19</td>
<td>GND</td>
<td>Ground</td>
<td>Signal Ground</td>
<td>Output</td>
<td>Output</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Output</td>
<td>Output</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Output</td>
<td>Output</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Output</td>
<td>Output</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AUX0 A, B</td>
<td>Output</td>
<td>Output</td>
<td>Output</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AUX1</td>
<td>Output</td>
<td>Output</td>
<td>Output</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AUX2</td>
<td>Output</td>
<td>Output</td>
<td>Output</td>
</tr>
<tr>
<td>37</td>
<td>SC</td>
<td>Send Common</td>
<td>Send Common</td>
<td>Send Common</td>
<td>Send Common</td>
</tr>
<tr>
<td>20</td>
<td>RC</td>
<td>Receive Common</td>
<td>Receive Common</td>
<td>Receive Common</td>
<td>Receive Common</td>
</tr>
<tr>
<td>34</td>
<td>NS</td>
<td>New Signal</td>
<td>Test Point</td>
<td>Test Point</td>
<td>Test Point</td>
</tr>
<tr>
<td>3,21</td>
<td>N A, B</td>
<td>National A, B</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>SHIELD</td>
<td>Shield</td>
<td>Frame Ground</td>
<td>Frame Ground</td>
<td>Frame Ground</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**AUXiliary Patch Panel:**

- **UA A, B**: Unassigned Input
  - +5 Volts
  - -5 Volts
- **GND**: Ground
- **AUX0 A, B**: Auxiliary
- **AUX1**: Auxiliary
- **AUX2**: Auxiliary
- **SC**: Send Common
- **RC**: Receive Common
- **NS**: New Signal
- **3,21 N A, B**: National A, B
- **1 SHIELD**: Shield

**Signal Description**

- Test Point: Signalling Rate Indicator
- RS-449 Output: Send Data
- RS-449 Input: Send Timing
- RS-449 Input: Receive Data
- RS-449 Output: Request to Send
- RS-449 Input: Clear to Send
- RS-449 Input: Local Loopback
- RS-449 Output: Data Mode
- RS-449 Input: Terminal Ready
- RS-449 Input: Receiver Ready
- RS-449 Output: Remote Loopback
- RS-449 Input: Select Standby
- RS-449 Input: Select Frequency Selector
- RS-449 Input: Signalling Rate Selector
- RS-449 Output: Signal Quality
- RS-449 Input: Standby Indicator
- RS-449 Input: Reserved
- RS-449 Input: Frame Ground
- Signal Ground: Test Point
- Send Common: Test Point
- Receive Common: Test Point
- New Signal: Test Point
- Reserved: Test Point
- Frame Ground: Test Point
- Output: Signal Ground
- Output: Send Common
- Output: Receive Common
- Output: New Signal
- Output: Reserved
- Output: Frame Ground
- -5 Volts: Test Point
- +5 Volts: Test Point
Figure 1-8 T1 Interface Module.
Table I-10
T-1 Test Interface Specifications

<table>
<thead>
<tr>
<th>Pin No.</th>
<th>Pin Name</th>
<th>Monitor</th>
<th>To DCE (Em DTE)</th>
<th>To DTE (Em DCE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Send Data Tip</td>
<td>High Impedance Input</td>
<td>Output</td>
<td>Input</td>
</tr>
<tr>
<td>2</td>
<td>Frame Ground</td>
<td>Ground</td>
<td>Ground</td>
<td>Ground</td>
</tr>
<tr>
<td>3</td>
<td>Receive Data Tip</td>
<td>High Impedance Input</td>
<td>Input</td>
<td>Output</td>
</tr>
<tr>
<td>5</td>
<td>Remote Test Make</td>
<td>High Impedance Input</td>
<td>Test Point</td>
<td>Test Point</td>
</tr>
<tr>
<td>6</td>
<td>Remote Test Break</td>
<td>High Impedance Input</td>
<td>Test Point</td>
<td>Test Point</td>
</tr>
<tr>
<td>9</td>
<td>Send Data Ring</td>
<td>High Impedance Input</td>
<td>Output</td>
<td>Input</td>
</tr>
<tr>
<td>11</td>
<td>Receive Data Ring</td>
<td>High Impedance Input</td>
<td>Input</td>
<td>Output</td>
</tr>
<tr>
<td>13</td>
<td>Remote Test Common</td>
<td>High Impedance Input</td>
<td>Test Point</td>
<td>Test Point</td>
</tr>
</tbody>
</table>

(1) Unlisted connectors are wired 1-for-1 through the two connectors. Test points are connected to switches and test points only.
Figure I-9 G.703 Interface Module.
Table I-11
G.703 Test Interface Specifications

<table>
<thead>
<tr>
<th>Pin No.</th>
<th>Pin Name</th>
<th>Monitor</th>
<th>To DCE (Em DTE)</th>
<th>To DTE (Em DCE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Receive Data Tip</td>
<td>High Impedence Input</td>
<td>Input</td>
<td>Output</td>
</tr>
<tr>
<td>2</td>
<td>Frame Ground</td>
<td>Ground</td>
<td>Ground</td>
<td>Ground</td>
</tr>
<tr>
<td>5</td>
<td>Send Data Tip</td>
<td>High Impedence Input</td>
<td>Output</td>
<td>Input</td>
</tr>
<tr>
<td>6</td>
<td>Receive Data Ring</td>
<td>High Impedence Input</td>
<td>Input</td>
<td>Output</td>
</tr>
<tr>
<td>9</td>
<td>Send Data Ring</td>
<td>High Impedence Input</td>
<td>Output</td>
<td>Input</td>
</tr>
</tbody>
</table>

(1) Unlisted connectors are wired 1-for-1 through the two connectors.
Figure 1-10  ISDN Interface Module.

Figure 1-11  ISDN LED overlay.
Table I-12  
ISDN Test Interface Specifications

<table>
<thead>
<tr>
<th>Pin No.</th>
<th>Pin Name</th>
<th>Monitor</th>
<th>To DCE (Em DTE)</th>
<th>To DTE (Em DCE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Send Data Tip</td>
<td>High Impedence Input</td>
<td>Output</td>
<td>Input</td>
</tr>
<tr>
<td>4</td>
<td>Receive Data Tip</td>
<td>High Impedence Input</td>
<td>Input</td>
<td>Output</td>
</tr>
<tr>
<td>5</td>
<td>Receive Data Ring</td>
<td>High Impedence Input</td>
<td>Input</td>
<td>Output</td>
</tr>
<tr>
<td>6</td>
<td>Send Data Ring</td>
<td>High Impedence Input</td>
<td>Output</td>
<td>Input</td>
</tr>
<tr>
<td>7&lt;sup&gt;2&lt;/sup&gt;</td>
<td>- voltage</td>
<td>Output</td>
<td>Output</td>
<td>Output</td>
</tr>
<tr>
<td>8&lt;sup&gt;2&lt;/sup&gt;</td>
<td>+ voltage</td>
<td>Output</td>
<td>Output</td>
<td>Output</td>
</tr>
</tbody>
</table>

(1) Unlisted connectors are wired 1-for-1 through the two connectors.
(2) Pins 7 and 8 have a voltage differential of 400 volts; see ISO 8877 (1987-08-15) and CCITT I.430.
Appendix J: Field Service on the INTERVIEW 7000 Series

This appendix is to guide you in proper removal, handling, and installation of logic boards and other components in the INTERVIEW 7000 Series.

- J1 alerts you to the problem of static electricity.
- J2 covers the removal of logic cards.
- J3 discusses the installation of logic cards.
- J4 covers the installation of the optional multiplexer boards.
- J5 discusses the replacement of firmware on the CPM board.
- J6 covers the installation of a hard disk drive in the INTERVIEW 7000 and 7200 TURBO, option OPT-951-01-1.
- J7 covers other components requiring attention.
Figure J1-1 Illustration of grounded workstation. Note grounded wrist strap and table mat.
Appendix J1: Eliminating Static Electricity

STATIC ELECTRICITY CAN DAMAGE THE UNIT WHEN THE COVER IS REMOVED. Before you begin to remove the cover, be certain you have taken appropriate anti-static precautions.

J1.1 Take Precautions

As a *minimum* precaution, a grounded wrist strap should be used in conjunction with an anti-static work surface mat. Without these precautions, walking (or just shifting your feet) on a carpet or tile floor, shifting your position in a chair, or simply rolling a chair as little as a foot or two can generate sufficient static electricity to damage circuitry. See Figure J1-1 for an example of a static-free workstation for working with sensitive parts and assemblies.

Place unit on anti-static mat with power OFF. Then put on a grounded wrist strap and proceed.

J1.2 Use of Anti-Static Packing

When returning any boards to the AR Division factory, reuse any bags and anti-static packaging from boards sent by the AR Division previously. Pack the parts tightly to prevent motion which could generate static. If you did not save the packaging, you *must* obtain special anti-static packing before you begin or you will damage the components on the board.
Appendix J2: Removing Logic Boards
Figure J2-1 Viewing interior components of the INTERVIEW 7000 Series.
Appendix J2: Removing Logic Boards

CAUTION: STATIC ELECTRICITY CAN DAMAGE THE UNIT WHEN THE COVER IS REMOVED. Before you begin this section, you must take the proper anti-static precautions given in Appendix J1.

J2.1 Remove the Cover

To remove the cover of your unit, first unplug all connecting cords on the back of the INTERVIEW. Place the unit on its back and remove the six long screws recessed in the base.

Place the unit in its working position, lying on its base. Open the front panel, sliding the top two blue latches back. At this point the hooks of the latches are exposed out the front of the unit. Press down slightly on the recessed circle of these latches and continue to slide the latches inside the unit until they stop. The indented circle should be almost centered in the sliding area and the hooks of the latches are no longer visible from the front of the unit. These latches must be properly placed or they will break when you remove the cover.

Keeping an eye on the position of the latches, grasp the sides of the cover and remove it. The handle on the left side and rubber bumper feet on the right side can help to give you some leverage. You may have to rock the cover slightly back and forth to free it. Replace the handle in its slot on the base.

J2.2 View the Interior Layout

(A) Anticipate Potential Hazards

The inside of your INTERVIEW reveals three potentially hazardous areas. DO NOT TOUCH THESE AREAS OR INJURY MAY RESULT! These three danger areas include the mechanical fan at the center back of the unit (physical hazard), the power supply at the back right side of the unit (230V shock hazard), and the area directly behind the LED display (190V shock hazard). See Figure J2-1 for their locations.
Take care as you remove the boards not to make contact with any of these three areas.

(B) View Interior Components of the INTERVIEW 7000 and 7200 TURBO

In addition to these three potentially hazardous components, on the left near the front of your unit is a bracket for a Winchester hard disk drive. The standard INTERVIEW 7000 does not have a hard drive, but the bracket to hold it is in position for the option of adding a hard drive. If your INTERVIEW 7000 has been upgraded with this option, you will have two cables running up the side of your hard drive and over it, connecting it to the PCM board.

Immediately behind the hard disk drive is the TIM (Test Interface Module) holder. The dual floppy disk drives are on the right near the front, with a cable connecting them to the PCM board. In the center of the unit are the nine slots which house your boards.

Reading from left to right, the slots contain the following boards:

- slot 1—empty (unless upgraded with optional multiplexer board)
- slot 2—GBM, Global Bus Module Board
- slot 3—CPM, 68K Processor Board
- slot 4—PCM, 68K Peripheral Board
- slot 5—FEB, Front End Buffer Board
- slot 6—reserved for XDRAM board (OPT-951-23-1)
- slot 7—empty (unless upgraded with optional MPM board)
- slot 8—empty (unless upgraded with optional MPM board)
- slot 9—MPM, 286 Processor Board

There is one cable connecting the LED display to the GBM board. Additionally, there are two jumper cables; one connecting the FEB and GBM boards and a smaller one connecting the CPM and PCM boards.

(C) View Interior Components of the INTERVIEW 7500 and 7700 TURBO

In addition to these three potentially hazardous components, on the left near the front of your unit is the Winchester hard disk drive. It has two cables running up its side and over it, connecting it to the PCM board.

Immediately behind the hard disk drive is the TIM (Test Interface Module) holder. The dual floppy disk drives are on the right near the front, with a cable connecting them to the PCM board. In the center of the unit are the nine slots which house your boards.
Reading from left to right, the slots contain the following boards:

- slot 1—empty (unless upgraded with optional multiplexer board)
- slot 2—GBM, Global Bus Module Board
- slot 3—CPM, 68K Processor Board
- slot 4—PCM, 68K Peripheral Board
- slot 5—FEB, Front End Buffer Board
- slot 6—reserved for XDRAM board (OPT-951-23-1)
- slot 7—MPM, 286 Processor Board
- slot 8—MPM, 286 Processor Board
- slot 9—MPM, 286 Processor Board

There is one cable connecting the LED display to the GBM board. Additionally, there are two jumper cables; one connecting the FEB and GBM boards and a smaller one connecting the CPM and PCM boards.

J2.3 Remove the Boards

As stated previously in Section J1.2, if you have bags and packaging from boards sent by the AR Division previously, reuse those anti-static materials for packing. If you did not save the packaging, you must obtain a special anti-static packing before you begin or you will damage the components on the board.

(A) Disconnect the Cables

Make certain the unit is on an anti-static mat and you are wearing your grounded wrist strap.

You may wish to record where each cable was attached for reference when your replacement boards arrive. Refer back to Figure J2-1. Your upgraded replacement boards will be sent as soon as possible.

Notice that several of the connectors are “keyed” for easy, correct alignment. Those that are not “keyed” need to have special care taken to line up the pins with the proper connecting holes when they are reconnected.

Disconnect the cables carefully. They may be easily removed with an IC clip holder. Lift the connector straight up, holding onto its edges. DO NOT PULL ON THE CABLE; you could break it or damage the connectors. If you do not have an IC clip holder, a blade screwdriver will also work.

CAUTION: If you use a screwdriver, use the top of the black mounting rack (which secures the ends of the boards) as your leverage point. Avoid using the edges of the boards or other components to pry the connector loose.
(B) Remove the Boards

CAUTION: Do not hold any board by its gold edge connector. Hold it by the sides, NEVER touching the components.

It is probably easiest to remove the boards from left to right. To do so, grasp the board by its top corner edges, and gently pull it straight up and out. Then, holding the board by its side edges, place it immediately into an anti-static bag and close the bag. Repeat the process with each board to be removed.

J2.4 Replace the Cover

If you are not going to replace any boards at this time, continue with this section to replace the cover to protect the remaining components. If you are ready to replace boards, proceed down to Section J2.5, Package the Boards, and then on to Section J3, Installing Logic Boards.

Grasp the cover with the latch area to the front facing you. Make certain the latches are recessed as far as they can go into the cover and their hooks are not beyond the edge of the cover. Slide the cover down. For the INTERVIEW 7500 and 7700 TURBO—and those INTERVIEW 7000s and 7200 TURBOs having the optional hard drive—take care that the two cables to the Winchester hard disk drive (on your left, directly behind the handle) are not being pinched or damaged by the cover. (Some models may contain a small, removable, protective plastic sheet to help shield the cables from the cover.)

Place the unit on its back once again and replace the six screws to hold the cover secure.

J2.5 Package the Boards

In preparation for storage or shipping, package each board removed from the INTERVIEW in its own anti-static bag. If shipping, wrap it further in anti-static packing material and place it securely in a shipping box. You may package several boards in a single box, as long as each board is wrapped individually in an anti-static bag. When returning a board, please reference the return authorization number from your original packing list. For shipping information, see Appendix E, Communications with the AR Division Factory.
Appendix J3: Installing Logic Boards
Figure J3-1  Some connectors are "keyed" for correct alignment.
Appendix J3: Installing Logic Boards

Observe the proper handling of the boards: refer to Appendix J1 and J2 for using proper anti-static precautions and removing the boards.

Figure J2-1 gave you visual locations of the components in the INTERVIEW 7000 Series. For a detailed listing of board placement in the slots, see Section J2.2. Each board is labeled in its upper left corner. The motherboard (on the floor of the unit) is labeled with the corresponding name of the board by the slot into which that particular board is set.

J3.1 INTERVIEW 7000 Series Hardware Architecture

In troubleshooting any malfunction, it is important to know which board contains the component that controls the particular function.

The INTERVIEW contains six types of board which are connected to the motherboard and which can be easily removed and replaced. (See Appendix J2 and J4 for board removal information.) These boards are:

- MUX Multiplexer Board (if upgraded with option)
- GBM Global Bus Module
- CPM Control Processor Module
- PCM Peripheral Control Module
- FEB Front-End Buffer
- MPM Main Processor Module

A seventh type of board is installed in the Test Interface Module.

Figure J3-2 is a block diagram showing the components on all of these boards. The figure also shows how the components are interconnected in the unit. Following the figure is a descriptive listing of the components.

NOTE: The symbol in the diagram represents a bi-directional buffer.
Figure J3-2  Block diagram of INTERVIEW 7000 Series hardware architecture.
(A) MUX (Multiplexer Board)

This board provides whatever additional processing is needed for special data formats such as T1, G.703, ISDN, etc.

The mux board clocks-in data from transmitter channel A on the GBM. Once at the mux board, the data is processed as required and then sent back to the GBM to be transmitted over the line.

(B) GBM (Global Bus Module)

1. *Mux*. The muxes multiplex various signals to be transmitted, received, or monitored depending upon which mode of operation is being used.

2. *Transmitter channel A*. The transmitter generates the data stream to be sent out when the INTERVIEW is in emulate DTE mode. The data can either be sent out directly to the TIM or sent to the mux board for further processing.

3. *Beep control*. The beep control does just that—it controls the beep. There are two types of beeps defined by their length—one second and one-half second. Only the half-second beep is currently in use.

4. *Baud rate generator*. The baud rate generator generates the clock for the transmit data when in emulate DTE mode.

5. *Gbus arbitor control*. This arbitrates requests by the GBM to obtain the global bus. When the bus is idle and the GBM has top priority over all other requests, this logic will enable the GBM to access the global bus.

6. *Power on reset*. This circuitry generates the proper timing for reset signals for the INTERVIEW unit from a power-up situation.

7. *10 MHz clock generator*. This generates the global bus clock.

8. *Lead control port*. The INTERVIEW writes to this port in order to send out the proper control signals for the data interface. These control signals are dependent upon the emulate mode for which the unit is configured.

9. *Lead sensor port*. The INTERVIEW reads the lead sensor port in order to monitor incoming control signals from the data interface.

10. *TIM control port*. The INTERVIEW writes to this port in order to setup the TIM to the desired mode of operation.

11. *Real time clock*. This circuitry keeps track of current time and date. This information can be displayed on the screen for the user and can also be used to timestamp the incoming data.

12. *Transmitter channel B*. This is a second transmit channel used to send data out into the data stream through the mux board.
13. **Mux control port.** The INTERVIEW software writes to this port to set up the muxes for the proper data flow through the GBM, depending upon the mode of operation.

14. **2 channel DMA.** In order for the INTERVIEW to transmit data in emulate mode, the DMA controller requests a byte of data from memory on an MPM. When this byte is obtained, the DMA controller then sends it out through one of the transmitter channels.

(C) **CPM (Control Processor Module)**

The CPM board controls most of the operations of the INTERVIEW unit.

1. **Disk control circuitry.** The disk controller is the 9580B. In record mode, the disk controller sends a block of data through the PCM to one of the disk drives. When in playback, the disk controller reads one of the drives through the PCM so that the CPM can send recorded receive data to the FEB. The 9580 controls up to three disk drives—one Winchester hard drive and two floppy drives.

2. **8 MHz clock generator.** This circuitry generates the clock that is used by the 68010 CPU and the peripherals on the CPM.

3. **Global bus interrupt port.** This port can be written to by any board that has access to the global bus. There are four global interrupts that can be set, each of which will cause an interrupt of the 68010 processor. Thus, any board that is connected to the global bus has the ability to interrupt the 68010.

4. **Global bus arbitor.** This circuitry queues requests by the CPM for access to the global bus and waits until priority is obtained before enabling the CPM onto the global bus. Control bits are set to determine the mode of operation of the arbitor. One mode causes the CPM to always release the bus after an access. Another mode causes the CPM to hold onto the global bus until it is requested by another board.

5. **4 channel DMA.** The CPM currently uses only two of the available four DMA channels. Channel zero is used for recording data and channel one is used for playback of data. When recording data, the FEB interrupts the CPM to signify that a byte of receive data is available. The DMA then reads the byte from the FEB into memory on the CPM. When enough bytes have been accumulated to form a block, the disk controller will send the block to the specified disk drive. When in playback mode, the DMA sends out bytes of receive data to the FEB. This is data that has been moved into CPM memory by the disk controller. In addition, the DMA circuitry allows for any memory-to-memory transfers necessary between the CPM and other boards.

6. **Software reset logic.** The CPM board contains logic which enables the 68010 to generate a software reset to the rest of the unit. The reset signal is
generated by executing a RESET instruction in the 68010. This will cause
the software reset to be generated without disturbing any of the registers
within the 68010.

7. _Interrupt controllers_. The three interrupt controllers monitor interrupts from
the disk controller, time-out logic, and global bus. These interrupt
controllers generate the appropriate vectors for the CPU’s interrupt vector
table.

8. _Channel timer_. This function is carried out by the 9513A, a chip used to
generate output signals that are used as CPU interrupts. The time intervals
for these interrupts are user-programmable.

9. _32 bit counter_. This circuitry counts a 1 MHz clock. The CPU can latch
this value at any time and then read the lower 16 bits of the count, as well
as the upper 16 bits if they are needed. The 32 (or less) -bit count is used
by the software to determine specific time intervals necessary for execution
of certain routines.

10. _EPROM_. The EPROM contains the power-up software and initialization
routines. When the power is turned on, the 68010 processor begins
execution by fetching from the EPROM.

11. _Wait/timer logic_. This logic generates an error interrupt to the CPU if an
instruction causes the board to hang up. If the processor does not receive an
acknowledgment within approximately .33 milliseconds after it has begun a
cycle, an interrupt will be generated.

12. _P bus arbitor_. The P bus (CPM bus on block diagram) connects the CPM
directly to the PCM and the FEB. Thus, the CPM can directly read and
write to ports on any of these boards without accessing the global bus. The
arbitor is simply a pair of pals which decode the address from the CPU. If
the address is one of the ports on the PCM or FEB, and an
acknowledgment has been received from the appropriate board, the data
buffers are enabled.

13. _DRAM_. The DRAM is 2 Mbyte of memory space that contains the
operating software for the INTERVIEW unit. At power up, this software is
loaded into the DRAM from either the hard or floppy disks. In addition,
any programs that are compiled for run mode are stored in the DRAM.

14. _68010 processor_. This processor is the brain of the INTERVIEW unit. It
controls virtually every operation of the box. The 68010 processor operates
using the software stored in the DRAM on the CPM.
(D) PCM (Peripheral Control Module)

The PCM board provides the interface to all the major peripherals in the INTERVIEW unit, as well as external peripherals attached to the unit.

1. **Dotclock generator.** The dotclock is generated by buffering a 23.9114 MHz oscillator and then dividing it by two. This gives a dotclock frequency of 11.9557 MHz.

2. **Aux controller.** The aux control function is performed by the 8536 chip on the PCM. (This chip is also known as a CIO.) This part enables the unit to read or write to other nodes through a DB-25 connector located on the back of the unit. The 8536 has two 8–bit ports, and each line may be configured as an input or an output. When configured as an input, the 8536 can be programmed to look for a specific condition on one bit or search for an entire word, and then generate an interrupt.

3. **Remote/printer controller.** The control functions for the remote port and the printer port are performed by the 8530A chip on the PCM. (This chip is also known as the SIO.) The SIO provides a serial interface between the INTERVIEW unit and other nodes. It receives serial data from external sources, strips off any flags or block-check characters, and causes an interrupt when it has an 8–bit word of data to be read out of it. On the transmit side, the SIO takes a word of data from the processor on the CPM, adds the necessary flags and block-check characters, and sends out the data over the serial interface.

   The SIO has two available ports on the PCM—one is used for the printer interface and the other is used for the remote interface. For the printer interface, the SIO acts as a DCE. It accepts TD as an input and generates RD as an output. It also generates the appropriate RS–232 handshake control signals. For the remote interface, the SIO acts as a DTE, sending out TD data to the remote port and receiving RD data from the remote port. The remote port is used to control the INTERVIEW unit from a remote terminal instead of from the keyboard.

4. **Display timing controller.** The display timing control functions are performed by the Signetics 2674 chip. During initialization, the CPM programs this chip with information about the display device, including the size of each character in dots, the number of characters per line, the number of lines per screen, and the size of the horizontal and vertical blanking intervals and sync pulses. Also programmed into the chip is the initial location and block size of the character data block in RAM. When the 2674 runs, it automatically generates the proper timing of character data, blanking intervals, and sync pulses. In addition, it also generates control signals which allow for overline, underline, and strikethrough.

5. **Attribute RAM and mux.** The attribute RAM takes a 12 bit address and generates 16 bits of output which define all attributes of a particular
character. The 12 bit address can come from two possible sources. When the display is scanning, data is being read from the attribute RAM and the address comes from a buffer attached to the output of the display controller. When writing to the display, data is being written into the attribute RAM and the address comes from a latch which stores the desired write location as output from the processor on the CPM. Attributes of a character include underline, overline, strikethrough, reverse image, blinking, blanking, low intensity (monochrome only), background color, foreground color, and data type (hex or ASCII).

6. Data RAM. The data RAM takes the same 12 bit address as the attribute RAM and generates 12 bits of output which provide character mapping information. The address is multiplexed from either the display controller (read) or the address latch (write) as described for the attribute RAM. One bit of each output word is dedicated as a flag to designate that a particular character may not be displayed in hex format. (These are usually flags, block-check, or control characters.) Three bits of the output represent 8 possible character formats (i.e. ASCII, EBCDIC, hex, etc.) The other eight bits represent 256 possible characters within each character format.

7. Mapping RAM and mux. The mapping RAM takes a 10 bit address and generates 10 bits of output which provide the location of a specific character dot pattern in the font RAM. The 10 bit address is multiplexed from two sources. When the display is scanning (read), the address comes from the output of the data RAM. When writing mapping information to the mapping RAM, the address comes from a latch which stores the desired write address from the processor on the CPM. The address input defines a specific character and a specific data format. The mapping RAM takes this information and generates an output which points to the location in font RAM where the actual dot pattern is stored.

8. Font RAM and mux. The font RAM takes a 14 bit address and generates 9 bits of actual display dot information. The 14 bit address is multiplexed from two sources. When the display is scanning (read), the lower 4 address bits come from a latch connected to the display controller chip and the upper 10 bits come from the mapping RAM. When writing font information to the font RAM, the address comes from a latch which stores the desired write address from the processor on the CPM. Each character on the screen is 9 dots wide by 12 dots high. The lower 4 address bits decode the specific row within the character and the upper 10 bits decode to a possible 1024 actual character dot patterns. The output is a 9–bit word which gives the actual on–off pattern of dots in a particular row of a given character.

9. Overline, strikethrough and latch. This circuitry actually generates the strikethrough and overline attributes by overriding dot information from the font RAM in specific rows of a character. Finally, the resulting dot data is
latched and processed by the video attribute controllers. (Underline is done internally by the attribute controller.)

10. **Monochrome attributes controller.** This function is performed by a 2675 that has been programmed to operate in monochrome mode. The 2675 creates a character clock from the dot clock, such that one character clock is generated for every nine dot clocks. (Each character is nine dots wide.) For each character to be output to the video monitor, the 2675 takes the nine bits of processed dot data, along with control signals for blanking, blinking, underlining, cursor position, and reverse imaging, and creates the serial stream of dot data called monochrome video. This data stream is then sent on through a buffer to the plasma display or through driving transistors to the RS–170 port that can be connected to a CRT.

11. **Color attributes controller.** This function is performed by a 2675 that has been programmed to operate in color mode. For each character to be output to a color monitor, the 2675 takes the nine bits of processed dot data—along with control signals for blanking, blinking, underlining, cursor position, foreground color, and background color—and creates four serial streams of data which will generate the color video. The four different data streams represent red, blue, green, and luminescence. This data passes through a buffer before being sent out to the color monitor through a port in the rear of the unit.

12. **Keyboard controller.** The keyboard controller is an 8051 microcontroller programmed to function like an INTEL 8278. (The 8278 is a keyboard controller that was used until it was obsoleted by the manufacturer.) The controller continually writes out a sequence of addresses that are decoded to represent rows and columns of the main keyboard and function keyboard. The scanning process continues until the controller finds a key depress or key release condition. The controller then interrupts the 68010 processor on the CPM which reads the key value out of the controller.

13. **Key encoder.** The key encoder is simply a combination of buffers and decoders which take the scan address from the keyboard controller and convert it into row and column select signals that go directly to the main keyboard and function keyboard. Each scan address decodes to a unique row and column combination on one of the keyboards.

14. **Interrupt control.** The interrupt control logic combines interrupts from several sources into one interrupt signal that is sent to the CPM board. The CPM then reads a 3-bit code generated by the interrupt logic to determine the source of the interrupt on the PCM. Interrupts from the PCM are caused by key press, key release, break detect on receive data, or interrupts from the 8536 chip (aux controller).

15. **Phase locked loop/disk control logic.** The disk control logic simply takes the disk control signals from the CPM and sends them out to the selected drive,
either floppy or hard. It also buffers control signals from the disk drives and sends them back to the disk controller on the CPM. The 9582 data disk separator recovers clock from a stream of data coming from a disk drive and sends it back to the CPM. The 9582 separator has a built-in phase-locked loop that is used for this clock recovery.

16. Mode control. This is simply a latch that is loaded from the CPM. The various bits of this latch control certain aspects of the display. By programming certain bits of this latch, one can force the display to hex format, control the cursor, choose half or full duplex, or program the display controller for operation from a 50 Hz power supply.

(E) FEB (Front-End Buffer)

The Front End Buffer provides the necessary pre-processing for data in the receive path. There are three paths that the data may take through the FEB. In run mode, raw data passes through the FEB to the receiver and gets read by an MPM. The data can also be stored in encoded form on one of the disk drives by the disk controller on the CPM. In playback mode, data does not come from an external device, but from a selected disk drive. In this mode, data from the disk controller is multiplexed into the receive path.

1. Idle suppress. When selected, this circuitry removes the idle characters from the receive data stream.

2. Data encoder. This circuitry takes the receive data and encodes it in a format that enables easy storage. The encoded format includes data and control lead information along with time ticks.

3. 1024B fifo (small). The small fifo ("first in, first out") stores the encoded data until it can be read by the DMA controller on the CPM board. This path is used when the INTERVIEW is in record mode.

4. Fifo/mux control. The CPM writes to this port to set the appropriate control bits for the fifos and to configure the mux for the proper mode of operation.

5. Mux. The mux multiplexes between real-time data and playback data in the receive path.

6. 64K fifo (big). The big fifo acts as a "rubber band" between the FEB and the MPM—it "stretches or shrinks" with the amount of data received and sends it out at the proper rate. The FEB puts data into the fifo as it receives it and waits for the receiver on the FEB to request a byte.

7. Data decoder. The data decoder puts the data back into raw form as it appeared before entering the data encoder. The raw data is then passed on to the receiver to be sent out to the MPM. Control lead information is also output from the decoder and sent to the lead port.
8. **Lead port.** The MPM reads this port on the GBM to obtain information on the status of the control leads.

9. **Encoder/decoder control.** The MPM writes to this port on the GBM to set the control bits for the data encoder and data decoder on the board.

10. **Receiver.** When a byte of data has been processed through the decoder, it passes on to the receiver circuitry. The MPM continuously polls the receiver to check for the presence of a byte of data, and if there is data present, the MPM reads it from the receiver into its memory.

**(F) MPM (Main Processor Module)**

The MPM does all the higher level processing of the receive data. This board also generates the transmit data to be sent out when in emulate mode. If the XDRAM board is present, see Table J3-1 for proper switch settings for switch S1 on the MPM's for specific allocations of memory space.

1. **Clock generator.** This circuitry generates the clock that runs the 80286 CPU and peripherals on this board.

2. **Global bus arbitor control.** This circuitry queues requests for the global bus. It then grants access to the global bus when the global bus is idle and this board has the highest priority of all requesting boards. This function is performed by the bus arbitor chip.

3. **Bank address register.** The CPU writes to this register to set the upper address bits for the 80286 when it is operating in real mode. In real mode, the CPU does not drive address bits 19 through 23, so these bits must be set in this register if the CPU is running in real mode.

4. **Address buffer latch.** This circuitry simply latches the address bits from the processor in order to guarantee that the address is stable throughout the entire CPU cycle.

5. **Mux.** This multiplexer simply chooses whether to take the upper address bits directly from the processor, or from the bank address register described above. The register is used when the CPU is in real mode, because the CPU will only drive these bits in protected mode.

6. **80286 CPU.** This processor controls the operation of the MPM. The processor operates on software located in the DRAM on the MPM. This software is compiled by the CPM and loaded into the MPM. The software will tell the MPM how to process the data, and what trigger conditions to look for in the data stream. The CPM continually polls the MPM to see if data is available to be sent to any of the user interfaces: printer, plasma display, and remote port.

7. **DRAM.** The DRAM on each MPM is dedicated to storage of receive data. In addition, the DRAM contains the operating software for the MPM. This
memory is dual-ported; this means that the DRAM can be accessed by either the local MPM bus from the 80286 processor or by the global bus. Operation of the DRAM and arbitration of requests from the two ports is controlled by the DRAM controller chip. This chip also generates refreshes for the DRAM at the proper intervals.

8. **Wait/timer logic.** This circuitry generates the proper amount of wait states for some local cycles. It also monitors the operation of the MPM to check for a condition where the CPU gets hung up. The timer logic will generate a non-maskable interrupt to the 80286 processor if a cycle begins and no acknowledge is received within approximately 7 milliseconds.

9. **Reset control logic.** The MPM can be reset in several ways. A global software reset will cause the MPM to go into a reset state. The CPM may also cause a reset on the MPM by writing to a specific global port. Once the MPM has been reset, it will remain in the reset state until the reset is released by the CPM writing to another dedicated global port. Thus, when the unit is first powered up, the CPM can execute its initialization routines, and load the MPM software into the MPM DRAM while the MPM is still in the reset state.

10. **32 bit counter.** The 32 bit counter is made up of four 74LS590’s in series which count a 1 MHz clock. These counters are used to determine elapsed time between certain events in the data stream. The count is latched by writing to a dedicated local port. The count is read 16 bits at a time, by reading one of two local ports. The count may be cleared to zero by writing to another dedicated local port.

11. **5 channel timer.** The 5 channel timer is the 9513A, a chip that uses six clock sources as inputs. The user programs the chip to count one of the clock sources and generate an output when the terminal count is reached. The output can be a high or low pulse, or simply a toggle of the output line. Each condition may be executed once or repeatedly. There are five separate outputs, one for each channel. Each channel is independent and may be programmed differently. The 9513A is used on the MPM to generate "timeout" interrupts at specific intervals as required by the user program.

12. **Interrupt control.** The interrupt control circuitry on the MPM consists of three 8259A chips cascaded to allow for a maximum of 22 interrupts to the 80286 processor. Five interrupts come from the 9513A timer logic. Ten interrupts come directly from the global bus. These global interrupts are initiated by peripherals on the global bus. Four other interrupts are software driven interrupts that are caused by another processor writing to dedicated memory–mapped I/O ports on the MPM. Two interrupts come from the FEB, but only one is used.
(G) TIM (Test Interface Module)

The test interface module (TIM) is the interface between the external data path and the INTERVIEW unit. Data enters the unit at the place marked DTE and exits the unit at the place marked DCE.

1. **Mode control.** The mode control circuitry determines the operating mode of the unit. There are three operating modes: monitor, emulate DTE, and emulate DCE. When in monitor mode, the mode control circuitry configures the relays on the TIM so that data may pass through from the DTE to the DCE while being monitored by the INTERVIEW unit. When in emulate DTE mode, the mode control circuitry configures the relays so that the connection to the DTE is broken and data generated by the INTERVIEW unit is transmitted out to the DCE. When in emulate DCE mode, the mode control circuitry configures the relays so that data generated by the DTE is received by the INTERVIEW unit and the connection to the DCE is broken. In addition, when in either emulate mode, the mode control circuitry sets the relays to enable the INTERVIEW to send or receive the appropriate control signals depending on the emulate mode selected.

2. **Patch panel.** The patch panel is simply a row of switches and headers which enable the user to selectively connect and break individual signals on the back of the test interface module.

**J3.2 Install a Board**

Remove the new board from its static-proof bag. Remember to handle the board by its edges, not touching its components or the gold edge-connector. Grasping the board by the two top corners, slide it gently into the correct slot, seating it properly in the connector.

Check the settings for switch S1 on each MPM board to make certain that they are correct. Check that the proper MPM board will be in its correct slot. For the INTERVIEW 7500 and 7700 TURBO they should read as shown in Table J3-1.

For the INTERVIEW 7000 and 7200 TURBO, place your MPM board in slot 9, with the settings the same as those referenced for that board in Table J3-1 also.
Table J3-1
S1-Switch Settings for MPM Boards in the INTERVIEW 7000 Series

<table>
<thead>
<tr>
<th>MPM</th>
<th>Slot 7</th>
<th>Slot 8</th>
<th>Slot 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pin</td>
<td>1 2 3</td>
<td>1 2 3</td>
<td>1 2 3</td>
</tr>
</tbody>
</table>

Units with 3 MPMs:

Normal†  | ON OFF ON | OFF ON ON | ON ON ON |
High-Speed†† | OFF OFF ON | ON OFF ON | ON ON ON |

Units with 2 MPMs:

Normal†  | OFF ON ON | ON ON ON |
High-Speed†† | OFF OFF ON | ON ON ON |

Units with 1 MPM:

Normal†  | ON ON ON |
High-Speed†† | ON ON ON |

† These are the normal MPM switch settings. Changing these settings will affect object program compatibility with other units.

†† These MPM switch settings maximize the size of high-speed record RAM when the file /sys/xdram_rord resides on the boot-up disk during power-up.

J3.3 Reconnect the Cables

Reconnect the cables in the reverse order that you removed them. Remember, the smallest jumper cable connects the CPM and PCM boards and the larger jumper cable connects the FEB and GBM boards. Of the other cables, one connects the LED display to the GBM board and another connects the dual floppy disk drives to the PCM board. Additionally, there are two cables connecting the Winchester hard drive to the PCM board in the INTERVIEW 7500 and 7700 TURBO (and those INTERVIEW 7000s and 7200 TURBOs containing the optional hard drive).

Recall that several of the connectors are "keyed" for easy, correct alignment. Those that are not "keyed" need to have special care taken to line up the pins with the proper connecting holes. See Figure J3-1.

J3.4 Test the Unit

Before you completely secure the cover, it is suggested that you test your unit to make certain that it functions after this board exchange. First, replace the cover.
Grasp the cover with the latch area to the front facing you. Make certain the latches are recessed as far as they can go into the cover and their hooks are not beyond the edge of the cover. Slide the cover down. For the INTERVIEW 7500 and 7700 TURBO—and those INTERVIEW 7000s and 7200 TURBOS with the optional hard drive—take care that the two cables to the Winchester hard disk drive (on your left, directly behind the handle) are not being pinched or damaged by the cover. (Some models may contain a small, removable, protective plastic sheet to help shield the cables from the cover.)

Now you can safely test the unit. For this test you must power the unit. Reconnect your power cable, turn on the unit, and check for the system self-tests.

The words “System RAM test now accessing all RAM :CPMOMIMIM2” appear on your screen, indicating that the first self test for the system is being performed. Following this first test are the tests for the rest of the system. The INTERVIEW prompts the user as each test is passed: RAM, timer, DMA, all MPMs, and 32-bit timer.

The statement “The unit has passed ALL system tests” should appear after these tests. The screen repaints as illustrated in Figure J3-3 for the INTERVIEW 7500 and Figure J3-4 for the INTERVIEW 7000.

If the self test produces an error, try reinstalling the new boards. Go to the beginning of this section and carefully follow the same precautions and instructions given. Replace the cover and power up the unit again so it can perform the self test once more. If the self test still gives errors, call Customer Service.

If the start-up screen is blank or the front-panel LEDs are not red or green (except for the REMOTE and FREEZE LEDs), the connectors may not be attached properly. Try reconnecting them again.
** INTERVIEW 7500 **

DISKS: FLOPPY 1  FLOPPY 2  HARD DISK

PROCESSORS: 4

SELF TEST ERRORS: NONE

Press:
[PROGRAM] to enter the menu page
[RUN] to run the default program

Software Version: 7.00
Firmware Version: 5.00

OPTIONS:
TIM: RS-232/V.24

Copyright (c) 1987, 1989
Telenex Corporation

Figure J3-3 INTERVIEW 7500 screen after self test.

** INTERVIEW 7000 **

DISKS: FLOPPY 1  FLOPPY 2

PROCESSORS: 2

SELF TEST ERRORS: NONE

Please insert system disk and press any key

Figure J3-4 INTERVIEW 7000 screen after self test.

J3.5 Secure the Cover

Place the unit on its back once again and replace the six screws to hold the cover secure.
Appendix J4: Installing Multiplexer Board

CAUTION: STATIC ELECTRICITY CAN DAMAGE THE UNIT WHEN THE COVER IS REMOVED. Before you begin this section, you must take the proper anti-static precautions given in Appendix J1.

Observe the proper handling of the boards: refer to Sections J1 and J2 on proper anti-static precautions.

J4.1 Prepare the Slot or Remove Board

(A) Locate the Slot

Read Appendix J2 for instructions on removing the cover and for information on the interior layout of the INTERVIEW 7000 Series. Then refer to Figure J4-1. The optional multiplexer (mux) board will be installed in the first slot, which is likely to be empty.
Connectors to be detached; see instructions for board being installed.

Figure J4-1 Three cable connections are detached to install the mux board in slot 1.
(B) Detach Surrounding Cables

Make certain the unit is on an anti-static mat and you are wearing a grounded wrist strap.

Disconnect the cable connections carefully. They may be easily removed with an IC clip holder. Lift the connector straight up, holding onto its edges. DO NOT PULL ON THE CABLE; you could break it or damage the connectors. If you do not have an IC clip holder, a blade screwdriver will also work.

CAUTION: If you use a screwdriver, use the top of the black mounting rack (which secures the ends of the boards) as your leverage point. Avoid using the edges of the boards or other components to pry the connectors loose.

Refer to Figure J4-1 to locate the three cable connectors to be detached—two from cables extending from the hard disk drive and one from a cable attached to the back of the LED display. The two from the hard disk drive are attached to the PCM board in slot 4. The other cable is attached to the GBM board in slot 2. Remove these three connectors and fold the cables back out of the way.

(C) Remove Old Board (If Present)

CAUTION: Do not hold any board by its gold edge connector. Hold it by the sides, NEVER touching the components.

Refer to Figure J4-1. If you are replacing the mux board, locate it in slot 1. Remove the board by grasping the top corner edges. Gently pull it straight up and out, and holding the board by its side edges, place it on the anti-static mat.

If requested to do so, package and return the old mux board to the AR Division. Instructions for proper packing and shipping may be found in Appendix Section J2.5.
J4.2 Replace the Board

(A) Install the Board in the Appropriate Slot

Install the mux board (T1 mux board represented in Figure J4-2) in slot 1, in the same manner in which you removed it. Take care to hold it only by the edges. Gently slide it into its connector.

(B) Reconnect the Cables

The cables should be reconnected in the reverse order in which you disconnected them.

First, replace the jumper cable running from the back of the LED display to the GBM board (in slot 2). Then reconnect the two cables from the hard disk drive to the PCM board (in slot 4).
J4.3  **Test the Unit**

Before replacing the cover, review your work to be certain you have followed all the instructions correctly. Follow procedures in Section J3.4 to replace the cover (as a safety measure before you test the unit) as well as to test the new firmware in a power-up before you secure the cover.

The mux board option should be listed on the power-up screen in the OPTIONS: field. If there are any errors listed on the SELF TEST ERRORS: line, repeat the procedures from the beginning of this section and try reseating the mux board. Replace the cover and power up the unit again so it can perform the self test once more. If self test errors persist, contact Customer Service.

J4.4  **Secure the Cover**

When the SELF TEST ERRORS: line shows NONE, place the unit on its back and replace the six screws to hold the cover secure.
Appendix J5: Replacing Firmware on the CPM Board

CAUTION: STATIC ELECTRICITY CAN DAMAGE THE UNIT WHEN THE COVER IS REMOVED. Before you begin this section, you must take the proper anti-static precautions given in Appendix J1.

From time to time there may be a PROM change to upgrade the boards in the INTERVIEW. Correct procedures for changing a PROM are documented in this section.

J5.1 Avoid Inherent Difficulties

When removing or installing PROMs, take care to avoid generating static electricity, bending or breaking pins, and making improper connections.

(A) Static Electricity

Preventing static electricity is essential and is covered in Appendix J1. Taking the proper precautions given in that section will protect the sensitive components.

(B) Bent or Broken Pins

Individual pins on any connector or PROM can easily bend or break. Attaching or detaching components as vertically as possible and aligning them properly will help prevent damage to the pins.

(C) Improper Connections

Improper connections can cause system self-test errors as well as damage to the components. You can avoid improper connections with proper alignment of the pins. Be certain each individual pin is secure in its corresponding socket.

Always align the PROM with its notched end above the notched end of the socket. See Figure J5-2.
Figure J5-1 Four cable connections are detached to remove the CPM board from slot 3.
J5.2 Remove the CPM Board

(A) Locate the Board

Read Appendix J2 for instructions on removing the cover and for information on the interior layout of the INTERVIEW. Refer to Figure J5-1. The CPM board is in the third slot from the left.

(B) Detach Surrounding Cables

Make certain the unit is on an anti-static mat and you are wearing a grounded wrist strap.

To remove the CPM board, first remove the cables surrounding it and attached to it. Disconnect the cable connections carefully. They may be easily removed with an IC clip holder. Lift the connector straight up, holding onto its edges. DO NOT PULL ON THE CABLE; you could break it or damage the connectors. If you do not have an IC clip holder, a blade screwdriver will also work.

CAUTION: If you use a screwdriver, use the top of the black mounting rack (which secures the ends of the boards) as your leverage point. Avoid using the edges of the boards or other components to pry the connectors loose.

Refer to Figure J5-1 to locate the four connectors to be detached—two from cables extending from the hard disk drive and one each from two small jumper cables. The two from the hard disk drive are attached to the PCM board in slot 4. One jumper cable is attached to the GBM board in slot 2 and the other is attached to the CPM board in slot 3. Remove these four connectors and fold the cables back out of the way.

(C) Remove the Board

CAUTION: Do not hold any board by its gold edge connector. Hold it by the sides, NEVER touching the components.

Referring to Figure J5-1, locate the CPM board in slot 3 and remove it by grasping the top corner edges. Gently pull it straight up and out, and holding the board by its side edges, place it on the anti-static mat.
J5.3 Exchange the PROMs on the CPM Board

(A) Locate the PROMs

The PROMs to be changed are located in the upper left-hand corner of the CPM board, one above the other. Refer to Figure J5-2. In the INTERVIEW 7000 and 7200 TURBO, they are identified as 951GA-101X and 951GB-102X; in the INTERVIEW 7500 and 7700 TURBO, they are identified as 951GA-111X and 951GB-112X. (The X refers to the present release level and this last letter will change for each new PROM.) Note that the PROM with GA in its identification number is always in the upper socket and the PROM with GB in its number is always in the lower one.
(B) Remove the PROMs

The PROMs may be removed with an IC clip holder, grasping the ends of the PROM and lifting it as straight up as possible. If an IC clip is unavailable, a blade screwdriver may be used. Do not use any component on the board to pry up the PROM with the screwdriver. Place the screwdriver under the smaller edge of the PROM and gently maneuver the blade in a rocking, semi-circular motion to loosen the PROM pins as evenly as possible.

Place the old PROM in the packing material in which the replacement PROM was sent for its return to the AR Division. (See Appendix F, Packing and Shipping Instructions.)

(C) Install the New PROMs

To install each replacement PROM, refer to Figure J5-2. After examining the PROMs to make sure the two rows of PROM pins are parallel, notice that one end of each PROM has a notch in it. Each notch has a corresponding one in that PROM’s socket on the board. Align the notches as well as the pins and their corresponding holes. Again, the PROM with GA in its identification number goes into the upper socket and the PROM with GB in its number goes into the lower one.

J5.4 Replace the Board

(A) Install the Board in the Appropriate Slot

Replace the board in slot 3 in the same manner in which you removed it. Take care to hold it only by the edges. Gently slide it into its connector.

(B) Reconnect the Cables

The cables should be reconnected in the reverse order in which you disconnected them. First, replace the smallest jumper cable from the PCM board to the CPM board. Next, replace the larger jumper cable from the FEB board to the GBM board (in slot 2). Finally, reconnect the two cables from the hard disk drive to the PCM board (in slot 4).

J5.5 Test the Unit

Before replacing the cover, review your work to be certain you have followed all the instructions correctly. Follow procedures in Section J3.4 to replace the cover (as a safety measure before you test the unit) as well as to test the new firmware in a power-up before you secure the cover.

If the SELF TEST ERRORS: line shows anything other than NONE and you are installing the CPM board, try reinstalling the PROMs. Go to the beginning of this section (Appendix J5) and carefully follow the same precautions and instructions given. Replace the cover and power up the unit again so it can perform the self test once more. If the self test still gives errors, call Customer Service.
J5.6 Secure the Cover

When the **SELF TEST ERRORS** line shows NONE, place the unit on its back and replace the six screws to hold the cover secure.
Appendix J6: Installing Hard Disk Drive (OPT–951–01–1)

CAUTION: STATIC ELECTRICITY CAN DAMAGE THE UNIT WHEN THE COVER IS REMOVED. Before you begin this section, you must take the proper anti-static precautions given in Appendix J1.

Refer to Appendix Section J2.1 for cover removal information and to Appendix Section J2.2 for a description of the interior layout.

NOTE: Your unit will not have the cables across the top of the hard drive bracket—we will install these later. Nor will it have MPM boards in slots 7 and 8, as Figure J2-1 indicates, unless you have the optional MPM boards installed.

OPT–951–01–1, Winchester 20 Megabyte hard disk drive, is for INTERVIEW 7000 and 7200 TURBO units. Installation consists of four steps: removing the empty hard disk bracket, securing the hard drive in the bracket, replacing the bracket and hard drive in the unit, and connecting cables to the hard drive.

J6.1 Remove the Hard Disk Drive Bracket

Locate the hard disk drive bracket at the front of the left side of the unit, as shown in Figure J2-1.

There is an unattached power cable with a four pin female connector behind this bracket. This cable will give power to the hard drive once it is installed. For now, move it towards the back and outside of your unit, clear of the bracket.

There are five screws holding this bracket in the unit—two small Phillips screws toward the front outside of the bracket and three standard-head captive screws holding the bracket to the bottom of the unit, as shown in Figure J6-1. Remove the
two small screws and loosen the three captive screws on the base of the bracket—one is in the front and the other two are in the back. Then lift the bracket straight up and out of the unit.

Figure J6-1 Hard disk drive bracket. Note placement of screws.

J6.2 Secure the Hard Disk Drive in the Bracket

Remove the Winchester 20 Megabyte Hard Disk Drive from its packing material. Notice one side of the drive has exposed components. This side will remain exposed when you place it in the bracket. Slide the drive into the bracket with the component side of the drive facing out on the open side of the bracket, as shown in Figure J6-2.

Use the four Phillips-head screws included in your installation kit to mount your hard drive into the bracket. Insert and secure two of these into the top of the bracket and the other two into the bottom of the bracket.
Appendix J6 Field Service: Installing Hard Disk Drive

J6.3 Replace the Bracket with the Hard Drive into the Unit

Place the hard drive and its bracket in the INTERVIEW 7000 or 7200 TURBO by sliding it down vertically in the same position that the empty bracket had occupied. The exposed side of the bracket and hard drive should face into the unit. Secure the same five screws you released in removing the bracket: three captive screws to the base of the unit and two Phillips screws on the outside of the bracket.

J6.4 Connect the Cables

There are three cables to be connected: the power cable already in your unit and the two cables provided in the installation kit. See Figure J6-2 for the locations of the male connectors on the back of the hard drive.

Figure J6-2 Slide the Winchester hard disk drive into the bracket with the component side exposed and secure the drive in the bracket with the four Phillips-head screws provided.
Locate the power cable (with the four pin female connector) behind the hard drive. With the white wire on top and the red wire toward the bottom, plug the connector into the back near the top of the hard drive.

The two cables provided in the installation kit will connect the hard drive with the PCM board in slot 4 as shown in Figure J2-1. First, note the cables are folded for a proper fit. Next, examine the connections on the cables. In each case the end near the fold is a female slot connector (which connects to the back of the hard drive) and the other end is a female pin connector (which connects to the PCM board).

Attach the smaller of the two cables, labeled "WINCHESTER B, PL-951-44-1," to the smaller male slot connector on the top of the back of the hard drive. Connect the other end of this cable to the smaller male pin connector on the top of the PCM board, taking care the align the pins in their proper holes. In the same manner, attach the other cable, labeled "WINCHESTER A, PL-951-43-1," to the larger male slot connector on the lower back of the hard drive. Connect the other end of this cable to the larger male pin connector on the top of the PCM board.

Your installation of OPT-951-01-1 is complete. It is recommended that you test your unit after installing your hard disk before you begin operations again.

J6.5 Test the Unit

Before you completely secure the cover, it is suggested that you test your unit to make certain that it functions after this installation of the hard disk drive. First, replace the cover.

Grasp the cover with the latch area to the front facing you. Make certain the latches are recessed as far as they can go into the cover and their hooks are not beyond the edge of the cover. Slide the cover down. Take care that the two cables to the hard disk drive are not being pinched or damaged by the cover. (Some models may contain a small, removable, protective plastic sheet to help shield the cables from the cover.)

Now you can safely test the unit. For this test you must power the unit. Reconnect your power cable, turn on the unit, and check for the system self-tests.

The words "System RAM test now accessing all RAM :CPMC[MOM][M2]" appear on your screen, indicating that the first self test for the system is being performed. Following this first test are the tests for the rest of the system. The INTERVIEW prompts the user as each test is passed: RAM, timer, DMA, all MPMs, and 32-bit timer.
The statement "The unit has passed ALL system tests" should appear after these tests. The screen repaints as illustrated in Figure J6-3.

The SELF TEST ERRORS: line should show NONE and the OPTIONS: line should display 01-1 as shown in the figure. If the self test produces an error, try reinstalling the hard drive. Go to the beginning of this section and carefully follow the same precautions and instructions given. Replace the cover and power up the unit again so it can perform the self test once more. If the self test still gives errors, call Customer Service.

If the start-up screen is blank or the front-panel LEDs are not red or green (except for the REMOTE and FREEZE LEDs), the connectors may not be attached properly. Try reconnecting them again.

** INTERVIEW 7000 **

DISKS: FLOPPY 1 FLOPPY 2
PROCESSORS: 2
SELF TEST ERRORS: NONE

Press:
[PROGRAM] to enter the menu page
[RUN] to run the default program

Software Version: 7.00
Firmware Version: 5.00
OPTIONS: 01-1
TIM: RS-232/V.24

Copyright (c) 1987, 1989
Telenex Corporation

Figure J6-3 INTERVIEW 7000 (with OPT-951-01-1) screen after self test.

** J6.6 Secure the Cover **

Place the unit on its back once again and replace the six screws to hold the cover secure.
Appendix J7: Servicing Other Components

Other components of the INTERVIEW 7000 Series may require changing or modifying. Those changes and modifications will be documented in this section.

J7.1 Changing the Dual Floppy Disk Drive Bracket

If either ejection button of your floppy disk drives tends to stick when you operate the drive, you can eliminate the problem by changing the bracket for the dual floppy disk drives. While this is a simple operation, there are certain steps you must follow to insure proper installation.

First you must prepare the drives for the exchange. Then, you will disconnect the power source from the drives and lift out the present bracket holding the drives. Next you will remove the cables, exchange brackets, and replace the cables. Finally, you will replace the disk drives into the unit and reconnect the power source cables.

(A) Prepare the Drives

To prepare the dual floppy disk drives for removal, remove any disks they contain. This secures the eject button inside the drive so it won’t dislodge upon removal of the unit. Locate the cable which attaches the dual floppy disk drives to the PCM board. Detach the connector from the board.

(B) Disconnect the Power Source from the Drives

Locate the small round cables which connect each drive to the power supply. They are found at the lower back of the floppy disk drives. Disconnect them from the drives.

(C) Lift out the Present Bracket

To prepare the present bracket for removal, locate the four captive screws which secure the bracket to the chassis. Using a common blade screwdriver, loosen these four screws from their bosses. These screws will not come out; loosen them sufficiently to release the bracket from the chassis.
Grasp the sides of the bracket and slide it towards the back of the chassis, freeing the eject buttons and the drives from the front panel slot. Then lift the drive assembly up and out of the unit.

(D) Remove the Cables

The dual floppy disk drives have a common cable which you previously detached from the PCM board. Remove the cable from the back of the disk drives, taking care not to damage the cable. Lift the connectors straight off of the connecting pins. For reference, this cable is labeled “PL-951-42-1-A.”

(E) Exchange the Brackets

In this next step, mark the disk drives so that you are certain to keep them in the same positions in the new bracket as they presently sit in the old bracket.

Using a common blade screwdriver, unscrew the four small screws on the top of the bracket and the four small screws on the bottom of the bracket. Place both brackets upright next to each other. Slide out one disk drive and place it in the same position in the new bracket. Similarly, place the second disk drive in the new bracket.

At this point, you will replace the four top screws, turning them just enough to catch the drive in the bracket. One or two turns will suffice on each screw. Turn the assembly over and attach the four bottom screws in the same manner.

Place the assembly upright once again, aligning it with the eject buttons to your right. The drives should still be moveable in the bracket. Grasp the drive closest to you and move it as far to the left and away from you as it will go. Tighten the top two mounting screws for that drive. Follow the same procedure for the drive farthest from you.

Turn the assembly over, keeping the eject buttons on your right. Grasp the drive closest to you and pull it towards you. Tighten the bottom two mounting screws for this drive. Follow the same procedure for the drive farthest from you.

(F) Replace the Cables

Attach the cable to the back of the disk drives. Notice that these cables are "keyed." (Use Figure J3-1 as a reference.) With the eject buttons facing you, wrap the cable around the drive so the cable fold is on the right side of the drive.

(G) Replace the Dual Floppy Disk Drives into the Unit

Check the four bosses which hold the captive screws securing the bracket. If the threaded insert is not flush with or lower than the top of the boss, push it down inside the boss with the tip of a hot soldering iron.
Appendix J7 Field Service: Servicing Other Components

Making sure the power cables you first disconnected are pushed out of the way, place the floppy drive assembly into the unit. Slide it forward, positioning the eject buttons and drive openings into the front panel slots. Tighten down the four captive screws and test the mechanical operation of each disk drive by inserting a floppy disk and then ejecting it.

(H) Reconnect the Power Source Cables

Locate the connecting pins for the power supply cables on the bottom of the rear of each floppy disk drive. Attach these cables to their connectors.

J7.2 Secure the Cover

Grasp the cover with the latch area to the front facing you. Make certain the latches are recessed as far as they can go into the cover and their hooks are not beyond the edge of the cover. Slide the cover down. For the INTERVIEW 7500 and 7700 TURBO—and those INTERVIEW 7000s and 7200 TURBOs with the optional hard drive—take care that the two cables to the Winchester hard disk drive (on your left, directly behind the handle) are not being pinched or damaged by the cover. (Some models may contain a small, removable, protective plastic sheet to help shield the cables from the cover.)

Place the unit on its back once again and replace the six screws to hold the cover secure.

J7.3 Return Parts to AR Division Factory

Any parts which need to be returned to the AR Division should be properly packaged. Contact Customer Service for a RETURN AUTHORIZATION (RA) number.

Customers outside the Washington D.C. 1–800–368–3261
Greater Metropolitan Area

Local customers 644–9190

The RA number should be posted on the outside of the package of all equipment returned for repair. The RA number, as well as a description of the problem, should be cited in all documentation, written correspondence, or telephone conversations concerning the specific repair.
International customers should address the shipment to

Telenex Corporation, AR Division
ATTN RA number
c/o Emery Customs Brokers
101A Executive Drive
Sterling, Virginia 22170
U.S.A.

NOTE: For customs purposes, international customers MUST identify the country of origin for returned equipment on the pro forma invoice. When returning an individual part, use the country of origin listed on the part.

Domestic customers should address the shipment to

Customer Service
Telenex Corporation
AR Division  ATTN RA number
7401 Boston Boulevard
Springfield, Virginia 22153
U.S.A.

Consult Appendix E for additional information on returning parts to the AR Division Factory.
Appendix K: C Language Summary

[The following material is adapted from the appendixes of the Proposed C Standard (ANSI document X3J11/86-098) with certain additions to describe the INTERVIEW implementation of C. The appendixes are not a part of the American National Standard. Information presented is collected from the Standard but is not necessarily complete.]

[For more information on C language and syntax, consult Section C of the Proposed Standard.]

K.1 LANGUAGE SYNTAX

[Editorial comments which appear in this section but are not part of the proposed standard are enclosed in non-italicized square brackets.]

In the syntax notation used in this section, syntactic categories (non-terminals) are indicated by italic type, and literal words and characters (terminals) by bold type. [These items have no sub-categories.] A colon following a non-terminal introduces its definition. Alternative definitions are listed on separate lines, except when prefaced by the words “one of.” An optional symbol is indicated by the subscript opt [here, represented as [opt] ], so that

```
{ expression [opt] }
```

indicates an optional expression enclosed in braces.

K.1.1 Lexical Grammar

K.1.1.1 Tokens

A token is a minimal lexical element of the language. [Categories of tokens are given below. Each of these categories is further described in a separate sub-section.]

token:

- keyword
- identifier
- constant
- string-literal
K.1.1.2 Keywords

[These words (entirely in lower-case) are reserved due to their special meanings.]

\textit{keyword:} one of

\begin{tabular}{llll}
\textit{auto} & \textit{double} & \textit{int} & \textit{struct} \\
\textit{break} & \textit{else} & \textit{long} & \textit{switch} \\
\textit{case} & \textit{enum} & \textit{register} & \textit{typedef} \\
\textit{char} & \textit{extern} & \textit{return} & \textit{union} \\
\textit{const} & \textit{float} & \textit{short} & \textit{unsigned} \\
\textit{continue} & \textit{for} & \textit{signed} & \textit{void} \\
\textit{default} & \textit{goto} & \textit{sizeof} & \textit{volatile} \\
\textit{do} & \textit{if} & \textit{static} & \textit{while} \\
\end{tabular}

[* The reserved word \textit{float} is not used in the INTERVIEW implementation of C.]

[The following two words are reserved in the INTERVIEW implementation.]

\textit{task} \quad \textit{waitfor}

K.1.1.3 Identifiers

An identifier is a sequence of nondigit characters (including the underscore and upper-case letters) and digits. An identifier may not consist of the same sequence of characters as a keyword. [An identifier should also be distinct from user functions or library functions.]

[Essentially, identifiers refer to variables, functions, labels, and various user-defined objects.]

\textit{identifier:}

\begin{tabular}{ll}
\textit{nondigit} & \\
\textit{identifier nondigit} & \\
\textit{identifier digit} & \\
\end{tabular}

\textit{nondigit:} one of

\begin{verbatim}
  a b c d e f g h i j k l m \\
  n o p q r s t u v w x y z \\
  A B C D E F G H I J K L M \\
  N O P Q R S T U V W X Y Z
\end{verbatim}

\textit{digit:} one of

\begin{verbatim}
  0 1 2 3 4 5 6 7 8 9
\end{verbatim}
K.1.1.4 Constants

[Constants may be any of the basic allowable data types. Floating point constants are not supported.]

constant:
  integer-constant
  enumeration-constant
  character-constant

fractional-constant:
  digit-sequence [opt] . digit-sequence
digit-sequence .

exponent-part:
  e sign [opt] digit-sequence
  E sign [opt] digit-sequence

sign: one of
  + -

digit-sequence:
  digit
  digit-sequence digit

integer-constant:
  decimal-constant integer-suffix [opt]
  octal-constant integer-suffix [opt]
  hexadecimal-constant integer-suffix [opt]

decimal-constant:
  nonzero-digit
  decimal-constant digit

octal-constant:
  0
  octal-constant octal-digit

hexadecimal-constant:
  0x hexadecimal-digit
  0X hexadecimal-digit
  hexadecimal-constant hexadecimal-digit

nonzero-digit: one of
  1 2 3 4 5 6 7 8 9

octal-digit: one of
  0 1 2 3 4 5 6 7
hexadecimal-digit: one of
  0 1 2 3 4 5 6 7 8 9
  a b c d e f
  A B C D E F

integer-suffix:
  unsigned-suffix long-suffix [opt]
  long-suffix unsigned-suffix [opt]

unsigned-suffix: one of
  u U

long-suffix: one of
  1 L

enumeration-constant:
  identifier

character-constant:
  'c-char-sequence'

c-char-sequence:
  c-char
  c-char-sequence c-char

c-char:
  any character in the source character set except
  the single-quote ',', backslash \, or new-line character

escape-sequence:
  one of
  \ ' " \? \\ \0 \00 \000 \xh \xhh \xhhh \a \b \f \n \r \t \v

K.1.1.5 String literals

A string literal is a sequence of zero or more characters enclosed in double quotes, as in "xyz".

A double quote within a string literal is represented by the escape sequence \\

string-literal:
  "s-char-sequence [opt]"

s-char-sequence:
  s-char
  s-char-sequence s-char
s-char:
any character in the source character set except
the double-quote "," backslash \, or new-line character
escape-sequence

K.1.1.6 Operators

An operator specifies an operation to be performed that yields a value.
[An operator is a symbol that tells the compiler to perform specific
mathematical or logical manipulations.]
operator: one of
[ ] ( ) . \->
++ -- & * + - ! sizeof
/ % << >> < > <= >= != | && | |
? : = *= /= %= += -= <<= >>= &*= *= |=
, # ##

K.1.1.7 Punctuators

A punctuator is a symbol that has independent syntactic and semantic
significance but does not specify an operation...that yields a value.
Depending on context, the same symbol may also represent an operator or
part of an operator.
punctuator: one of
[ ] ( ) { } * , : = ; ... #

K.1.2 Phrase structure grammar

K.1.2.1 Expressions

An expression is a sequence of operators and operands [variables and
constants] that specifies how to compute a value or (in the case of a void
expression) how to generate side effects.
primary-expression:
identifier
constant
string-literal
( expression )

postfix-expression:
primary-expression
postfix-expression [ expression ]
postfix-expression (argument-expression-list [opt])

K-5
postfix-expression . identifier
postfix-expression -> identifier
postfix-expression ++
postfix-expression --

argument-expression-list:
  assignment-expression
  argument-expression-list , assignment-expression

unary-expression:
  postfix-expression
  ++ unary-expression
  -- unary-expression
  unary-operator cast-expression
  sizeof unary-expression
  sizeof ( type-name )

unary-operator: one of
  & * + - - !

cast-expression:
  unary-expression
  ( type-name ) cast-expression

multiplicative-expression:
  cast-expression
  multiplicative-expression * cast-expression
  multiplicative-expression / cast-expression
  multiplicative-expression % cast-expression

additive-expression:
  multiplicative-expression
  additive-expression + multiplicative-expression
  additive-expression - multiplicative-expression

shift-expression:
  additive-expression
  shift-expression << additive-expression
  shift-expression >> additive-expression

relational-expression:
  shift-expression
  relational-expression < shift-expression
  relational-expression > shift-expression
  relational-expression <= shift-expression
  relational-expression >= shift-expression
Appendix K  C Language Summary

equality-expression:
  relational-expression
equality-expression == relational-expression
equality-expression != relational-expression

AND-expression:
equality-expression
AND-expression & equality-expression

exclusive-OR-expression:
  AND-expression
exclusive-OR-expression ^ AND-expression

inclusive-OR-expression:
  exclusive-OR-expression
inclusive-OR-expression | exclusive-OR-expression

logical-AND-expression:
  inclusive-OR-expression
logical-AND-expression && inclusive-OR-expression

logical-OR-expression:
  logical-AND-expression
logical-OR-expression || logical-AND-expression

conditional expression:
  logical-OR-expression
logical-OR-expression ? expression : conditional-expression

assignment-expression:
  conditional-expression
  unary-expression assignment-operator assignment-expression

assignment-operator: one of
  = *= /= %= += -= <<= >>= &= ^= |=

expression:
  assignment-expression
  expression , assignment-expression

constant-expression:
  conditional-expression
K.1.2.2 Declarations

A declaration specifies the interpretation and attributes of a set of identifiers. A declaration that also causes storage to be reserved for an object or function named by an identifier is a definition.

**declaration:**

```
declaration-specifiers init-declarator-list [opt];
```

**declaration-specifiers:**

```
storage-class-specifier declaration-specifiers [opt]
type-specifier declaration-specifiers [opt]
```

**init-declarator-list:**

```
init-declarator init-declarator-list , init-declarator
```

**init-declarator:**

```
declarator
declarator = initializer
```

**storage-class-specifier:**

```
typedef extern static auto register
```

**type-specifier:**

```
char short int long signed [float not supported] unsigned double const volatile void struct-or-union-specifier enum-specifier typedef-name task-specifier
```

[The last type-specifier in the list above, task-specifier, is specific to the INTERVIEW implementation and is not standard C.]
struct-or-union-specifier:
  struct-or-union identifier [opt] { struct-declaration-list } struct-or-union identifier

struct-or-union:
  struct
  union

struct-declaration-list:
  struct-declaration
  struct-declaration-list struct-declaration

struct-declaration:
  type-specifier-list struct-declarator-list ;

struct-declarator-list:
  struct-declarator
  struct-declarator-list , struct-declarator

struct-declarator:
  declarator
deaclarator [opt] : constant-expression

enum-specifier:
  enum identifier [opt] { enumerator-list } enum identifier

enumerator-list:
  enumerator
  enumerator-list , enumerator

enumerator:
  enumeration-constant
  enumeration-constant = constant-expression

declarator:
  pointer [opt] declarator2
declarator2:
  identifier
  ( declarator )
declarator2 [ constant-expression [opt] ]
declarator2 ( parameter-type-list )
declarator2 ( identifier-list [opt] )

pointer:
  * type-specifier-list [opt]
  * type-specifier-list [opt] pointer
type-specifier-list:
  type-specifier
type-specifier-list type-specifier

parameter-type-list:
  parameter-list
parameter-list , ...

parameter-list:
  parameter-declaration
parameter-list , parameter-declaration

parameter-declaration:
  declaration-specifiers declarator
type-name

identifier-list:
  identifier
identifier-list , identifier

type-name:
  type-specifier-list abstract-declarator [opt]

abstract-declarator:
  pointer
  pointer [opt] abstract-declarator2

abstract-declarator2:
  ( abstract-declarator )
abstract-declarator2 [opt] [ constant-expression [opt] ]
abstract-declarator2 [opt] ( parameter-type-list [opt] )

typedef-name:
  identifier

initializer:
  assignment-expression
  { initializer-list }
  { initializer-list , }

initializer-list:
  initializer
  initializer-list , initializer

task-specifier:
  task-identifier { task-body }
task { task-body }
task-identifier
task-body:
  external-definition
  layer-declaration
  task-body external-definition
  task-body layer-declaration

layer-declaration:
  #pragma layer integer-constant

K.1.2.3 Statements
A statement specifies an action to be performed.

statement:
  labeled-statement
  compound-statement
  expression-statement
  selection-statement
  iteration-statement
  jump-statement
  waitfor-statement

[The last statement in the list above, waitfor-statement, is specific to the INTERVIEW implementation and is not standard C.]

labeled-statement:
  identifier : statement
  case constant-expression : statement
  default : statement

compound-statement:
  { declaration-list [opt] statement-list [opt] }

declaration-list:
  declaration
  declaration-list declaration

statement-list:
  statement
  statement-list statement

declaration-list:
  declaration

expression-statement:
  expression [opt] ;

selection-statement:
  if ( expression ) statement
  if ( expression ) statement else statement
  switch ( expression ) statement
iteration-statement:
    while (expression) statement
do statement while (expression);
for (expression[opt]; expression[opt]; expression[opt]) statement

jump-statement:
goto identifier;
continue;
brake;
return expression[opt];

waitfor-statement:
    waitfor {}
    waitfor { waitfor_list }

waitfor-list:
    expression : statement
    waitfor_list expression : statement

K.1.2.4 External definitions

file:
    external-definition
    file external-definition

external-definition:
    function-definition
    declaration

function-definition:
    declaration-specifiers [opt] declarator function-body

function-body:
    declaration-list [opt] compound-statement

K.1.3 Preprocessing directives
[These are instructions to the compiler included in source code.]
The implementation can include sections of program text, conditionally
include other source files, and replace macro's.

preprocessing-file:
    group
    preprocessing-file group

group:
    group-part
    group group-part
group-part:
  tokens [opt] new-line
  if-section
  control-line

tokens:
  token
  tokens token

if-section:
  if-group elif-groups [opt] else-group [opt] endif-line

if-group:
  # if constant-expression new-line group [opt]
  # ifdef identifier new-line group [opt]
  # ifndef identifier new-line group [opt]

elif-groups:
  elif-group
  elif-groups elif-group

elif-group:
  # elif constant-expression new-line group [opt]

else-group:
  # else new-line group [opt]

endif-line:
  # endif new-line

control-line:
  # include <x-char-sequence> new-line
  # include “x-char-sequence” new-line
  # define identifier tokens [opt] new-line
  # define identifier lparen identifier-list [opt]) tokens [opt]
  new-line
  # undef identifier new-line
  # line digit-sequence string-literal [opt] new-line
  # pragma x-char-sequence new-line
  # new-line

x-char-sequence:
  x-char
  x-char-sequence x-char

x-char:
  any character in the source character set except the
  new-line character
lparen:
the left-parenthesis character without preceding white space

new-line:
the new-line character

K.1.3.1 AR pragmas

#pragma hook hook_type hook_text
  hook_type: decimal-constant
  hook_text: string-constant

#pragma layer integer-constant

#pragma nowarn

#pragma object object_file_name
  object_file_name: string constant

#pragma tracebuf trace_buffer_declaration_list [opt]
  trace_buffer_declaration_list:
    trace_buffer_declaration
  trace_buffer_declaration_list trace_buffer_declaration

trace_buffer_declaration:
  decimal-constant decimal-constant
  * decimal-constant
Index To ADD–951–10

A

Allocating disk space, 12–3
AUX I/O, connector, 1–9
AUX leads, 1–9
Auxiliary TTL connector, 1–9

B

Back panel, 1–7
  fan, 1–8
  fan filter, clean to prevent overheating, 1–9
  frequency selection, 1–8
Input/Output connectors, 1–9
  AUXILIARY TTL, 1–9
  CRT/RGB, 1–9
  internal MODEM, 1–9
  PRINTER, 1–9
  REMOTE RS–232, 1–9
  RS–170 composite video, 1–10
  on/off (power) switch, 1–8
  power connector, 1–7
  voltage selection, 1–7
BCC Setup, overview of screen, 2–11
BERT Setup, overview of screen, 2–11
Boot-up
  creating a user interface, HRD/usr/user_intrf, 2–4
  running default program, /usr/default, 2–5
  system disk, 2–3

C

Capture Memory, field on Record Setup menu, 12–9
Character buffer, storage capacity, 1–12

Clock
  signal, 1–13
  See also Speed
  time–of–day, 1–14
  See also Date/Time Setup

Compilation
  error diagnostics, 2–15
  fields that can be modified without causing recompile, 2–16
  rerun without recompiling, 2–15
  seven phases, 2–14

Conditions, EIA, fails to come true, 2–19

Connectors
  back panel, 1–7
  power, back panel, 1–7
  Test Interface Module, 1–10–1–16

Control leads, playback, 2–17
  of bit–image data, 2–17
  of character data, 2–17

CRT/RGB connector, 1–9

D

Data acquisition tracks, 12–3
Data capture, 2–17
  See also Playback; Recording data
  RAM, data storage, 1–12
Data plus leads, failure of leads to transition, 2–20
Data source, connection to, 1–16
Data Transfer, Disk Maintenance, menu selection, 12–7
Data transfer, INTERVIEW 5, 10, 15 PLUS data, 12–10–12–16
Default menus, how to change, 2–6
Directories
  /usr, 2–4
  /usr/default, 2–5

MAY 1990
Disk drives, 1–5
  drive references and priority, 12–3
microfloppies
  compatibility, 1–5
  write protection, 1–5
microfloppy disks, storage capacity, 1–5
Winchester hard disk, 1–7
Disk Maintenance, 12–3, 12–16
  allocating disk space, 12–3
  command
    Data Transfer, 12–7
    Disk Summary, 12–7
    Duplicate Disk, 12–10
    Format Disk, 12–5
    INT 10, 12–10, 12–16
  data acquisition tracks, 12–3
  description of disks, 12–3
  initializing system, INTERVIEW 7000, 2–3
  installing new system software, 2–7
  menu selections, 12–2, 12–4, 12–16
  overview, 2–14
Disk Name, subfield on Disk Maintenance menu, 12–5
Disk Number, subfield on Disk Maintenance menu, 12–5
Disk Summary, Disk Maintenance, menu selection, 12–7
Display, plasma, 1–3
Display Setup, traces, 2–20
Display Setup screen, overview, 2–11
Drive Type, subfield on Disk Maintenance menu, 12–6
Duplicate Disk
  Disk Maintenance, menu selection, 12–10
  installing new system software via the DUPDISK command, 2–7
  Fan
  back panel, 1–8
  clean filter to prevent overheating, 1–9
FEB Setup, overview of screen, 2–11
File Maintenance, File Maintenance screen, overview, 2–13
Format Disk, Disk Maintenance, menu selection, 12–5
Freeze LED, front panel, 1–5
Frequency selection, back panel, 1–8
From, subfield on Disk Maintenance menu, 12–8
From Disk Number, subfield on Disk Maintenance menu, 12–10
Front end buffer
  applies to playback of bit-image data, 2–17
  Idle Suppress field, does not apply to playback of bit-image data, 2–17
  on playback path of bit-image data, 2–17
  time ticks and EIA leads, storage of, 1–12
  2–17
Front panel, 1–3
  function keys, 1–4
  LED overlay, 1–5
  LED's, 1–4, 2–17
  U/A, 1–11
  plasma display, 1–3
Function keys, 1–4
General operation,
  boot-up program, creating a user interface, 2–4
  changing default menus, 2–6
  common problems, 2–19, 2–21
  data capture, 2–17
    front end buffer, 2–17
    front end buffer, on playback path of bit-image data, 2–17
  initializing system, INTERVIEW 7000, 2–3
  installing new system software, 2–7
  overview of menus, 2–9–2–14
    power up, 1–16, 2–1
    rerunning a test program, 2–15
    running a test program, 2–14
    running default program, 2–5
MAY 1990
ADDENDUM

H

Hardware, 1-3—1-16
  back panel, 1-7
clock, 1-13
disk drives, 1-5
front panel, 1-3
keyboard, 1-3
  operating environment, 1-14
  operating positions, 1-14
  physical dimensions, 1-3
  power up, 1-14
  storage capacity, 1-12
High speed, optimizing performance, 2-22
HRD/usr/user_intrf
  affect on Start Up screen, 2-2
  boot-up program, creating a user interface, 2-4

I

Idle Suppress, field on Front-End Buffer Setup
  screen, does not apply during playback, 2-17
Initializing system software, INTERVIEW 7000, 2-3
Input/output connectors, back panel, 1-9
  AUXILIARY TTL, 1-9
  CRT/RGB, 1-9, 1-10
  internal MODEM, 1-9
  PRINTER, 1-9
  REMOTE RS-232, 1-9
  RS-170 composite video, 1-10
Installing new system software, on hard disk, 2-7
INT 10, Disk Maintenance selection, 12-14—12-16
INTERVIEW transfer, INTERVIEW 5, 10, 15
  PLUS data, transfer from Disk Maintenance
  screen, 12-10—12-16
Instrument, menu field on Disk Maintenance
  screen, for INTERVIEW 10 transfer, 12-14
Internal MODEM connector, 1-9

K

Keyboard, 1-3
  See also ASCII keys; entry under each special
  key function keys, 1-4

L

Layer Setup screen, overview, 2-13
Layers, passing data between, 2-20
LED's
  front panel, 1-4, 1-5, 2-20
  INTERVIEW status, 1-5
  interface status, 1-4
  U/A, 1-5, 1-11
Test Interface Module, 1-11
Line data, data capture, 2-17
Line Setup screen, overview, 2-10

M

Maximum data rates
  data analysis, 1-13
  data recording, 1-13
Memory, capacity, 1-12
Menus, overview, 2-9—2-14
  See also Separate listing, each menu name
  configuring menus, 2-10
  Program Menu, 2-9
Microfloppy disks
  compatibility, 1-5
  storage capacity, 1-12
  write protection, 1-5
Miscellaneous Utilities, overview, 2-14
Mode, test mode field on Line Setup menu, 1-11
Modem connector
  external, 1-9
  internal, 1-9
MPM errors, 2-16

O

Object code, rerunning object version of
  program, 2-15
On/off (power) switch, back panel, 1-8
Operating environment, 1-14
Operating positions, 1-14
Overlay, Test Interface Module, 1-5

P

Physical dimensions, size and weight, 1-3
Playback
  control leads, 2-17
disk data, 2-17
EIA leads, storage, 1-12
time ticks, 2-17

MAY 1990
ADDENDUM

Power connector, 1-7
Power switch, back panel, 1-8
Power up, 1-14, 2-1
self tests, 2-1
Printer connector, 1-9
Program key, 2-4
unit unexpectedly enters Run mode, 2-19
Protocol Spreadsheet
increasing the size of, 2-13
overview, 2-12
syntax errors, 2-15
unexplained strike-through’s, 2-19

RAM, data storage, 1-12
REMOTE RS-232 connector, 1-9
Recording data
maximum rate, 1-13
with EIA lead transitions, 1-12
Record Setup, overview of screen, 2-11
Remote LED, front panel, 1-5
Resolution, display, 1-3
RGB video connector, 1-9
RS-170 video connector, 1-10
RS-232, connector, REMOTE, 1-9
RS-232/V.24, test connector, 1-10
Run mode, unit fails to enter, 2-19

Screen buffer, storage capacity, 1-12
Self tests, 2-1
Setup menus
BCC Setup, overview, 2-11
BERT Setup, overview, 2-11
Display Setup screen, overview, 2-11
FEB Setup, overview, 2-11
Line Setup, overview, 2-10
overview, 2-10
See also Separate entry under name of each menu
Record Setup, overview, 2-11
Speed, optimizing high-speed performance, 2-22
Start At Block, subfield on Disk Maintenance menu, 12-9
Start up screen, 2-2
Statistics menus, overview, 2-13
Strike–through’s, Protocol Spreadsheet, 2-19
Syntax errors, Protocol Spreadsheet, 2-15
/sys/fifty_hertz, file name, 1-8
System disk, boot-up, 2-3

Temperature, operating, 1-14
Test connectors
software control, 1-11
Test Interface Module, back panel, 1-10—1-16
TO DCE, 1-11
TO DTE, 1-11
Test Interface Module
installation, 1-14
LED overlay, 1-5
installation, 1-15
LED’s, back panel, 1-11
software control, 1-11
test connectors, 1-10—1-16
Time ticks
playback, 2-17
of bit-image data, 2-17
of character data, 2-17
storage of, 1-12
Time/Date Setup, overview, 2-14
Timers, no values displayed, 2-20
TO DCE, test connector, 1-11
TO DTE, test connector, 1-11
To Disk Number, subfield on Disk Maintenance menu, 12-10
Transmit string, does not appear on display, 2-20
Trigger conditions, EIA, fails to come true, 2-19
Trigger Setup menus
overview, 2-11
transmit string, does not appear on screen, 2-20
Trouble-shooting
data plus leads, failure of leads to transition, 2-20
data–plus–leads display, failure of leads to transition, 2-20
layers, passing data between, 2-20

MAY 1990
overheating, 1-9, 2-21
Program key, unit unexpectedly enters Run mode, 2-19
Protocol Spreadsheet, unexplained strike-through's, 2-19
Run mode, unit fails to enter, 2-19
timers, no values displayed, 2-20
transmit string, does not appear on screen, 2-20
trigger conditions, EIA, fails to come true, 2-19
Type, subfield on Disk Maintenance menu, 12-8

U

U/A, LED, 1-11
Unresolved reference, error message, 2-19
/usr/default
affect on Start Up screen, 2-2
boot-up menu configuration, 2-5
default program, 2-5

V

V.35, test connector, 1-10
Video connectors
CRT/RGB, 1-9
RS-170 composite video, 1-10
Voltage selection, back panel, 1-7

W

Winchester hard disk
installing new system software, 2-7
storage capacity, 1-12
Write protection, microfollies, 1-5
Symbols

_abort, overlay for BOP abort, 5–13
  C coding, 59–6

badbcc, overlay for bad BCC or FCS, 5–13
  C coding, 59–6

bit mask symbol, in Receive string condition, 21–7

» close double paren symbol, 3–5, 22–4, 25–5
*/ closing delimiter for comment, 24–13

don’t care symbol
  in Receive string condition, 21–7
  in spreadsheet string search, 33–35

flag, 7E flag symbol, 5–13
  C coding, 59–6

goodbcc, overlay for good BCC or FCS, 5–13
  C coding, 59–6

highlighted plus symbol, indicates wrap in logical line, 26–3

not equal flag symbol
  in Receive string condition, 21–7
  in spreadsheet string search, for beginning of frame, 33–35

open double paren symbol, 3–5, 22–4, 25–5
/* opening delimiter for comment, 24–13

pad, in Outsync Char field, 4–8

sync, sync symbol, 4–7, 5–13
  C coding, 59–6
~ tilde symbol, 3–4, 24–3

time-fill, 5–8
/ (root) directory, 13–4

A

A Bus En, indicator on RS-485 Test Interface Module, 47–5

Abbreviations, glossary, B–1–B–10

Abort
  adjunct to monitor-frame condition
    LAPD, 39–13, 39–15
    SDLC, 35–14, 35–16
    X.25 Layer 2, 33–14, 33–15
  adjunct to send-frame action
    LAPD, 39–25
    SDLC, 35–27
    X.25 Layer 2, 33–28
  adjunct to send-string action, Layer 1 (BOP), 28–9
  appended to transmit string, 8–6
  defined for BOP, 8–6, 33–10, 34–14, 39–9, 40–6, 42–6, 43–7
  field on BCC Setup menu, 8–12
  in C, 59–6
  monitor/receive condition, Layer 1 (BOP only), 28–4
  overlay, BOP only, 8–4

Absolute pathnames, files and directories, 13–5

Accumulate, layer-independent action, 27–15

Accumulate action, 17–7
  may apply to current, last, minimum, or maximum values, 17–9
  not found on trigger menus, 17–8
  translated into C, 62–13
  used to log one hour per day over days or weeks, 17–10

Accumulate counter, layer-independent action, 27–15

Accumulate timer, layer-independent action, 27–15

Accumulator
  created by being named in Accumulate action, 27–15
  printing line of tabular statistics for, 27–16
  structures, accumulator_struct, 62–14

Acronyms, glossary, B–1–B–10
Actions
  capture memory on/off, 11-10
  Enhance, control of color display, 16-5
  in C, 53-12
  Layer 1, 28-6
  accessed via Done key, 28-6
  Layer-independent, 27-8–27-13
  Protocol Spreadsheet, programming block, 24-8
  comments in, 24-13
  record on/off, 11-10
ADD, address, trace column
  SDLC, 35-7
  X.25 Layer 2, 33-7
Address
  adjunct to monitor/receive-frame condition
  SDLC, 35-13
  X.25 Layer 2, 33-14
  adjunct to send-frame action
  SDLC, 35-24
  X.25 Layer 2, 33-25
  trace field, SNA-SDLC, 36-6
Affects, field on BCC Setup menu, 8-12
Again, editor command, 26-9
Aggregate G.703 record, 50-5, 50-26
Aggregate T1 Record, 49-5, 49-26
Alarm
  field on Trigger Setup menu, 22-11
  layer-independent action, 27-13
  routines, sound_alarm, 69-16
Allocating disk space, 12-3
Alternate Mark Inversion, transmission technique, 48-4, 49-3
AMI. See Alternate Mark Inversion
ANSI format, SS#7 layer 3, 43-4
Append, run-mode printer output to existing disk file, 14-7
Array
  may be initialized by a string in C, 56-29,
  57-13
  name of array is 4-byte address in C, 58-2
  size, 56-20
ASCII
  default BCC parameters, 8-8
  hex-to-display conversion table, D2-3
  keyboard-to-ASCII conversion table, D1-3
  keys, in programming menus and spreadsheet, 3-4

Async
  data setup, 4-10
  sample Line Setup, 4-13
Attributes
  format of 32-bit word same in Display Window and trace buffers, 61-14
  in character buffer, 59-6
  in Display Window
    color, 61-4
    current font, 61-4
    derived from the current window_color and window_modifier values, 61-35
    enhancements, 61-4
  mapping of %m argument to attribute variables, illustrated, 61-33
  set via %m conversion specifier in format of displayf routine, 61-12
  stored in window_color and window_modifier variables, 61-12
  three bytes of attributes to one byte of data, 61-4
  in trace buffer
    mapping of %m argument to attribute variables, illustrated, 61-34
    updated by %m conversion in format of tracef routine, 61-27
    written via %m conversion specifier to trace_buf.hdr structure, 61-29
    less flexible set of attributes in character buffer than in Display Window, 59-15
    not available via display_prompt routine, 61-3
AUX Control, field on Interface Control menu, 10-17
AUX I/O, connector, 1-9
AUX leads, 1-9
AUX outputs
  driven on/off by spreadsheet actions, 10-9
  driven on/off on Interface Control menu, 10-17
  location on TIM, 10-8
  on/off, Layer 1 emulate-mode action, 28-10
Auto Configure, screen in automonitor mode, 5-3
Auto Termn, DIP switch, on RS-485 Test Interface Module, 47-4
Auto-indent, editor command, 26-9
Automatic OSI primitives. See Primitives
Automatic X.25 Layer 2, 33-36
Automonitor mode
  setting up, 4-3
  stage in autoconfiguration displayed in Status field, 5-3

Index to Part I and Part II
updates Line Setup screen, 5–4
with no clock, speed defaults to 168 kbps, 5–4

Autosync, subfield on Line Setup menu, 4–7, 4–8

Auxiliary connector, 1–4

Auxiliary TTL connector, 1–9
AUX port controlled by C program,
68–3–68–11

Average, column on Tabular Statistics screen,
17–6, 17–11, 27–10

accumulator_struct, accumulator structure,
62–13
defined, 62–14

B

B Bus En, indicator on RS–485 Test Interface
Module, 47–5

B channels, ISDN, 48–3, 48–4

B8ZS. See Binary 8 Zero Suppression

Back panel, 1–7
fan, 1–8
fan filter, clean to prevent overheating, 1–9
frequency selection, 1–8
Input/Output connectors, 1–9
AUXILIARY TTL, 1–9
CRT/RGB, 1–9
internal MODEM, 1–9
PRINTER, 1–9
REMOTE RS–232, 1–9
RS–170 composite video, 1–10
on/off (power) switch, 1–8
power connector, 1–7
voltage selection, 1–7

Backslash, entry of inside prompt message,
27–12

Backslash (\), escape character in C string,
56–20

Bad BCC
adjunct to monitor–frame condition
LAPD, 39–13, 39–15
dce_bad_bcc, C variable, 77–1
dte_bad_bcc, C variable, 77–1
SDLC, 35–14, 35–16
dce_bad_bcc, C variable, 73–1
dte_bad_bcc, C variable, 73–1
SS#7 Layer 2
dce_bad_bcc, C variable, 79–1
dte_bad_bcc, C variable, 79–1

X.25 Layer 2, 33–14, 33–15
dce_bad_bcc, C variable, 71–1
dte_bad_bcc, C variable, 71–1

adjunct to send–frame action
LAPD, 39–25
X.25, 33–28

adjunct to send–string action, Layer 1, 28–9
appended to transmit string, 8–5
as condition, 8–5
fevar_bd_bcc_td and fevar_bd_bcc_rd, C
events, 59–2

monitor/receive condition, Layer 1, 28–4
operational only when Rcv Blk Chk
enabled, 28–4

Baud rate, default value for remote port, 67–20

Baudot

hex–to–display conversion table, D2–3
keyboard–to–Baudot conversion table, D1–9
no default BCC parameters, 8–8

BCC
See also Block checking
cross between a Layer 1 and Layer 2
function, 28–4
indicated by transmit tag in header of IL
buffer, 55–7
Layer 1 condition, 28–4
operational only when Rcv Blk Chk
enabled, 28–4

subfield on Trigger Setup menu, 8–5, 22–4

trace column
LAPD, 39–9
Q.931, 40–6
SDLC, 35–10
SNA–SDLC, 36–6
SS#7 Layer 2, 42–6, 43–7
X.25 Layer 2, 33–10
X.25 Layer 3, 34–14

BERT
“force–loopback” programming example,
9–21–9–22
analyze–only mode, 9–18
automatic error injection, 9–14
enabling/disabling by softkey, 9–18
status message, 9–21
block size, 9–13
G.703, 9–42
T1, 9–29
clearing counters, 9–6
clearing the results screen, 9–18
counters, 9–19
five pseudorandom patterns, 9–3
algorithms, 9–3

Index to Part I and Part II
freeze mode, 9-17
G.703. See T1, BERT
G.703 BERT, run-time function key, 9-46
half duplex, 9-6
"receive and analyze" versus "generate"
mode, 9-7
initiating the send-receive cycle, 9-7
invert, G.703, 9-41
manual error injection, 9-18
operating mode, selected on Line Setup menu, 9-5
pattern, G.703, 9-41
patterns, 9-7
reinitializing a running test, 9-18
relation of BERT Setup menu to Interface
Control screen, 9-7
relation of BERT Setup menu to Line Setup
menu, 9-6, 9-15
synchronous versus asynchronous, 9-15
T1. See T1, BERT
T1 BERT, run-time function key, 9-34
test length, 9-14
G.703, 9-42
T1, 9-29
BERT modes, setting up, 4-4
Begin, editor command, 26-5
Begin CAS MF w/frame containing frame align.
signal, G.703, field on Interface Control menu, 50-24
Binary, user-defined routine that displays binary
value of byte, 58-5
Binary 8 Zero Suppression
T1 Interface Control screen, 49-26
transmission technique, 49-4
Binary display, of cursor characters, 5-15
in relation to order of transmission, 5-15
Bipolar violations
BPV's received
G.703, statistics display, 50-28
T1 statistics display, 49-29
G.703 transmissions, 50-19
T1 transmissions, 49-26
Bisync
CRC mode, 8-13
advantage over selectable mode, 8-13
field on BCC Setup menu, 8-11
sample Line Setup, 4-13
Bit errors
G.703 BERT statistics, 9-44
T1 BERT statistics, 9-31
Bit Mask key, 3-6

Bit mask
in Protocol Spreadsheet strings, 29-3
in Receive string condition, 21-7
in Suppress field, 5-11
masking bits in C variables, 57-8, 70-2
to detect XON and XOFF only, 5-12
Bit Order/Polarity, field on Line Setup menu,
4-11, 9-16
significance in BERT testing, 9-16
Bit order
in relation to hexadecimal notation, 5-14
in relation to pattern sync in BERT, 9-11
normal versus reverse, 4-11
Bit-image data, 11-3
playback, 11-4
Bit-robbing, T1 transmissions, 49-32
Bits
field on Line Setup menu, 4-6, 9-16
in BERT testing, 9-15
number of, in setup, 4-6
per character, default value for remote port,
67-20
Bits In Error, BERT counter, 9-20
Bitwise and (&), C operator, 57-8, 61-36, 70-2
Blink, subfield on Trigger Setup menu, 22-7
Block
component of BERT test, 9-13, 9-29, 9-42
editor command, 26-5
Block checking
automatically on for BOP, 8-4
distinction between transmitting and evaluating
BCC, 8-3
enabling BCC overlays, 4-9, 8-3.
for DDCMP
automatically on, 37-2
data BCC may be tested as event variable in
C, 37-2
header BCC only may be tested by trigger,
37-2
looking under BCC overlay, 8-4
parameters defined on BCC Setup menu, 8-6
result used as trigger condition, 8-5
Block No, field on Line Setup menu, 4-5
Block Size, field on BERT Setup menu, 9-13
Blocks In Error
BERT counter, 9-20
G.703 BERT statistics, 9-44
T1 BERT statistics, 9-31
Blocks Received, BERT counter, 9-20
Blocks received
G.703 BERT statistics, 9-44
T1 BERT statistics, 9-31

Index to Part I and Part II
Blocks Sent, BERT counter, 9–20

Blocks sent
G.703 BERT statistics, 9–44
T1 BERT statistics, 9–31

BNC, G.703 connectors, 50–8, 50–9

BOP, synchronization and BCC parameters always defined for, 4–9, 8–4, 8–8

Boards
See also Field Service packaging and returning. See Field Service

Boot-up
creating a user interface, HRD/usr/user_intrf, 2–4
running default program, /usr/default, 2–5 system disk, 2–3

BPV’s received
G.703, statistics display, 50–28
T1 statistics display, 49–29

BPV–free seconds
G.703, statistics display, 50–28
T1 statistics display, 49–29

BPVs. See Bipolar violations

BREAK, T1 test access point, 49–10

Break
C statement, 53–6, 53–13, 56–2
used to exit a waitfor, 53–3, 53–6, 53–9
transmitting a break, set_tcr_b, C routine, 59–14

Break key, 26–5

Breakout panel, on Test Interface Module, 10–5
RS–449, 45–4
RS–485, 47–3
V.35, 44–4
X.21, 46–4

Buffer Control Leads, field on Front–End Buffer Setup screen, 7–5

Buffer Full
condition, Trigger Setup menus, 21–11
fevar_rcv_buffer_full, C event, 59–2
layer–independent condition, 27–5
rcv_buffer_full, C variable, 59–2

C

C, color, field on Graphical Statistics menu, 18–7

C language
array, name of array is 4–byte address, 58–2
care mask, 57–8, 60–1, 61–36, 70–2

comments, 53–10
condition clause, equivalent to trigger, 53–8
may contain multiple expressions, 53–9
constants
character, 57–7
decimal, 57–7
hexadecimal, 57–7
octal, 57–7
conversion specifiers, 61–37
%u, hex character, 57–8
%b, 61–12
%c, 61–11
character, 56–16, 57–8
%d, 61–11
signed decimal, 56–14, 56–16, 57–8
%H, 61–11
%i, 61–11
%sm, 61–12, 61–27, 61–30, 61–32
%o, 61–11
octal, 57–8
%p, 61–11
%q, 61–11
%u, 61–11
unsigned decimal, 56–14, 56–16
%X, 61–11
%x, 61–11
hex, 56–14, 56–16, 57–8
data types, 56–13
char, 56–13
int, 56–13
long, 56–13
long routine returns a long, 58–3
short, 56–13
short routine returns a short, 58–3
signed, 56–13
unsigned, 56–13
void, return statement invalid with this type, 58–3
declarations
automatic, 53–14
format, 53–14, 57–2
positioning and grouping, 53–15
scope, 52–16, 53–17
global, 53–17
error messages, issued by compiler and preprocessor, A3–1–A3–16
event variables
may be created by user, 57–6
one used by translator for every spreadsheet condition, 53–10
programming rules, 53–11, 57–4, 57–6
executable statements, location on spreadsheet, 53–17
expressions, conditional
nonzero always true, 53–12, 58–2
zero always false, 53–12, 58–2
Index to Part I and Part II

initialization, variable must be static to pass initialized value into waitfor statement, 53-17, 56-17
introduction to AR version, 56-3—56-15
variations from standard C, 56-3
keywords
label, equivalent to spreadsheet State, 53-2
task, 56-3
equivalent to spreadsheet Test, 53-1
placed at highest level of source code, 52-16
locating compilation errors, 56-4
main function
created by translator, 52-1
placed at highest level of source code, 52-16
nonevent variables
checked when event is signalled, 57-5
ture in expressions with nonzero value, 53-12
operators
++, 58-6
->, 57-17
bitwise and (&), 57-8, 61-36, 70-2
precedence, 56-17
right shift (>>, has different effect on signed and unsigned variables, 56-15
sizeof, 56-20, 71-13, 72-17, 73-13
pointer
always 32 bits no matter what the data type, 57-11
creating a pointer, 57-11
incrementing pointers of various data types, 57-12
m_packet_info_ptr, pointer to first data byte in X.25 packet, 72-8
m_ptr_to_call_ref, pointer to Q.931 call-reference field, 78-4
m_ptr_to_info_element, pointer to Q.931 info-element field, 78-4
making a pointer to the data in a received frame, 71-8, 73-8, 77-8
making a pointer to the data in an IL buffer, 57-10, 63-41
pointing with subscripts, 57-12, 57-13
rcvd_pkt_info_ptr, pointer to first data byte in X.25 packet, 72-8
rh_ptr, pointer to first byte of SNA request/response header, 74-4
ru_ptr, pointer to first byte of SNA request/response unit, 74-4
string, 56-19
structure pointer, creating a structure pointer, 57-16
th_ptr, pointer to first byte of SNA transmission header, 74-4
predeclared identifiers
event, 56-4
fast_event, 56-4
label, 56-4
preprocessor directives
#define, 56-5, 61-40
#include, 56-6, 61-23, 61-29, 61-30, 65-3
#pragma, placed inside of task definition, 52-16
#pragma hook
defining the hook text, 56-12
format of, 56-11
hook text added to top-level main function, 56-12
in linkable-object files, 56-11
system-generated during Compile spreadsheet, 56-11
used to "force" a call to a routine, 56-11
using multiple hooks, 56-12
#pragma layer, used to declare a layer, 53-1
#pragma nowarn, used to suppress compiler warnings, A3-1
#pragma object
format of, 56-8
placement of, 56-8
used to combine routine definitions with spreadsheet program, 56-8
#pragma tracebuf, used to configure size of trace-buffer arrays, 61-26
program main, 52-1
recommended sources, 56-22
regions on spreadsheet actions, 53-12
conditions, 53-7
enter state, 53-3
layer, 53-1
next state, 53-4
state, 53-2
summary, 53-14
test, 53-1
routines, 58-1—58-6
always followed by parentheses, 58-2
most associated with specific spreadsheet condition or action, 58-1
nonzero return makes conditional statement true, 58-6
not usually necessary to declare, 58-1
user-defined, 58-4—58-6
display_binary, 58-5
strcmp, 58-6
temporary_prompt, 58-4
statements
break, 53-6, 53-13, 56-2
used to exit a waitfor, 53-3, 53-6, 53-9
goto, 53-4, 53-5, 53-7, 53-13
placed inside of state loop, 52-16
used to move program control to a
different state-label, 53-2, 53-4
if, 57-4, 58-6
nonzero expressions always true inside of
if statement, 58-2
routine that returns nonzero makes if
statement true, 58-6
used in Enter State conditions, 53-4
return, 58-2
breaks out of while loop, 58-6
waitfor, 53-2, 53-4, 53-5, 53-6, 53-7,
53-9, 53-13, 56-3, 57-4
defines a set of interrupts (events), 54-1
placed inside of state loop, 52-16
variable must be static
to pass value into
waitfor, 53-17, 56-17
while, 58-6
nonzero expressions always true inside of
while statement, 58-2
status variables. See Nonevent variables
storage classes, static, variable must be static
to pass value into waitfor, 53-17, 56-17
storage-class specifiers, extern, cannot be
declared below Test level, 53-15
stream, 65-1
strings, 56-19
comparing strings, 57-14, 58-6
creating a string, 57-12
non-literal characters inside strings, 56-21
nonliteral characters inside set_print_header
strings, 64-8
structure, accessing an element of a structure,
57-15
syntax summary, K-1-K-14
third tier in programming hierarchy, 19-4
translator
 creates automatic main function, 52-1
levels of source code, 52-16
uses external routines, 58-1
variables, 57-1-57-17
C translator, 52-1
error messages, A2-1-A2-6
C/R
adjunct to monitor/receive-frame condition,
LAPD, 39-13
adjunct to send-frame action, LAPD, 39-22
Command/Response, trace column, LAPD,
39-8
CAL-REF-VAL, call reference value, trace
column, Q.931, 40-5
CALL, send action, X.25 Layer 3, 34-31
CALL_CONF, send action, X.25 Layer 3,
34-31
CALLED, field on X.25 Packet Level Setup
screen, 34-6
CALLING, field on X.25 Packet Level Setup
screen, 34-6
CAS MF resync criteria, G.703, field on
Interface Control menu, 50-25
CAS MF sync criteria, G.703, field on Interface
Control menu, 50-25
CAS multiframe, G.703 frame structures,
50-31
Cable length, T1 Interface Control screen,
49-21
Cable type, T1 Interface Control screen, 49-20
Cables
c connection. See Field Service
disconnection. See Field Service
null-modem cable for remote port I/O, 67-24
Call Confirm
send-packet action, X.25 Layer 3, sending
"short" version without addresses and
facilities, 34-33
sent down (as primitive) to Layer 2, 30-9
Call Request
as character data, 34-9
as entry on X.25 Packet Level Setup screen,
34-6
as packet on trace display, 34-8
send-packet action, X.25 Layer 3, 34-32
sent up (as primitive) from Layer 2, 30-9
Call Request user data, may be longer than ten
character spaces, 34-7
Call reference value
adjunct to monitor/receive-message condition,
Q.931, 40-10
monitor/receive condition, Q.931, 40-10
trace column, Q.931, 40-5
Called address, entered in CALLED field on
X.25 Packet Level Setup screen, 34-6
Calling address, entered in CALLING field on
X.25 Packet Level Setup screen, 34-6
Capture, field on Trigger Setup menu, 22-11,
22-12
Capture data to screen (on/off)
ctl_capture_rd, C routine, 59-9
ctl_capture_td, C routine, 59-8
Layer 1 action, 28-14
Capture Memory, field on Record Setup menu,
11-4, 11-6, 11-10, 12-8
Capture memory
See also Data capture; Recording data
Freeze key, 3-11

Index to Part I and Part II
Care mask, 60-1
C device for isolating bits in a variable, 57-8
masking for status of given EIA lead, 60-2
masking to detect EIA lead change, 60-1
Carriage Return, produced by operation of
CTRL and M keys, 29-1
Carrier losses, T1 statistics display, 49-31
Cause byte
adjunct to Restart, Reset, Clear, and Reg
Confirm actions, 34-36
adjunct to Restart, Reset, Clear, and Reg
Confirm conditions, 34-20
listed for Reset, Clear, and Reg Confirm
packets, 34-22
listed for Restart packet, 34-21
listed for Send Clear actions, 34-37
CCITT
format, SS#7 layer 3, 43-4
Open Systems Interconnection model,
20-5, 20-8
See also Layers
CCSS#7. See SS#7
CD
available for triggering, 28-5
field on Interface Control menu, 10-10,
10-14, 10-15
field on RS-232 Interface Control menu, 9-4,
9-6
CD on/off, Layer 1 Emulate DCE action, 28-10
CD-off delay, 10-16
CD-on delay, 10-16
Chaining, of programs via Load Program action,
27-19
Change Directory, File Maintenance, menu
selection, 13-16
Change idle character, Layer 1 action, 28-12
Channel, ISDN, ISDN Interface Setup selection,
48-10
Channel mode
G.703, field on Interface Control menu,
50-22
G.703 BERT, 9-40
T1 BERT, 9-27
Channel number
G.703, field on Interface Control menu,
50-23
T1 Interface Control selection, 49-24
Char
C data type, 56-13
subfield on Line Setup menu, 4-8
Character
C constant, 57-7
conversion for display, 56-16, 57-8
received, detected in C, 59-5
types, data versus special characters in C,
59-5
Character buffer, 11-4
attributes less flexible than in Display Window,
59-15
capacity, 5-27
data, 11-3
enhancement attributes carried in high byte of
event word, 59-6
playback, 11-4
recording, 11-10
storage capacity, 1-12
writing to, 59-15, 59-19
Character data
buffer correlation with trace data, 5-26
display of
accessed by DATA softkey, 5-7
dual line, 5-8
single line, 5-8
Character field, defined, 34-7
Circuit Identifier Code (CIC), SS#7 Layer 3,
43-6, 43-10
Clear, editor command, 26-5
Clear key, 26-4
in menu fields, 3-6
in spreadsheet, 3-7
Clear path, emulate–mode action, X.25 Layer
3, 34-41
Clear statistical accumulator values, 27-15
Clear statistical counter values,
layer–independent action, 27-10
Clear statistical timer values, layer–independent
action, 27-11
Clock
field on Line Setup menu, 9-16
in BERT testing, 9-16
signal, 1-13
See also Speed
time–of–day, 1-14
See also Date/Time Setup
Clock Source, field on Line Setup menu, 4-10
COMMON, T1 test access point, 49-9
CONNECT IND primitive, example on
spreadsheet, 30-7
CONNECT REQ primitive, example on
spreadsheet, 30-8
in C, 63-12

Index to Part I and Part II
Code
conversion charts, D1-2—D1-14
field on Line Setup menu, 4-5, 8-8, 9-15, 9-16
significance in BERT testing, 9-15
standard codes, 4-5
user-defined, D3-1
Coding type, G.703, Interface Control selection, 50-19
Color, applied to RGB output, not to plasma screen, 61-1
Color display
color graphics, 16-6
connectors for external monitors, 16-3
miscellaneous utilities, 16-3—16-6
selectable options, 16-4—16-5
background color, 16-5
blink, 16-5
color, 16-5
toggle control of, 16-5—16-6
Command, field on File Maintenance menu, 3-6
Command addressing
adjunct to receive condition, X.25 Layer 2, 33-16
adjunct to send-frame action, X.25 Layer 2, 33-24
Comment, 24-13
debugging tool, 24-14
delimiters, 24-13
in C region, 53-10
length of, 24-13
location on spreadsheet, 24-13
purpose of, 24-14
valid characters, 24-13
Common Channel Signalling System #7. See SS#7
Compilation
compilation automatic during object-code save, 13-15
error diagnostics, 2-16
fields that can be modified without causing recompile, 2-16
rerun without recompiling, 2-15
seven phases, 2-15
Compile, File Maintenance
compiles contents of file or spreadsheet, 13-21
compiling spreadsheet generates #pragma hooks, 13-21
menu selection, 13-21
Condition clause, C construction corresponding to trigger, 53-8
may contain multiple expressions, 53-9
Conditions
EIA, fails to come true, 2-20
in C, 53-7
Layer 1, 28-1
layer-independent, 27-3—27-7
counters in linkable-object files, 24-12
flags in linkable-object files, 24-13
Protocol Spreadsheet
namng requirements, 27-1
programming block, 24-8
comments in, 24-13
rules for combining conditions, 27-2
transitional vs. status, 27-2, 27-6
Confirm primitives, 30-5
Connectors
back panel, 1-7
interface specifications, I-1—I-13
power, back panel, 1-7
RGB, 16-3
RS-170 video, 16-3
RS-232 printer connector, 14-3
Test Interface Module, 1-10—1-16
Constants, 25-3—25-7
expansion of, 25-7
fox message, 29-4
in C string, 56-22
in Receive string condition, 21-8
in spreadsheet string, 29-3
legal names of, 25-4
nesting of, 25-6
Protocol Spreadsheet, 20-8
programming block, 24-8
comments in, 24-13
referencing, 25-5
scope of, 25-4
transmitting, 29-4
Control characters
data-entry of, 29-1
enhancement of via bit mask, 5-12
Control leads
See also EIA leads
playback, 2-18
of bit-image data, 2-18, 7-5
of character data, 2-17, 7-5
Conversion specifiers
in C routines, 61-10
table of C conversion specifiers, 61-37
Copy
editor command, 26-6
File Maintenance, menu selections, 13-17

Index to Part I and Part II
Counter accumulated, 27–15
action
Protocol Spreadsheet, 17–3
Trigger Setup menus, 17–3
condition
Protocol Spreadsheet, 17–4, 27–5
when used in linkable–object files, 24–12
Trigger Setup menus, 17–4, 21–11
identified by name on statistics screen, 17–4
layer–independent action, 27–9
maximum value vs. maximum stat display, 27–6
may be identified on statistics screen following run, 17–6
printing line of tabular statistics for, 27–16
relational operators, 27–5
shared between spreadsheet and Trigger Setup menus, 27–9
transmitted, 29–4
Cover removal. See Field Service replacement. See Field Service CPM board block diagram, 2
color connections for, J4–2
detachment of connectors, J5–2
firmware replacement. See Field service hardware architecture, J3–3, J3–6–J3–7
view as a component, J2–2
CR control character, 3–5
CRC Mode, field on BCC Setup menu, 8–8, 8–11, 8–13, 8–15
CRC–4 errors, G.703, statistics display, 50–28
CRC–6 errors, T1 statistics display, 49–30
CRT/RGB connector, 1–9
CTS
available for triggering, 28–5
field on Interface Control menu, 10–10, 10–12, 10–14, 10–15
field on RS–232 Interface Control menu, 9–4, 9–6
CTS on/off, Layer 1 Emulate DCE action, 28–10
CTS–off delay, 10–15
CTS–on delay, 10–15
Current, column on Tabular Statistics screen, 27–9, 27–10
Current Date, field on Date/Time Setup menu, 15–4
Current directory
File Maintenance screen, 13–10
filing system, 13–4
Current Time, field on Date/Time Setup menu, 15–4
Cursor
positioning the cursor in the Display Window, pos_cursor, C routine, 61–8
restoring cursor to previous position, restore_cursor, C routine, 61–22
Cursor keys
in spreadsheet, 3–8
may be programmed in the Display Window, 3–11, 5–22, 61–4
on menu screens, 3–8
used to control playback speed, 3–11
D
%d, C conversion specifier, converts char to short, 56–14
D, trace column, X.25 Layer 3, 34–13
D bit
adjunct to monitor/receive–packet condition, 34–18
adjunct to send–packet action, 34–36
position diagrammed, 34–13
value selectable for Call and Call Confirm packets as well as Data, 34–19
D channel, ISDN, 48–3, 48–4
D4, T1 superframing, 49–23
D4 superframes, T1 frame structures, 49–32
DATA, field on X.25 Packet Level Setup screen, 34–7
DATA IND primitive, example on spreadsheet, 30–9
DATA REQ primitive, example on spreadsheet, 30–9
Data
See also Character data
bit–image data, 11–3
buffered automatically in FEB, 7–3
character–oriented, 11–3
in IL buffer, 63–4
Data acquisition tracks, 12–3
Data capture, 2–17
See also Playback; Recording data manual control of, 11–11
RAM, data storage, 1–12
trigger control of, 11–10

Index to Part I and Part II
Data compression, SS#7, Layer 1, 41-4

Data display
black and white enhancements, 16-6
C character types, data versus special characters, 59-5
character buffer 16-bit word, 59-16
data event-word, 59-15
enhancements, created by attribute bits in high byte of event word, 59-6
special-receive word, 59...

Data event-word, data display, 59-15

Data Path
G.703, field on Interface Control menu, 50-22
T1, field on Interface Control menu, 49-24
T1 Interface Control selection, 49-23

Data packet
monitor/receive condition, X.25 Layer 3, 34-15
translates into two C variables, 72-1
send action, X.25 Layer 3, 34-31

Data plus leads
display available during playback, 7-5
display enabled/disabled by FEB setup, 7-5
display of, 5-9
control leads selected for, 5-9
RS-449, 45-7
V.35, 44-7
X.21, 46-7
softkey access, 5-9
X.21, 46-7
failure of leads to transition, 2-21

Data source, connection to, 1-16

Data speeds, selectable, C-1

Data Transfer, Disk Maintenance, menu selection, 12-7

Data to Record, field on Record Setup menu, 11-6

Data transfer
INTERVIEW 5, 10, 15 PLUS data, 12-10—12-16
prior to playback, 11-4

Data—character buffer, 63-3
See also IL buffer

Data—start offset, in PDU, 63-5

Data—transmit delay, 10-14

Date/Time Setup, 15-3—15-4
menu selections, 15-2
set date, 15-3
set time, 15-3

Day of month, as trigger condition, 27-6

DCE, monitor condition
LAPD, 39-9
Layer 1, 28-3
Q.931, 40-9
SDLC, 35-11
SS#7 Layer 2, 42-6, 43-7
X.25 Layer 2, 33-11
X.25 Layer 3, 34-15

DDCMP
Layer 1 package, 37-1

Decimal
conversion for display, 56-16, 57-8
conversion specifier, 61-11
in C, constant, 57-7

Decimal field, defined, 34-6

Decrement counter, layer-independent action, 27-9

Decrement flag byte, as 16-bit binary counter, layer-independent action, 27-14

Default menus, how to change, 2-6
#define, C preprocessor directive, 56-5, 61-40
example, 67-15

Degraded minutes
G.703 BERT statistics, 9-44
T1 BERT statistics, 9-31

Delete
editor command, 26-5
File Maintenance
menu selection, 13-20
remove, C routine, 65-32

Delete Char key, 3-6, 26-4
Delete Line key, 3-7, 26-4

Destination Point Code (DPC), SS#7 Layer 3, 43-10

Diagnostic byte
adjunct to Restart, Reset, Clear, Diag, and Reg Confirm conditions, 34-23
adjunct to Restart, Reset, Clear, Diag, and Reg Confirm send actions, 34-38
entered as two hex digits, 34-23

Directories
/sys, 13-6
/usr, 2-4, 13-6
/usr/default, 2-5
absolute pathnames, 13-5
directory listings, 13-10
filing system, how to create, 13-5
naming conventions, 13-7
relative pathnames, 13-6
root (/) directory, 13-4
write-protected, 13-10
Disk, source of playback data, 4–4
Disk drives, 1–5
  current disk, filing system, 13–10
  drive references and priority, 12–3
  filing systems, moving from disk to disk, 13–5
microfloppies
  compatibility, 1–5
  write protection, 1–5
microfloppy disks, storage capacity, 1–5
Winchester hard disk, 1–7
installation. See Field Service

Disk Maintenance, 12–3–12–16
  allocating disk space, 12–3
command
  Data Transfer, 12–7
  Disk Summary, 12–6
  Duplicate Disk, 12–9
  Format Disk, 12–5
  INT 10, 12–10–12–16
data acquisition tracks, 12–3
data transfer, 13–7
description of disks, 12–3
initializing system, INTERVIEW 7000, 2–3
installing new system software, 2–7
menu selections, 12–2, 12–4–12–16
overview, 2–14

Disk Name, subfield on Disk Maintenance menu, 12–5
Disk No
  field on Line Setup menu, 11–4
  field on Record Setup menu, 11–6, 11–10
Disk Number, subfield on Disk Maintenance menu, 12–5
Disk Summary, Disk Maintenance, menu selection, 12–6

Display, plasma, 1–3

Display Abort
  field on Line Setup menu, 21–6
  subfield of BOP Format, 4–9
  subfield on Line Setup menu, 4–9

Display Idle
  field on Line Setup menu, 28–11
  subfield on Line Setup menu, 4–8, 5–10, 7–4
  cannot display idle if suppressed in FEB, 7–4

Display Mode
  current display mode tracked via C variables, 66–1
  field on Display Setup menu, 5–7, 5–17
  information on current display stored in C variable, 61–1

Display Setup
  menu selections, 5–2
  traces, 2–21
Display Setup screen, overview, 2–11
Display States, field on Display Setup menu, 5–19, 27–19
Display Window
  array of 1,088 long integers, 61–35
  cursor keys under programmer's control, 5–22, 61–4
display mode, 5–22

DL data, 30–10

DL_CONNECT CONF
  entered manually at Layer 2 to "fool" Layer 3 into thinking there is a link, 34–46
  primitive forced up by user program at Layer 2, 30–6
  primitive sent upward by Layer 2 to confirm the link, 30–9

DL_CONNECT IND
  action primitive at Layer 2, 30–3
  condition primitive at Layer 3, 30–3

DL_CONNECT REQ
  automatic when data primitives are passed down by Layer 3, 31–1
  primitive passed down from Layer 3, 30–8
  primitive triggered automatically by Layer 3 Send action, 30–6
  sent down automatically at Layer 3 if Layer 2 inactive, 34–44, 34–46

DL_DATA
  macro, 29–5
  primitives between Layers 2 and 3, 30–9

DL_DATA IND
  condition at Layer 3, 39–31
  primitive code for, 55–4

DL_DATA REQ, sent down automatically by Send or Resend action at Layer 3, 34–44

Don't Care key, 3–6
  in Receive string condition, 21–7

Done key
  on menu screens, 3–9
  used to change real-time display softkeys, 5–6
  used to exit softkey rack in spreadsheet, 3–9
  used to move from Conditions to Actions, 33–21, 34–29, 35–21, 39–19

Double parens, 3–5
  in Protocol Spreadsheet string, 29–3
  in Receive string condition, 21–8

Down Arrow key, 26–4
Drive, field on Layer Setup screen, 6-3
Drop-and-insert mode
G.703 transmissions, 9-38, 50-6
Interface Control selection, 50-21
ISDN transmissions, ISDN Interface Setup selection, 48-10
T1 transmissions, 9-26, 49-5
Interface Control selection, 49-22

DS1
T1 circuits, 49-3
T1 physical interface, 49-6

DSR, available for triggering, 28-5
DSR on/off, Layer 1 Emulate DCE action, 28-10
DTE, monitor condition
LAPD, 39-9
Layer 1, 28-3
Q.931, 40-9
SDLC, 35-11
SS#7 Layer 2, 42-6, 43-7
X.25 Layer 2, 33-11
X.25 Layer 3, 34-15

DTR
available for triggering, 28-5
enables/disables B bus (RS-485), 47-7
DTR on/off, Layer 1 Emulate DTE action, 28-10
Dual floppy disk drive bracket, changing. See Field Service
Dual-channel testing, ISDN, 48-4
Duplicate Disk
Disk Maintenance, menu selection, 12-9
installing new system software via the DUPDISK command, 2-7

E

EBCD
default BCC parameters, 8-9
hex-to-display conversion table, D2-3
keyboard-to-EBCD conversion table, D1-4
reverse bit order appropriate for, 4-11

EBCDIC
default BCC parameters, 8-8
hex-to-display conversion table, D2-3
keyboard-to–EBCDIC conversion table, D1-2

Echo program
BOP Info-field echo, 55-10
sync or async data, 55-9

Edit key, 3-7
Editing keypad, 26-3

Editor, Protocol Spreadsheet
See also Protocol Spreadsheet editor
ing editing a C program, 56-4
function keys, 26-5-26-10

EIA
condition, Trigger Setup menus, 21-9
Layer 1 conditions, 28-5
Layer 1 emulate-mode actions, 28-10
RS-485 application, 47-7
toggle conditions, fails to come true, 2-20

EIA leads
buffered or discarded in FEB, 7-3
effect on character–buffer capacity, 5-27
effect on data–plus–leads display, 7-5
effect on EIA trigger conditions, 7-3
driven on/off as trigger action, 10-9
four kinds of status indicators, 10-8
handshaking, 10-10
maintaining lead statuses during program chaining, 27-20
masking for status, 60-2
masking to detect a change, 60-1
monitoring by trigger, 10-9, 44-7
storage of, 1-12

Emulate
field on LAPD Frame Level Setup screen, 39-3, 39-4
field on SDLC Frame Level Setup screen, 35-3, 35-4
field on SNA/SDLC Frame Level Setup screen, 36-3
field on X.25 Frame Level Setup screen, 33-3, 33-4, 33-24
field on X.25 Layer 2 Setup screen, 33-16
field on X.25 Packet Level Setup screen, 34-3, 34-4
indicator on RS-485 Test Interface Module, 47-5

Emulate DCE, indicator on Test Interface Module, 10-4
Emulate DTE, indicator on Test Interface Module, 10-4

Emulate modes
effect of open breakout switch when INTERVIEW is driving signal, 10-6
effect of open switch when INTERVIEW is receiving signal, 10-6
installing connectors for, 10-4, 49-7, 50-9
setting up, 4-3

Emulation, connectors used, 1-10
Enable CRC-4, G.703, field on Interface Control menu, 50-23

Index to Part I and Part II
End, editor command, 26-5
End/Incl, field on BCC Setup menu, 8-12
End/N/Incl, field on BCC Setup menu, 8-12
End/Staystarted/Incl, field on BCC Setup menu, 8-12
End/Staystarted/N/Incl, field on BCe Setup menu, 8-12
Enhance
field on Display Setup menu, 5-12, 11-5
field on Miscellaneous Utilities menu, 16-4, 16-5
field on Trigger Setup menu, 22-7
Enhance character data
as Layer 1 action, 28-13
ctl_enhance_rd, C routine, 59-8
ctl_enhance_td, C routine, 59-7
on Display Setup, 5-12
Enhance selected trace rows
LAPD action, 39-29
l2_enhance, C variable, 77-8
map to color display, 33-33, 34-42, 35-30, 39-30, 40-12, 42-10, 43-11
Q.931 action, 40-12
l3_enhance, C variable, 78-4
SDLC action, 35-30
l2_enhance, C variable, 73-8
SNA action, 36-3
l2_enhance, C variable, 74-4
SS#7 Layer 2 action, 42-10
l2_enhance, C variable, 79-4
SS#7 Layer 3, l3_enhance, C variable, 80-7
SS#7 Layer 3 action, 43-10
X.25 Layer 2 action, 33-32
l2_enhance, C variable, 71-8
X.25 Layer 3 action, 34-42
l3_enhance, C variable, 72-9
Enhancements
black and white, 16-6
color, 16-3-16-6
low intensity, 28-13
must be turned off as well as on at Layer 1, 28-13
used to highlight Bisync addresses, 28-13
Enter State
in C, 53-3
layer-independent condition, 27-3
ERR INJ
G.703 BERT, run-time function key, 9-45
T1 BERT, run-time function key, 9-32
Error Injection Rate, field on BERT Setup menu, 9-14, 9-22
Error messages
issued by C translator, A2-1-A2-6
issued by compiler, A3-1-A3-16
locating errors, 56-4
issued by translator, locating errors, 56-4
Error-free seconds
BERT counter, 9-21
G.703 BERT statistics, 9-44
T1 BERT statistics, 9-31
T1 statistics display, 49-30
Error-free secs, G.703, statistics display, 50-29
Errors
in BERT
automatic injection, 9-14
manual injection, 9-18
recoverable, 2-17
ESF, T1 superframing, 49-23
ESF errors, T1 statistics display, 49-30
ESF superframes, T1 frame structures, 49-32
Event
C type specifier, 56-4
program interrupt
two events never simultaneous, 54-3
various possible origins, 54-2
Event variable
in C, may be created by user, 57-6
one used by C translator for every spreadsheet condition, 53-10
Execute key, 3-6
Extern, C storage-class specifier, cannot be declared below Test level, 53-15
External monitors
control of enhancements
black and white, 16-6
color, 16-3-16-6
RGB color video connector, 1-9, 16-3
RS-170 video connector, 1-10, 16-3
Extra bits, G.703, field on Interface Control menu, 50-24

F

FACILITIES, field on X.25 Packet Level Setup screen, 34-7

Facilities
adjunct to send-call action on Protocol Spreadsheet, X.25 Layer 3, 34-33
relation to FACILITIES field on X.25 Packet Layer Setup screen, 34-33
length byte handled automatically, 34-7

Index to Part I and Part II
Failed seconds
  G.703 BERT statistics, 9–44
  T1 BERT statistics, 9–31

Fan
  back panel, 1–8
  clean filter to prevent overheating, 1–9

Fast event, C type specifier, 56–4

Fault
  in half-duplex BERT, 9–9
  on noisy circuit, 9–9

Faults
  G.703 BERT statistics, 9–44
  T1 BERT statistics, 9–31

FDL, T1 transmissions. See Frame Data Link

FEB. See Front end buffer

FEB board
  block diagram, 2
  connections for, J4–2
  hardware architecture, J3–3, J3–11–J3–12
  view as a component, J2–2

FEB Setup screen, T1 options, 9–25

Field Service, J–1–J–2
  boards
    installation, J3–3
    removal, J2–3–J2–6
  CPM board, firmware replacement,
    J5–1–J5–6
  cables
    connection, J3–15
    disconnection, J2–5
  cover
    removal, J2–3
    replacement, J2–6
  dual floppy disk drive bracket, changing,
    J7–1–J7–4
  hard drive, installation, J6–1–J6–6
  logic board
    installing, J3–1–J3–3
    removal, J2–1–J2–6
  MPM board, S1 switch settings, J3–14
  mux board, firmware replacement, J4–1–J4–5
  PROMs, exchanging, J5–4
  self tests, J3–15
  static electricity elimination, J1–3

File Maintenance, 13–3–13–16
  absolute pathnames, 13–5
  C routines, 65–31–65–38
  creating new directories, 13–5
  current directory, 13–10
  default directory, 13–4
  directories, 13–4
  File Maintenance screen, 13–9
  current disk, 13–10
  directory listings, 13–10
  overview, 2–14

files
  data files, 13–7
  description of, 13–6
  linkable-object files, 13–7, 24–10, 56–8
  compiled contents of spreadsheet, 24–10
  loading and saving, 13–3
  marking files, 13–13
  moving from file to file, 13–4
  object files, 13–6, 13–14
  Protocol Spreadsheet, 13–7
  program files, 13–6, 13–14
  selecting files for command execution, 13–12
  setup files, 13–6, 13–14
  types, 13–12
    get_file_type, C routine, 65–36
    set_file_type, C routine, 65–34
  unmarking files, 13–13
  menu selections, 13–2
  Change Directory, 13–16
  Compile, 13–21
    compiles contents of file or spreadsheet, 13–21
    compiling spreadsheet generates #pragma hooks, 13–21
  Copy, 13–17
  Delete, 13–20
  how to execute, 13–13
  Load, 13–14
  Make Directory, 13–16
  Print, 13–19
  Rename, 13–19
  Save, 13–15
  View, 13–18
  Write Enable, 13–19
  Write Protect, 13–20
  moving from disk to disk, 13–5
  naming conventions, files and directories, 13–7
  pathnames, the use of periods, 13–8
  relative pathnames, 13–6
  root (/) directory, 13–4
  the /sys directory, 13–6
  the /usr directory, 13–6
  write-protected files, 13–10

Fill-in frame, monitor/receive condition, SS#7
  Layer 2, translates into two C variables, 79–3

Find, editor command, 26–9

FLG, flag, trace column, Q.931, 40–5

Flag key, 3–6
  in Receive string condition, 21–7
  valid in Suppress field, 5–11
Flags  
common to all tests and layers, 27-13  
condition, Trigger Setup menus, 21-10  
layer-independent action, 27-13  
as toggling mechanism, 27-14  
layer-independent condition, 27-7  
when used in linkable-object files, 24-13  
transmitted, 29-4  

Force data-packet transmit, 34-45  
Force receivers out of sync, Layer 1 action,  
28-11  

Format, field on Line Setup menu, 4-6, 4-9,  
9-15, 21-6, 28-4, 28-12  
significance in BERT testing, 9-15  

Format Disk, Disk Maintenance, menu  
selection, 12-5  

Fox message, 28-8, 33-28, 34-38, 35-26,  
39-25  
"forced down" from Layer 3 to the Layer 1  
interface, 30-6  
in BERT, 9-7  
really a built-in constant, 29-4  

Frame Data Link  
T1 Interface Control selection, 49-24  
T1 transmissions, with ESF framing, 49-35  

Frame fields  
diagrammed for LAPD, 39-7  
diagrammed for SDLC, 35-8  
diagrammed for X.25, 33-8  

Frame resync criteria, G.703, field on Interface  
Control menu, 50-25  

Frame sent, emulate-mode condition  
LAPD, 39-17  
frame_sent, C variable, 77-7  
SDLC, 35-18  
frame_sent, C variable, 73-7  
should be used along with More/No More to  
Resend, 33-20, 35-20, 39-18  
X.25 Layer 2, 33-18  
frame_sent, C variable, 71-7  

Framed mode  
G.703 BERT, 9-41  
T1 BERT, 9-28  

Frames received  
G.703, statistics display, 50-27  
T1 statistics display, 49-29  

Framing bits  
ISDN transmissions, 48-3  
recording of, 49-26  
T1 Interface Control selection, 49-24  
T1 transmissions  
with D4 framing, 49-33, 49-34  
with ESF framing, 49-35  

Framing error  
fevar_frm_error_td and fevar_frm_error_rd, C  
events, 59-2  
monitor/receive condition, Layer 1, 28-4  
Framing errors, G.703, statistics display, 50-28  
Framing mode, T1, Interface Control selection,  
49-23  
Framing pattern sequence (FPS), T1  
transmissions, with ESF framing, 49-35  

Freeze key, 3-11  
contrasted with Capture On/Off trigger action,  
3-11  
Freeze LED, 10-4, 46-4, 48-6, 49-9, 50-11  
front panel, 1-5  

Freeze mode  
current mode status stored in C variable, 61-2  
in BERT, 9-17  
initiated by trigger. See Capture data to screen  
(on/off)  
parallel cursor movement during, 5-26, 33-6,  
34-8, 35-6, 36-4, 40-4, 42-4, 43-4  
Frequency selection, back panel, 1-8  

From  
field on Disk Maintenance menu, 11-4  
subfield on Disk Maintenance menu, 12-7  

From Disk Number, subfield on Disk  
Maintenance menu, 12-9  

Front End Buffer Setup, menu selections, 7-2  

Front end buffer  
applies to playback of bit-image data, 2-18  
Idle Suppress field, does not apply to  
playback of bit-image data, 2-18  
on playback path of bit-image data, 2-18,  
7-5, 7-6  
setup  
effect on character-buffer capacity, 5-27  
effect on data-plus-leads display, 5-9  
X.21, 46-7  
may inhibit EIA activity (except data),  
10-7  
time ticks and EIA leads, storage of, 1-12,  
2-17  

Front panel, 1-3  
function keys, 1-4  
LED overlay, 1-5  
LED's, 1-4, 2-18  
U/A, 1-11  
plasma display, 1-3  
FT Errors, T1 statistics display, 49-30
FT/FS Errors, T1 statistics display, 49–30
Full duplex handshaking, 10–12, 10–15
Function keys, 1–4

G

G.703, 50–1–50–17
aggregate data capture, 50–5
BERT, 9–35–9–46
automatic error injection rate, 9–42
bit errors, 9–44
block size, 9–42
blocks in error, 9–44
blocks received, 9–44
blocks sent, 9–44
channel mode, 9–40
degraded minutes, 9–44
error–free seconds, 9–44
failed seconds, 9–44
framed mode, 9–41
invert, 9–41
number of faults, 9–44
pattern, 9–41
run–time function keys, 9–45
Setup screen, 9–39
Statistics screen, 9–43
setting up, 9–36
severely errored seconds, 9–44
test length, 9–42
test seconds, 9–44
unframed mode, 9–40
Bipolar violations, 50–19
CCITT recommendation, 50–3
T1, 49–3
channel 0, 50–4
channel data capture, 50–4
data displays, 50–7
drop–and–insert, 9–38, 50–6
emulation modes, 50–6, 50–9
framing characteristics, 50–4
Interface Control screen, 9–37, 50–2, 50–17
begin CAS MF w/frame containing frame
align. signal, 50–24
CAS MF resync criteria, 50–25
CAS MF sync criteria, 50–25
channel mode, 50–22
channel number, 50–23
coding type, 50–19
data path, 50–22
enable CRC–4, 50–23
extra bits, 50–24
frame resync criteria, 50–25
include channel 16, 50–26
international bits, 50–25
line clock select, 50–21
line impedance, 50–19
national bits, 50–25
receiver gain, 50–19
signalling type, 50–23
termination, 50–19
transmit mode, 50–20
xmit distant MF alarm, 50–22
xmit remote alarm, 50–22
xmit signalling all 1’s, 50–22
line conditions, statistics display, 50–29
monitor mode, 50–6
multiframe structure, CAS, 50–31
Primary Rate ISDN, 50–7
physical connectors, 50–8
record setup, 11–9
setting up menus for testing, 50–16
signalling bits
with CAS signalling with channel 16, 50–32
with CCS/CAS signalling with CRC–4, 50–34
statistics display, 50–26
as alternate run–time display, 50–30
BPV’s received, 50–28
BPV–free seconds, 50–28
CRC–4 errors, 50–28
error–free secs, 50–29
Frames received, 50–27
framing errors, 50–28
G.703 line conditions, 50–29
sync loss time, 50–28
sync losses, 50–28
test seconds, 50–27
Test Interface Module, 50–2, 50–8, I–19
signal direction, 50–10
Transmit mode, 9–38
testing and layer protocols, 50–17
testing configurations, 50–11
transmission speeds, 50–3
G.703 BERT, testing modes, 9–39
G.703 line conditions, G.703, statistics display, 50–29
G.704, CCITT recommendation
G.703 framing, 50–4
T1 superframing, 49–4
G703STA, G.703 BERT, run–time function key, 9–46
GBM board
block diagram, 2
connections for, J4–2
hardware architecture, J3–3, J3–5–J3–6
view as a component, J2–2
General operation, 2–1–2–25
  boot-up program, creating a user interface, 2–4
  changing default menus, 2–6
  common problems, 2–20–2–22
  data capture, 2–17
  front end buffer, 2–17
  front end buffer, on playback path of bit-image data, 2–18
  initializing system, INTERVIEW 7000, 2–3
  installing new system software, 2–7
  overview of menus, 2–9–2–14
  power up, 1–16, 2–1
  rerunning a test program, 2–15
  running a test program, 2–15
  running default program, 2–5

Give data, 29–5
  LAPD, 39–26
  LAPD action, l2_give_data, C routine, 77–9
  SDLC action, 35–27
  l2_give_data, C routine, 73–9
  X.25 Layer 2 action, 30–10, 33–28
  l2_give_data, C routine, 71–9
  X.25 Layer 3 action, 34–39
  l3_give_data, C routine, 72–10

Glitch catcher, 10–8, 44–6, 45–6, 46–6

Glossary, abbreviations, B–1–B–10

Go-error, editor command, 26–10

Go-line, editor command, 26–9

Good BCC
  adjunct to monitor-frame condition
    LAPD, 39–13, 39–15
    dce_good_bcc, C variable, 77–1
    SDLC, 35–14, 35–16
    dce_good_bcc, C variable, 73–1
    X.25 Layer 2, 33–14, 33–15
    dce_good_bcc, C variable, 71–1
  adjunct to send-frame action
    LAPD, 39–25
    SDLC, 35–26
    X.25 Layer 2, 33–28
  adjunct to send-string action, Layer 1, 28–8
  appended to transmit string, 8–5
  as condition, 8–5
  default BCC for frames, 33–28, 35–26, 39–25
  monitor/receive condition, Layer 1, 28–4
    operational only when Rcv Blk Chk enabled, 28–4
  overlay, 8–3
  parameters defined on BCC Setup menu, 8–6

  placed inside of state loop, 52–16
  used to move program control to a different state-label, 53–2, 53–4

Graphical Statistics menu, color graphics, 16–6

H

Half duplex handshaking, 10–12, 10–15

Half-duplex BERT, 9–6

Handshake
  default for remote port, 67–20
  field on BERT Setup menu, 9–5, 9–6, 9–7, 9–13
  relation to Interface Control screen, 9–7

Hard drive, installation. See Field Service

Hardware, 1–3–1–16
  back panel, 1–7
  block diagram of architecture, 2
  clock, 1–13
  current hardware configuration stored in unit_config C variable, 66–1
  disk drives, 1–5
  hard drive, J6–1–J6–6
  front panel, 1–3
  interior layout, J2–3–J2–5
  keyboard, 1–3
  operating environment, 1–14
  operating positions, 1–14
  physical dimensions, 1–3
  power up, 1–14
  storage capacity, 1–12

Hazardous areas, J2–3

HDB3. See High Density Bipolar 3

Hex, subfield on Trigger Setup menu, 22–7, 22–8

Hex key
  for hexadecimal data entry, 3–6
  valid inside C strings, 56–22
  for hexadecimal translation of line data, 5–14
  LED display on keycap, 5–14

Hexadecimal
  C constant, 57–7
  conversion for display, 56–16, 57–8, 61–20
  conversion specifier, 61–11

Hexadecimal code, conversion charts, D1–2–D1–14

Hexadecimal display
  in relation to order of transmission, 5–14
  turned on/off by trigger action, 5–15

Hexadecimal field, defined, 34–6

Index to Part I and Part II
High Density Bipolar 3, transmission technique, 50–3
High Outgoing Channel #, field on X.25 Packet Level Setup screen, 34–3
High speed aggregate G.703 record, 50–5 aggregate T1 record, 49–5, 49–26 data recording, 11–9 G.703 aggregate record, 50–26 optimizing performance, 2–22
Home key, 3–8, 26–4
Hook text added to top-level main, 56–12 C code in #pragma hook directive, 56–11 definition indirectly referencing routines, 56–12 may reference tasks, 56–12
Hook_type, decimal constant to identify type of #pragma hook directive, 56–11 Host Port, SNA frame setup selection, 36–3
HRD/usr/user_intrf affect on Start Up screen, 2–2 boot-up program, creating a user interface, 2–4

I, intensity, field on Graphical Statistics menu, 18–7
Idle change idle–line character, 28–12 display in relation to Outsync action, 28–11 display used for visual record of time intervals, 7–4 displaying for visual record of lead timings, 5–10 displaying synchronous idle, 4–8 idle_action, C routine, 59–13 retained/discarded on FEB Setup menu, 7–3, 7–4 selecting transmit idle, 4–9 voltage not affected by inverted polarity, 4–12 Idle line action, used in X.21 bis, 32–4 Idle Suppress, field on Front–End Buffer Setup screen, 7–4 does not apply during playback, 2–18 Idle select, T1, Interface Control selection, 49–22

Idle Timeout conditions under which timer expires, 35–4 expired, emulate–mode condition, SDLC, 35–18 field on SDLC Frame Level Setup screen, 35–3 field on SNA/SDLC Frame Level Setup screen, 36–3 maximum and minimum values, 35–4
If, C statement, 58–6 nonzero expressions always true inside of if statement, 58–2 routine that returns nonzero makes if statement true, 58–6 used in Enter State conditions, 53–4
IL buffer advantage as storage medium, 55–1 and primitives, 30–3 created by DDCMP package, 37–2 creating a structure–pointer to an IL buffer, 63–6 data–character buffer, 63–3 being passed up the layers, 55–2, 63–4 downward moving, illustrated, 55–6, 63–2 identified by segment number, 55–5 number of the buffer in PDU, 63–5 pointer–list buffer, 63–3 being passed down the layers, 63–2 pointing to data inside an IL buffer, 63–5, 63–41 SDU shrinks as IL buffer moves up the layers, 55–1 string to be referenced in, 30–6 structure of, 63–3–63–4 header, 63–3 service data unit, 63–3–63–4 data, 63–4 list node, 63–4 list–header node, 63–4 upward moving, illustrated, 55–2 INFO frame monitor/receive condition LAPD, 39–10 translates into two C variables, 77–6 SDLC, 35–12 translates into two C variables, 73–6 X.25 Layer 2, 33–12 translates into two C variables, 71–6 send action LAPD, 39–20 SDLC, 35–22 used to convey DL data sent down from Layer 3, 33–23, 39–20 X.25 Layer 2, 33–22
INFO–ELEMENT, trace column, Q.931, 40–5

Index to Part I and Part II
INJ1ERR
G.703 BERT, run-time function key, 9-45
T1 BERT, run-time function key, 9-32
INT 10, Disk Maintenance selection,
12-14—12-16
INTERVIEW transfer, INTERVIEW 5, 10, 15
PLUS data, transfer from Disk Maintenance
screen, 12-10—12-16
In-band signaling, T1 transmissions, 49-23,
49-32
In/out, editor command, 26-6
#include, C preprocessor directive, 56-6, 61-23,
61-29, 61-30
stdio.h file included with all disk I/O routines,
65-3
Include channel 16, G.703, field on Interface
Control menu, 50-26
Increment counter, layer-independent action,
27-9
Increment flag byte, as 16-bit binary counter,
layer-independent action, 27-14
Indication primitives, 30-5
versus “Requests”, 30-10
Info element, Q.931, 40-5
Initial Condition, field on Record Setup menu,
11-9
Initial Phase, field on X.21 Interface Control
menu, 46-9
Initial State, field on BCC Setup menu, 8-11
Initializing system software, INTERVIEW 7000,
2-3
Injection Rate
field on BERT Setup menu, 9-19
status field on BERT results screen, 9-21
Input/output connectors, back panel, 1-9
AUXILIARY TTL, 1-9
CRT/RGB, 1-9, 1-10
internal MODEM, 1-9
PRINTER, 1-9
REMOTE RS-232, 1-9
RS-170 composite video, 1-10
Insert Char key, 26-4
in menu fields, 3-6
used to exit insert mode, 3-8
Insert Line key, 26-4
in spreadsheet, 3-7
on statistics screens, 3-7
Insert mode, 3-8, 26-4
Installing new system software, on hard disk,
2-7
Instrument, menu field on Disk Maintenance
screen, for INTERVIEW 10 transfer, 12-14
Int, C data type, 56-13
same as short int, 56-13
Integral promotion, 56-16, 61-11
Integrated Services Digital Network (ISDN),
SS#7 Layer 3, 43-22
Interactive messages, A1-1—A1-15
Interface Control menu, 10-10
in relation to Line Setup menu, 10-10
RS-485 application, 47-7
X.21, 46-8
Interface Control screen, G.703 options, 9-37
Interlayer buffer. See IL buffer
Interlayer message buffer. See IL buffer
Internal MODEM connector, 1-9
International bits, G.703, field on Interface
Control menu, 50-25
Interrupt packet, sample program to enhance
all, 34-43
Invalid frame
defined, 33-7, 35-7, 39-8
receive condition
LAPD, 39-16
SDLC, 35-16
X.25 Layer 2, 33-16
Invalid packet
defined, 34-12
receive condition, X.25 Layer 3, 34-26
invalid_packet, C variable, 72-8
Invert BCC, field on BCC Setup menu, 8-11
IPARS
default BCC parameters, 8-8
default sync pattern, 4-7
hex-to-display conversion table, D2-3
keyboard-to-IPARS conversion table, D1-6
reverse bit order appropriate for, 4-11
SY characters inappropriate for, 4-7
ISDN
See also Q.931
B channels, 48-3, 48-4
Basic Rate ISDN, 48-3
channel, ISDN Interface Setup selection,
48-10
D channel, 48-3, 48-4
dual-channel testing, 48-4
ISDN_D, 38-3
LED's front panel, 48-6

Index to Part I and Part II
line conditions, ISDN statistics display, 48–11
monitor mode, installing connectors for, 48–8
NT, network termination, 48–7
NT state, 48–13
physical devices, 48–6
RD line status, 48–13
sample Line Setup, 4–13
single-channel testing, 48–5
speaker, ISDN Interface Setup selection, 48–10
statistics display, as alternate run-time display, 48–11
TD line status, 48–12
TE, terminal equipment, 48–7
TE state, 48–11
Test Interface Module, 48–2, 48–6, I–21
testing interfaces, 48–6, 48–7
Isoc
data setup, 4–10
format in BERT, 9–16

J

JIS7
hex–to–display conversion table, D2–3
keyboard–to–JIS7 conversion table, D1–10
JIS7/JIS8, optional codes, H–1
JIS8
hex–to–display conversion table, D2–3
keyboard–to–JIS8 conversion table, D1–12, D1–13

K

Katakana, JIS7/8 optional code set, H–1
Keyboard, 1–3
See also ASCII keys; entry under each special key
condition, Trigger Setup menus, 21–12
editing keypad, 26–3
function keys, 1–4
layer–independent condition, 27–4
programming keys, 3–3–3–7
real–time keys, 3–10–3–12
soft (function) keys, 3–3
for editing, 26–5–26–10
translation tables, D–1–D–2
Keyword, 29–3

L

L, label, field on Graphical Statistics menu, 18–4
LAPD, 39–3–39–14
diagram of frame fields, 39–7
send actions, 39–19
used with ISDN D channel, 38–3
LAPD Frame Level Setup screen, 39–2
Label
C keyword, equivalent to a spreadsheet State, 53–2
C type specifier, 56–4
Last, column on Tabular Statistics screen, 17–6, 27–10, 27–11
Layer, field on Display Setup menu, 5–17, 5–19, 27–19
Layer 2, user program to force packets up to Layer 3 and down to Layer 1, 34–24
Layer Setup, 6–3–6–6
how to save, 6–5
Personality packages, 6–3
reside on user and hard disks, 6–3
Protocol Configuration screen, 6–4
protocols, select and load, 6–3
Layer Setup screen, overview, 2–14
Layers
identified on Program Trace, 5–19
in C, 53–1
Protocol Spreadsheet, programming block, 24–8
comments in, 24–13
passing data between, 2–21
program model, 20–3–20–8
LCN
adjunct to monitor/receive–packet condition, X.25 Layer 3, 34–18
allocation sequence, 34–5, 34–31
assigned dynamically on per–call basis, 34–4
column on X.25 Packet Level Setup screen, 34–4, 34–5
predefined for a particular call address (“path”), 34–4, 34–6
trace column, X.25 Layer 3, 34–12
LED’s
front panel, 1–4, 1–5, 2–18
during playback, 10–4
Freeze, 10–4, 46–4, 48–6, 49–9, 50–11
INTERVIEW status, 1–5
interface status, 1–4
not affected by FEB suppression of EIA leads, 10–7

Index to Part I and Part II
Remote, 10-4, 48-6, 49-9, 50-11
RS-232 overlay, 10-4
RS-449 overlay, 45-2, 45-4
U/A, 1-5, 1-11, 10-4, 44-4, 45-4
V.35 overlay, 44-2, 44-4
X.21 overlay, 46-4
green-red characteristics not affected by logic (polarity), 4-12
Test Interface Module, 1-11
Left Arrow key, 26-4
/lib directory, filing system, 13-21, 24-11, 56-9
Line clock select
G.703, Interface Control selection, 50-21
T1, Interface Control selection, 49-22
Line data, data capture, 2-17
Line impedance, G.703, Interface Control selection, 50-19
Line number, of cursor position in frozen Program Trace, 5-20
Line Setup, 4-3-4-4
current line setup parameters stored in unit_setup C variable, 66-1
significance in BERT testing, 9-15
Line Setup screen
menu selections, 4-2
Async, 4-13
Bisync, 4-13
ISDN, 4-13
SNA, 4-13
SS#7, 4-13
X.25, 4-13
overview, 2-11
Line utilization, programming example, 17-11
Linefeed, not valid in C string, 56-20
Linkable-object files, 13-7
#pragma hook directives in, 56-11
accessed via #pragma object, 56-8
accessed via Object block-identifier, 24-10
advantage over object files, 13-22
C code
contents of LOBJ files, 13-7
must be compatible with menu selections, 13-22
combined with spreadsheet program, 56-19
compiled spreadsheet
accessed via OBJECT block-identifier only, 13-21
contents of LOBJ files, 13-7, 13-21
must be a valid program, 13-21
contents of, 13-21, 24-10
counters or flags in, 24-12
efficiently use memory and spreadsheet,
24-12, 56-10
in /lib directory, 13-21
indirectly referencing routines, 56-10
scope of routine definitions contained in, 56-8
search rules for, 24-11, 56-8
transparent to unit configuration, 13-22
List header, beginning of linked list in IL buffer, 55-7
List node. See IL buffer
List-header node. See IL buffer
Lists, on Protocol Spreadsheet, 29-1
Load, File Maintenance, menu selections, 13-14
Load key, 3-6, 13-14
Load program
layer-independent action, 27-19
load_program, C routine, 69-12
Logical DTE/DCE
contrasted to physical DTE/DCE, 33-4
determines command and response addresses, 33-4, 33-16
determines order (ascending/descending) of LCN selection, 34-4
Long, C data type, 56-13
long routine returns a long, 58-3
Loop down
T1 BERT, run-time function key, 9-33
T1 command, 49-9
Loop up
T1 BERT, run-time function key, 9-32
T1 command, 49-9
Loop-back C/R bit, adjunct to send-frame action, LAPD, 39-22
Loop-back P/F bit
adjunct to resend-frame action
LAPD, 39-27
X.25 Layer 2, 33-30
adjunct to send-frame action
LAPD, 39-23
SDLC, 35-24
X.25 Layer 2, 33-25
Low, subfield on Trigger Setup menu, 22-7
Low Outgoing Channel #, field on X.25 Packet Level Setup screen, 34-3
LRC Parity, field on BCC Setup menu, 8-11
LU 6.2, SNA selection, 36-3

Index to Part I and Part II
M

M, trace column, X.25 Layer 3, 34-14

M bit
adjunct to monitor/receive-packet condition, 34-18
adjunct to send-data-packet action, 34-36
position diagrammed, 34-14
MAKE, T1 test access point, 49-9
Main
C function, placed at highest level of source code, 52-16
program main created by C translator, 52-1
hook text from #pragma hook directives added to, 56-12
using multiple hooks, 56-12
Maintain, field on Interface Control menu, 10-18
X.21, 46-9
Maintain bit
freeing via the _free_il_msg_buff routine, 55-5
setting via the _set_maint_buff_bit routine, 55-5
used to lock an IL buffer against reallocation, 55-5
Make Directory, File Maintenance menu selection, 13-16
mkdir, C routine, 65-33
Mark key, 26-5
on File Maintenance screen, 3-6, 13-13
used as program tab in spreadsheet, 3-9
Maximum, column on Tabular Statistics screen, 17-6, 27-10
Maximum data rates
data analysis, 1-13
data recording, 1-13
Memory, capacity, 1-12
Menus
overview, 2-9-2-14
See also Separate listing, each menu name
configuring menus, 2-10
Program Menu, 2-9
Record Setup, 11-5-11-11
Message, indicator on RS-485 Test Interface Module, 47-6
Message Buffer, field on BERT Setup menu, 9-10
Message fields, diagrammed for Q.931, 40-7
Message Signal Units (MSU’s), Layer 3, 43-6
Message type, Q.931, 40-5
MIL, field on Line Setup menu, 4-12
MIL-188, 4-12
MISC, trace column, X.25 Layer 3, 34-14
Microfloppy disks
compatibility, 1-5
storage capacity, 1-12
write protection, 1-5
Minimum, column on Tabular Statistics screen, 17-6, 27-10
Miscellaneous Utilities
overview, 2-14
with color mapping options, 16-3-16-6
See also Graphical statistics menu, color graphics
Miscellaneous Utilities screen
black and white enhancements, 16-6
color display, selectable options, 16-4-16-5
background color, 16-5
blink, 16-5
color, 16-5
controlling color displays from, 16-3-16-6
menu selections, 16-2
Mnemonics, glossary, B-1-B-10
MOD 128, 33-5, 34-4, 35-4, 39-4
MOD 8, 33-5, 34-4, 35-4, 39-4
Mode
default handshake for remote port, 67-20
field on Line Setup menu, 4-3, 9-5, 9-6, 9-22, 10-3, 10-10, 11-4, 28-10, 33-15, 34-24, 39-15, 49-7, 50-8
test mode field on Line Setup menu, 1-11
Mode of Operation
field on Frame Level Setup screen, 33-12, 34-16, 39-10
field on LAPD Frame Level Setup screen, 39-3, 39-4
field on SDLC Frame Level Setup screen, 35-3, 35-4
field on SNA/SDLC Frame Level Setup screen, 36-3
field on X.25 Frame Level Setup screen, 33-3, 33-5
field on X.25 Packet Level Setup screen, 34-3, 34-4
Modem connector
external, 1-9
internal, 1-9
Modem eliminator, patching example, 10-7
Monitor mode
installing connectors for, 10–3, 49–7, 50–8
ISDN, 48–8–48–9
not affected by position of breakout switches, 10–6
setting up, 4–3

Monitor path, upward path of IL buffer in monitor (or emulate) mode, 55–3

More to resend, emulate-mode condition
LAPD, 39–18
translated into C, 77–8
SDLC, 35–19
translated into C, 73–8
X.25 Layer 2, 33–19
translated into C, 71–8
X.25 Layer 3, 34–28
translated into C, 72–11

Move, editor command, 26–6

MPM board
block diagram, 2
connections for, J4–2
hardware architecture, J3–3, J3–12–J3–13
S1 switch settings. See Field Service
view as a component, J2–2

MPM errors, 2–17
MSG–TYPE, trace column, Q.931, 40–5

MUX board, hardware architecture, J3–3, J3–5

Multidrop handshaking, 10–12, 10–15, 10–16

Mux board, firmware replacement. See Field service

N

N, name, field on Graphical Statistics menu, 18–6

N_DATA, macro, 29–5

N_DATA IND, sent up automatically by Give Data action at Layer 3, 34–39, 34–44

Naming, of variables in C, 57–2

National bits, G.703, field on Interface Control menu, 50–25

National Format, field on SS#7 Packet Level Setup screen, 43–3

Negative exponent, meaning in BERT formulas, 9–14

Network Indicator, SS#7 Layer 3, 43–9

Network Management (NETM) Headers, SS#7 Layer 3, 43–18
NETM condition, translates into two C variables, 80–1

Newline, nonliteral used inside C string, 56–20
writes fresh blank line into trace buffer, 61–27
writes hex 0D 0A (ASCII CR-LF) to printer output, 64–9

Next Page key, 26–4
in spreadsheet, 3–8
on statistics screens, 3–8
on trace display, 33–6, 34–8, 35–5, 36–4,
40–4, 42–4, 43–4

Next state
in C, 53–4
Protocol Spreadsheet, programming block, 24–9
comments in, 24–13

No BCC, appended to transmit string, 8–6
interpreted as bad BCC, 8–6

No display, display mode, 5–28

No more to resend, emulate-mode condition
LAPD, 39–18
translated into C, 77–8
SDLC, 35–19
SDLC variable, translated into C, 73–8
X.25 Layer 2, 33–19
translated into C, 71–8
X.25 Layer 3, 34–28
translated into C, 72–11

Nonevent variables, 59–5
in C
checked when event is signalled, 57–5
true in expressions with nonzero value, 53–12

Nonliteral characters
inside C strings, 56–21
inside set_print_header strings, 64–7

Nonzero conditional expression, always true in C, 53–12, 58–2

Not Equal key, 3–6
in Receive string condition, 21–7
used in Suppress field to indicate “display only”, 5–11

NRZI, field on Line Setup menu, 4–12

Nr
acknowledging last Ns, adjunct to send-frame action
LAPD, 39–23
SDLC, 35–24
X.25 Layer 2, 33–26

Index to Part I and Part II
calculated automatically, adjunct to send-frame action
LAPD, 39–23
SDLC, 35–24
X.25 Layer 2, 33–26
repeating last Nr, adjunct to send-frame action
LAPD, 39–23
SDLC, 35–24
X.25 Layer 2, 33–26
reset, emulate-mode action
LAPD, 39–28
SDLC, 35–30
X.25 Layer 2, 33–31
trace column
LAPD, 39–8
SDLC, 35–9
staggered to indicate two separate numbering sequences, 33–9, 35–9
X.25 Layer 2, 33–9
value, adjunct to send-frame action
LAPD, 39–23
SDLC, 35–24
X.25 Layer 2, 33–26
Nr error, emulate-mode condition
LAPD, 39–16
SDLC, 35–18
X.25 Layer 2, 33–18
Ns
calculated automatically, adjunct to send-I-frame action
LAPD, 39–24
SDLC, 35–26
X.25 Layer 2, 33–27
reset, emulate-mode action
LAPD, 39–28
SDLC, 35–30
X.25 Layer 2, 33–31
same as last-received Nr, adjunct to send-I-frame action
LAPD, 39–24
SDLC, 35–26
X.25 Layer 2, 33–27
skip to correct Ns plus one, adjunct to send-I-frame action
LAPD, 39–24
SDLC, 35–26
X.25 Layer 2, 33–27
trace column
LAPD, 39–8
SDLC, 35–9
staggered to indicate two separate numbering sequences, 33–9, 35–9
X.25 Layer 2, 33–9
value, adjunct to send-I-frame action
LAPD, 39–24
SDLC, 35–25
X.25 Layer 2, 33–26
Ns error, emulate-mode condition
LAPD, 39–16
SDLC, 35–17
X.25 Layer 2, 33–17
NT
ISDN network termination, 48–7
state on ISDN line, 48–13
Null
added by compiler to terminate string, 56–19, 57–12
not valid in C string, 56–20
octal or hex version legal inside C string, 56–21
terminates execution of display and print routines, 56–21
termination overridden by %H conversion specifier, 61–10
Number of Faults, BERT counter, 9–21

O
Object, Protocol Spreadsheet, programming block, 24–8
C regions in relation to, 53–1, 52–16, 53–17
comments in, 24–13
format of, 24–11
must be used to access compiled spreadsheet, 13–21
only C regions and comments may precede, 24–8
placement of, 24–11
used to access #pragma hook routines, 56–12
used to access linkable-object files, 24–10, 56–12
Object code
contents of linkable-object files
compiled C code, 13–7
compiled spreadsheet, 13–7
accessed via OBJECT block-identifier only, 13–21
must be a valid program, 13–21
contents of object files, 13–6
loaded in automatically via Load Program action, 27–19
rerunning object version of program, 2–15
Object files, 13–6, 13–12, 13–14
compared to linkable-object files, 13–7, 13–22
not as versatile as source-code files, 13–15
use disk-space intensively, 13–15

Index to Part I and Part II
Octal
C constant, 57-7
conversion for display, 57-8, 61-20
conversion specifier, 61-11

Offset, 5-26
On Signal, layer-independent condition, 27-8
On/off (power) switch, back panel, 1-8
One, transmitting steady one, set_tcr_b, C routine, 59-14
1 OF, field on Trigger Setup menu, 21-8
One-of character list
effect of not-equal character, 21-8
monitor/receive condition, Layer 1, 28-3

OOF events, T1 statistics display, 49-30
OPT-951-01-1, J6-1-J6-6
OPT-951-22-1, optional codes JIS7/8, H-1
OPT-951-98-1, rack mount, G-1-G-4

Operating environment, 1-14
Operating positions, 1-14

Operator messages
issued by C translator, A2-1-A2-6
issued by compiler, A3-1-A3-16

Operator precedence, C language, 56-17

Operators, relational, in counter conditions, 27-5
Order of transmission
in relation to binary display, 5-15
in relation to code charts, 4-11
in relation to hex display, 5-14
in relation to pattern sync in BERT, 9-11

Originating Point Code (OPC), SS#7 Layer 3, 43-9

Origination/destination link, message-type condition, Q.931, 40-11

OSI
Layer Setup, 6-3
Open Systems Interconnection, layered programming, 20-3-20-8
See also Layers
primitives, 20-8

OSI primitives. See Primitives

Other frame
monitor/receive condition
LAPD, 39-11
SDLC, 35-12
X.25 Layer 2, 33-12
send action
LAPD, 39-21
SDLC, 35-23
X.25 Layer 2, 33-24

Other packet
monitor/receive condition, X.25 Layer 3, 34-17
send action, X.25 Layer 3, 34-33

Out of sync, status message in BERT, 9-21

Output jacks
on RS-232 TIM, 10-8
on RS-449 TIM, 45-6
on V.35 TIM, 44-6
on X.21 TIM, 46-6

Outsync
called "resync" in BERT, 9-8
Layer 1 action, 28-11
compared to Capture Off action, 28-11
outsync_action, C routine, 59-10
parameters not selectable in DDCMP, 37-1
subfield on Line Setup menu, 4-7, 4-8

Outsync Char, field on Line Setup menu, 3-6
Overlay, Test Interface Module, 1-5
Overrun, of print buffer, 64-1
minimize by suspending playback, 64-2
Overstrike mode, 3-8

P

P/F, trace column
LAPD, 39-8
SDLC, 35-10
SNA-SDLC, 36-6
X.25 Layer 2, 33-10

P/F bit
adjunct to monitor/receive-frame condition
LAPD, 39-13
SDLC, 35-14
X.25 Layer 2, 33-14
adjunct to resend-frame action
LAPD, 39-27
SDLC, 35-29
X.25 Layer 2, 33-30
adjunct to send-frame action
LAPD, 39-23
SDLC, 35-24
X.25 Layer 2, 33-25

PATH, field on X.25 Packet Level Setup screen, 34-5

Packages Loaded
column on Layer Setup menu, 33-3, 34-3,
39-3, 43-3
column on Layer Setup screen, 6-4

Index to Part I and Part II
Packet fields, diagrammed for X.25, 34–10
Packet sent, emulate-mode condition
should be used along with More/No More to Resend, 34–29
X.25 Layer 3, 34–27
packet_sent, C variable, 72–8

Parity
a consideration when entering BCC parameters, 8–11
adjustment automatic in Sync Chars field, 4–7
always the last bit transmitted, 4–11
automatic calculation of in receive sync pattern, 4–7
default value for remote port, 67–20
field on Line Setup menu, 4–6, 4–7, 9–16, 21–6, 28–4
field on Printer Setup menu, 14–4
in BERT testing, 9–15
in setup, 4–6

Parity errors
monitor/receive condition, Layer 1, 28–4
special display of, 4–6

Patching, modem-eliminator example, 10–7

Path
adjunct to receive–packet condition, X.25 Layer 3, 34–25
used in all packet types except Restart, 34–34
correspondence at different layers, 30–5, 34–34
more “programmable” than LCN, 34–25, 34–34
part of definition of data primitive, 30–5
rcvd_device_path, C variable, 72–9
tied to a set of Call Request parameters on X.25 Packet Level Setup screen, 34–5, 34–25, 34–34, 72–9

Pattern, field on BERT Setup menu, 9–7
Pattern Sync Status
field on BERT results screen, 9–20
line on BERT results screen, 9–21
Pattern sync, in half–duplex pseudorandom BERT, 9–11
two default sets, 9–13

PCM board
block diagram, 2
connections for, J4–2
hardware architecture, J3–3, J3–8–J3–11
view as a component, J2–2

PDU. See Primitive data unit; Primitives

Perc(centage), 5–26
of Program Trace buffer storing previous data, 5–20
Percentages, computed through the sampling action, 17–11

Personality packages. See Protocol packages; Protocol packages and Layers

PH_ACTIVATE REQ, sent down automatically at Layer 2 if Layer 1 inactive, 33–34, 35–32, 39–31

PH_DATA, primitives between Layers 1 and 2, 30–9
PH_DATA REQ, sent down automatically by Send or Resend action at Layer 2, 33–34, 35–32, 39–31

PH_TD_DATA IND
implemented by a set of monitor–path variables, 55–3
implemented by a set of receive–path variables, 55–3
signalled by DDCMP package, 37–2

Physical DTE/DCE
basis of Source column on trace display, 33–7, 34–12, 35–5, 36–4, 39–6, 40–5
contrasted to logical DTE/DCE, 33–4

Physical dimensions, size and weight, 1–3

Playback
control leads, 2–17, 7–5
disk data, 2–18
EIA leads, storage, 1–12
manual control of, 3–11
source of data selected on Line Setup screen, 4–4

start_rcrd_play, C routine, 65–3, 69–16
suspend_rcrd_play, C routine, 65–3, 69–17
time ticks, 2–18, 7–6
timer values not affected when time ticks enabled, 27–10
transfer of data prior to, 11–4

Pointer, in C
always 32 bits no matter what the data type, 57–11
creating a pointer, 57–11
incrementing pointers of various data types, 57–12
m_packet_info_ptr, pointer to first data byte in X.25 packet, 72–8
m_ptr_to_call_ref, pointer to Q.931 call-reference field, 78–4
m_ptr_to_info_element, pointer to Q.931 info-element field, 78–4
making a pointer to the data in a received frame, 71–8, 73–8, 77–8

Index to Part I and Part II
making a pointer to the data in an IL buffer, 57–10, 63–41
pointing with subscripts, 57–12, 57–13
rcvd_pkt_info_ptr, pointer to first data byte in X.25 packet, 72–8
rh_ptr, pointer to first byte of SNA request/response header, 74–4
ru_ptr, pointer to first byte of SNA request/response unit, 74–4
string, 56–19
structure pointer, 57–16
th_ptr, pointer to first byte of SNA transmission header, 74–4

Pointer–list buffer, 63–3
See also IL buffer

Polarity
field on Line Setup menu, 4–11
normal versus inverted, 4–12

Pound sign (#), precedes preprocessor directives, 56–5

Power connector, 1–7
Power switch, back panel, 1–8
Power up, 1–14, 2–1
self tests, 2–1
PROMs, exchanging. See Field Service

PROTSEL, Protocol Select, Layer Setup function key used to select protocol–configuration screen for a given layer, 33–3, 34–3, 43–3

Pr
acknowledging last Ps, adjunct to send–packet action, X.25 Layer 3, 34–36
calculated automatically, adjunct to send–packet action, X.25 Layer 3, 34–36
repeating last Pr, adjunct to send–packet action, X.25 Layer 3, 34–36
reset, emulate–mode action, X.25 Layer 3, 34–41
trace column
staggered to indicate two separate numbering sequences, 34–13
X.25 Layer 3, 34–12
value, adjunct to send–packet action, X.25 Layer 3, 34–35

Pr error, emulate–mode condition, X.25 Layer 3, 34–27
pr_error, C variable, 72–8
#pragma, C directive, placed inside of task definition, 52–16
#pragma hook, C preprocessor directive, 56–11
defining the hook text, 56–12
format of, 56–11

hook text added to top–level main function, 56–12
in linkable–object files, 56–11
system–generated during Compile spreadsheet, 56–11
using multiple hooks, 56–12

Preamble, field on BERT Setup menu, 3–6
Preamble characters, in half–duplex BERT, 9–7, 9–9

Precision
length of conversions in display and print routines, 61–10
size of data types, 56–13

Prev Page key, 26–4
in spreadsheet, 3–8
on statistics screens, 3–8
on trace display, 33–6, 34–8, 35–5, 36–4, 40–4, 42–4, 43–4
used to restore previous menu, 3–8

Primary (host) in SDLC, 35–4

Primary Rate ISDN, 49–6
G.703, 50–7

Primitive data unit
See also Primitives and IL buffers, 30–3
being passed down the layers, illustrated, 63–2
being passed up the layers, 55–2
illustrated, 63–4
structure of, 63–5
data length, 63–5
data–start offset, 63–5
IL buffer number, 63–5

Primitives
accessing information in, 63–5–63–11
as conditions and actions, 30–3
automatic, 31–1
at Layer 1, 30–3
below the top layer, 31–1
monitor primitives, 31–1
varies with protocol package, 31–1
currently not accessible at Layer 1, 30–3
indications versus requests, 30–10
Layer 1 not automatic, 30-3
Layers 1 through 7, listed, 30-11—30-13
layered programming, 30-3
OSI, 20—8
OSI routines
  sending primitives up and down the layers, 63—44
on spreadsheet
  indication of direction, 30—4
  indication/confirm, 30—5
  path, 30—5
  request/response, 30—5
  type, 30—4
pointing to data inside PDUs, 63—3
prefixes, 30—4
several automatic at given layer
  LAPD, 39—31
  SDLC, 35—33
  X.25 Layer 2, 33—34
  X.25 Layer 3, 34—44
shared by layers, 30—3
used for passing data macros downward, 29—5

Print
  File Maintenance, menu selection, 13—19
  layer—dependent action, 27—16
  used to check status of print buffer, 64—1
Print accumulator, layer—dependent action, 27—16
Print buffer
  overrun, 64—1
  minimize by suspending playback, 64—2
  queues unprinted text from print actions, 64—1
Print counter, layer—dependent action, 27—16
statistical log produced by, 17—9
Print key
  used to print data, 3—13
  used to print programming screens or spreadsheet, 3—6, 14—7—14—8
Print prompt, layer—dependent action, 27—18
Print server, transfers output from print buffer to printer port, 64—1
Print timer, layer—dependent action, 27—16
statistical log produced by, 17—9
Printer, 14—3—14—5
  C print routines, 64—1—64—10
  nonliteral characters inside set_print_header
  strings, 64—8
  C print structures, 64—1
  Printer Setup screen, 14—3—14—5
  menu selections, 14—2, 14—3—14—5
  characters per line, 14—5
  form feed, 14—5
  format character buffer, 14—5
  handshake mode, 14—6
  lines per page, 14—5
  new line, 14—4
  number of bits, 14—4
  number of pads, 14—5
  parity, 14—4
  print to file instead of printer, 14—6—14—7
  printer type, 14—5
  speed, 14—4
print server, transfers output from print buffer to printer port, 64—1
printing data, 14—10—14—15
  from display window, 14—16—14—17
  line data, 14—10—14—15
  program trace, 14—14—14—15
  protocol traces, 14—13—14—15
  statistics, 14—15
  user traces, 14—17
printing disk files, 14—17—14—18
printing static displays, 14—7—14—8
  Layer Setup screen, 14—8
  Protocol Spreadsheet, 14—8
  program menus, 14—8
  setup menus, 14—8
  Trigger Setup screens, 14—8
  RS—232 printer connector, 14—3
  special characters, data, display of, 14—10
  special characters, menus, display of, 14—7
  spreadsheet control of, 14—17
  unprinted text queued in print buffer, 64—1
Printer connector, 1—9, 14—3, 64—1, I—3
Program files, saving and loading, 19—6
Program key, 2—4, 3—4
  unit unexpectedly enters Run mode, 2—20
Program trace, 5—18
  #pragma_tracebuf, 5—18, 61—26
  as customized protocol analysis, 27—19
  as debugging tool, 27—19
  buffer containing 4096 characters, 61—23
  buffer may be scrolled through in Freeze mode, 5—18, 61—23
  buffer size may be increased, 5—18, 61—26
  C routines, 61—27
  generated by trace actions on the Protocol Spreadsheet, 27—18
  one of eight trace buffers, 61—23
  printing, 14—14
  run—mode softkey available if Trace action invoked or if state trace requested, 5—18
  sample trace, 5—18, 5—20
  selecting state names from via Display Setup, 5—19, 27—19
  specific to Layer/Test selected on Display Setup, 5—19, 27—19

Index to Part I and Part II
structures, declared in trace_buf.h
#include
file. 61-23
trace_buf, C structure, 61-23
trace_buffer_header, C structure, 61-23

Programming
concepts of
branching (changing states), 20-3, 24-9
OSI layers, 20-3–20-8
simultaneous tests, 20-3–20-4
states, 20-3
three-tiered design, 19-1–19-6
program structure, Protocol Spreadsheet, 24-6–24-7

Prompt
blanking the entire prompt line, 58-4
display_prompt, C routine, 61-3, 61-21
field on Trigger Setup menu, 22-3
layer–independent action, 27-12
most recent prompt retained in Display Window, 61-3
printing, 27-18
prompt line never accessed by trace routines, 61-4
using backslash and double-quote characters inside of, 27-12

Protocol header
applied to user data by Send action, 30-6
not applied if Data Req primitive used instead of Send, 30-7
Protocol hex, user program to convert X.25
headers to hexadecimal, 33-35
Protocol packages, 6-3
general description, 19-3–19-4, 20-7–20-8
user disk, 6-3
Protocol Spreadsheet
comments, 24-13
constants, 20-8, 25-3–25-7
creating and editing, 19-4–19-5
See also Protocol Spreadsheet editor. See Protocol Spreadsheet editor files
reading and writing, 26-6–26-10
saving and loading, 19-6
function key hierarchy (editor), 26-2
function key hierarchy (programming), 24-2
function keys, 24-3–24-5
fundamentals, 24-3–24-7
general description of capabilities, 19-1, 19-3–19-4
increasing the size of, 2-13
Layer 1 conditions and actions enabled automatically, 28-1
Mark key, used as program tab, 3-9
overview, 2-12
printing, 14-8
program format, 24-9
program structure, 24-6–24-7
special ASCII characters
backslash (\), 3-5
double quote ("), 3-5
space ( ), 3-5
spreadsheet editor, WRITE command, 13-7
syntax errors, 2-16
unexplained strike-through’s, 2-20
use of cursor keys, 3-8
use of softkeys and the Done key, 3-9
variables shared with Trigger Setup menus, 19-5

Protocol Spreadsheet editor, 26-3–26-10
insert mode, 3-8
Mark key, used as program tab, 3-6
WRITE command, 13-7

Protocol Trace
buffer
logical beginning offset, 61-40
logical end offset, 61-40
monitor position within, 61-38
physical beginning of, 61-38
physical end of, 61-38
size of, 61-38

Protocol trace
See also Trace display
display entering Run mode enabled on Display Setup, 5-17
enabled on Layer Setup screen, 5-16
printing, 14-13

Protocols
compatibility with line setup, 6-4
how to select and load, 6-3
Ps
reset, emulate-mode action, X.25 Layer 3, 34-41
same as last-received Pr, adjunct to send-data-packet action, X.25 Layer 3, 34-35
skip to correct Ps plus one, adjunct to send-data-packet action, X.25 Layer 3, 34-35
trace staggered to indicate two separate numbering sequences, 34-13
X.25 Layer 3, 34-12
value, adjunct to send-data-packet action, X.25 Layer 3, 34-34
Ps error, emulate-mode condition, X.25 Layer 3, 34-27
ps_error, C variable, 72-8

Q
Q, trace column, X.25 Layer 3, 34-13
Q bit
adjunct to monitor/receive-packet condition, 34-18
adjunct to send-packet action, 34-36
position diagrammed, 34-13
Q.931
diagram of message fields, 40-7
message types, adjunct to DTE and DCE receive conditions, 40-9
used with ISDN D channel, 38-3
Quotation mark, entry of inside prompt message, 27-12

R
RAM
data storage, 1-12
RAM-to-disk transfer
bit-oriented data, 11-4
calendar buffer, 11-4
Rack mount assembly, G-1-G-4
RC-8245. See RS-485
Rcv Blk Chk
enabled automatically for BOP, 28-4
field on Line Setup menu, 8-3, 21-5, 28-4, 28-9
field on Line Setup screen, 22-5
must be enabled for BCC conditions to come true, 28-4
subfield on Line Setup menu, 4-9
REJ
monitor/receive condition
LAPD, 39-10
SDLC, 35-12
X.25 Layer 2, 33-12
address needed for Receive REJ, 33-16
X.25 Layer 3, 34-16
send action
LAPD, 39-20
SDLC, 35-23
X.25 Layer 2, 33-23
address required for Send REJ, 33-23
X.25 Layer 3, 34-31
REMOTE RS-232 connector, 1-9
See also Remote port
Read, editor command, 26-6
formatted, 26-6
unformatted, 26-7
Receive
emulate-mode condition
LAPD, 39-15
Layer 1, 28-3
SDLC, 35-16
X.25 Layer 2, 33-15
X.25 Layer 3, 34-24
does not see the data line directly, 34-24
may specify path as added condition, 34-24
via REMOTE RS-232 port, 67-1
Receive path, upward path of IL buffer in emulate mode, 55-3
Receiver, Conditions, Trigger Setup menus, 21-5
Receiver gain, G.703, Interface Control selection, 50-19
Record
layer-independent action, 27-20
start_rccrd_play, C routine, 14-6, 65-3, 69-16
suspend_rccrd_play, C routine, 65-3, 69-17
Record Ch16, G.703 Interface Control screen, 50-5
Record Framing Bits, T1 Interface Control selection, 49-5, 49-26
Record key, 3-11, 11-11
Record Setup
defaults, 11-4
menu selections, 11-2, 11-5-11-10
overview of screen, 2-12
the screen buffer, 11-4
Record Speed
field on Record Setup menu, 11-7
G.703 channel data capture, 50-5
T1 channel data capture, 49-4
Recording data, 11-3-11-11
format of recorded data, 11-3
bit-image data, 11-3
character-oriented, 11-3
manual control of, 3-11
maximum rate, 1-13
medium used, 11-3
record speed
high-speed, 11-9
normal, 11-7

Index to Part I and Part II
screen buffer
  manual control of, 11-11
  trigger control of, 11-10
spreadsheet control of, 27-20
trigger control of, 11-10
with EIA lead transitions, 1-12

Redirect run-mode output
  Line and Record setups override, 14-6
  terminated by recording to disk, 14-6
to disk file instead of printer, 14-6

Relative pathnames, files and directories, 13-6

Relay baton. See Maintain bit

Remote connector, 1-2

Remote LED, 10-4, 48-6, 49-9, 50-11
front panel, 1-5

Remote port
  default configuration, 67-20
transmit and receive data, 67-1
Remote RS-232 connector, Remote port
controlled by C program, 67-1-67-27

Rename, File Maintenance
  menu selection, 13-19
rename, C routine, 65-31

Repair, replacement, return, assistance,
E-1-E-3, H-1-H-4

Replace, editor command, 26-9

Request primitives, 30-5
  versus "indications", 30-10

Resend frame
  effect on Frame Sent condition, 33-18,
  35-18, 39-17
first in window, 33-30, 35-29, 39-27
  action resets resend pointer, 33-30, 35-29,
  39-27
  in relation to window, 33-5, 34-4, 35-5, 39-5
  LAPD action, 39-26
resend_frame, C routine, 77-10
next in window, 33-30, 35-29, 39-27
default resend, 33-30, 35-29, 39-27
SDLC action, 35-27
resend_frame, C routine, 73-10
used with More To Resend and No More To
  Resend conditions, 33-19, 35-19, 39-18
X.25 Layer 2 action, 33-29
  resend_frame, C routine, 71-10

Resend packet
  effect on Packet Sent condition, 34-27
first in window, 34-39
  action resets resend pointer, 34-40
next in window, 34-39
  default resend, 34-39
  programming example, 34-46
  used with More To Resend and No More To
    Resend conditions, 34-28
X.25 Layer 3 action, 34-39
resend_packet, C routine, 72-14

Resend pointer, reset automatically by
  acknowledgement, 33-30, 35-29, 39-27

Resend window, programming example, 34-46

Reset
  G.703 BERT, run-time function key, 9-45
T1 BERT, run-time function key, 9-32

Reset Nr, emulate-mode action
  LAPD, 39-28
  reset_nr, C routine, 77-11
SDLC, 35-30
  reset_nr, C routine, 73-11
X.25 Layer 2, 33-31
  reset_nr, C routine, 71-11

Reset Ns, emulate-mode action
  LAPD, 39-28
  resetNs, C routine, 77-11
SDLC, 35-30
  resetNs, C routine, 73-11
X.25 Layer 2, 33-31
  resetNs, C routine, 71-11

Reset Pr and Ps, emulate-mode action, X.25
  Layer 3, 34-41
resetPrPs, C routine, 72-15

Resolution, display, 1-3

Response addressing
  adjunct to receive condition, X.25 Layer 2,
  33-16
  adjunct to send-frame action, X.25 Layer 2,
  33-24

Response primitives, 30-5

Restart
  G.703 BERT, run-time function key, 9-45
T1 BERT, run-time function key, 9-32

Restart (or start) timeout, layer-independent
  action, 27-11

Restart (or start) timer
  in C, 62-11
  layer-independent action, 27-10

Resync, 9-9
  field on BERT Setup menu, 9-8, 9-21
  in full-duplex BERT, 9-8
    may be inappropriate on noisy circuit, 9-9
    triggered by a fault, 9-21
  not available in half-duplex BERT, 9-9
  outsync mode in BERT, 9-9
Retransmitted I-frames, sample program to enhance all, 33–33
Return, C statement, 58–2
breaks out of while loop, 58–6
Return key, 3–5, 29–1
Rev, subfield on Trigger Setup menu, 22–7
Reverse EBCD
hex-to-display conversion table, D2–3
keyboard-to-reverse-EBCD conversion table, D1–7
RGB monitor, I–5
RGB video connector, 1–9, 16–3
RI, available for triggering, 28–5
Right Arrow key, 26–4
RJ45, ISDN connectors, 48–6
RNR
monitor/receive condition
LAPD, 39–10
SDLC, 35–12
X.25 Layer 2, 33–12
address needed for Receive RNR, 33–16
X.25 Layer 3, 34–16
send action
LAPD, 39–20
SDLC, 35–23
X.25 Layer 2, 33–23
address required for Send RNR, 33–23
X.25 Layer 3, 34–31
ROLL, function key, used to roll through packet-level “causes”, 34–20, 34–36
Robbed bits, T1 Interface Control selection, 49–24
Robbed-bit signaling, T1 transmissions, 49–23
Roll Back key, 3–8, 26–4, 33–5, 34–8, 35–5, 36–4, 40–4, 42–4, 43–4
Roll Fwd key, 3–8, 26–4, 33–5, 34–8, 35–5, 36–4, 40–4, 42–4, 43–4
Root directory, filing system, 13–4
Routines, in C, 58–1–58–6
always followed by parentheses, 58–2
nonzero return makes conditional statement true, 58–6
not usually necessary to declare, 58–1
user-defined, 58–4–58–6
RR
monitor/receive condition
LAPD, 39–10
SDLC, 35–12
X.25 Layer 2, 33–12
address needed for Receive RR, 33–16
X.25 Layer 3, 34–16
send action
LAPD, 39–20
SDLC, 35–23
X.25 Layer 2, 33–23
address required for Send RR, 33–23
X.25 Layer 3, 34–31
RS–170 video connector, 1–10, 16–3
RS–232
connector, REMOTE, 1–9
Test Interface Module, I–7
RS–232/V.24, test connector, 1–10
RS–449
circuits, monitoring by trigger, 45–8
data-plus-leads display, 45–7
lead status, in C, 60–2
Test Interface Module, 45–2, I–15
DIP switches
balanced circuits, 45–5
unbalanced circuits, 45–5
RS–485
data display
A bus as TD data, 47–6
B bus as RD data, 47–6
dual tri-state bus interface, 47–3
enable/disable buses
DTR controls B bus, 47–7
RTS controls A bus, 47–7
via C_ctl_eia routine, 47–8
via EIA spreadsheet (or trigger) action, 47–7
via Interface Control Screen, 47–7
Line Setup configuration, 47–6
minimum length of message, 47–6
only valid emulate mode is EMDTE, 47–3
Test Interface Module, 47–2, I–13
activates drivers to allow transmission, 47–6
connectors, 47–3
too controls output of protocol flags, 47–6
DIP switches
balanced data circuits, 47–4
connector-termination, 47–4
LEDs
A BUS EN, 47–5
B BUS EN, 47–5
EMULATE, 47–5
MESSAGE, 47–6
suppresses non-protocol flags, 47–6
test points, 47–4
transmit message, via SEND spreadsheet action, 47–7

Index to Part I and Part II
RTS
available for triggering, 28–5
enables/disables A bus (RS-485), 47–7
field on Interface Control menu, 10–10,
10–12, 10–14, 10–15
field on RS-232 Interface Control menu, 9–4,
9–6
RTS on/off, Layer 1 Emulate DTE action,
28–10
RTS–off delay, 10–14
RTS–on delay, 10–12
Rub Out key, 3–7, 26–4
Run mode, unit fails to enter, 2–20

S
S, scale, field on Graphical Statistics menu,
18–5, 18–6
SABM
monitor/receive condition, X.25 Layer 2,
33–12
sample program to enhance all occurrences on
trace display, 39–30
send action, X.25 Layer 2, 33–23
SABME, monitor/receive condition, X.25 Layer
2, 33–12
SAPI
adjunct to monitor/receive-frame condition,
LAPD, 39–12
adjunct to send-frame action, LAPD, 39–22
trace column, LAPD, 39–6
Sample action
on counter, 17–6
clears current value, 17–7
translated into C, 62–4
on timer, 17–6
used to compute percentages, 17–11
Sample counter value
in C, 62–4
layer–independent action, 27–10
Sample test, force data–packet transmit, 34–45
Sample timer
in C, 62–12
layer–independent action, 27–11
Save, File Maintenance, menu selection, 13–15
Save key, 3–6, 13–15
Screen buffer, storage capacity, 1–12
Screen display of data
sixteen data lines in center of, 5–5
three divisions of, 5–5
three lines of softkey functions at bottom of,
5–5
two status lines at top of, 5–5
SDLC
diagram of frame fields, 35–8
SDLC Frame Level Setup screen, 35–2
SDU. See Service data unit
SELECT, function key, used to select a rolling
packet–level “cause”, 34–20, 34–36
SETUP, sample program to enhance all
occurrences on trace display, 40–13
Secondary (drop) in SDLC, 35–4
identified in ADDR column of trace display,
35–7
Segment, in 80286 processor, number used to
identify IL buffer, 55–5, 63–5
Selectable, CRC mode, 8–13
versus Bisync mode, 8–13
Selections, column on Layer Setup screen, 6–4
Selectric
default BCC parameters, 8–9
hex–to–display conversion table, D2–3
keyboard–to–Selectric conversion table, D1–8
Self tests, 2–1
See also Field Service
Send frame, Layer 2 action
effect on Frame Sent condition, 33–18,
35–18, 39–17
LAPD, 39–20
send_frame, C routine, 77–12
SDLC, 35–22
send_frame, C routine, 73–12
SNA, 36–3
send_frame, C routine, 74–5
X.25, 33–22
default parameters, 33–22, 33–23
send_frame, C routine, 71–12
Send packet, Layer 3 action
does not send packet directly out on line,
34–24, 34–30
effect on Packet Sent condition, 34–27
X.25, 34–29
send_packet, C routine, 72–16
Send string
Layer 1 action, 28–7, 33–27, 34–38, 35–26,
39–24, 47–7
l1_il_transmit, C routine, 59–12
l1_transmit, C routine, 59–11

Index to Part I and Part II
Service data unit
component in IL buffer structure, 63–3
in transmit routine, 55–4
offset, 55–1
shrinks as IL buffer moves up the layers,
55–1
size in PDU, 63–5
Service Indicators (SIO’s), SS#7 Layer 3, 43–17
Set (and start) timeout, layer–independent
action, 27–11
Set counter value, layer–independent action,
27–9
Set Date, field on Date/Time Setup menu, 15–3
Set flag bits, layer–independent action, 27–14
Set idle character, Layer 1 action, 28–12
Set Time, field on Date/Time Setup menu, 15–3
Setup files, saving and loading, 19–5–19–6
Setup menus
Display Setup screen, overview, 2–11
Line Setup, overview, 2–11
overview, 2–11
See also Separate entry under name of each
menu
Record Setup, 11–5–11–10
overview, 2–12
Severely errored seconds
G.703 BERT statistics, 9–44
T1 BERT statistics, 9–31
Shipping, how to pack, F–1–F–4
Short, C data type, 56–13
short routine returns a short, 58–3
S10
monitor/receive condition, LAPD, 39–10
send action, LAPD, 39–21
S11
monitor/receive condition, LAPD, 39–10
send action, LAPD, 39–21
Sig Channel Polarity, T1 Interface Control
selection, 49–6
Sign extension, occurs during conversion of
signed data types in C, 56–14
Signal
layer–independent action, advantage over flag
or counter, 27–14
layer–independent condition, 27–8
used in layer–to–layer communication, 55–7
Signal Channel Idle Char, T1 Interface Control
selection, 49–6
Signal Channel Number, T1 Interface Control
selection, 49–6
Signal channel idle char, T1 Interface Control
screen, 49–27
Signal channel number, T1 Interface Control
screen, 49–27
Signal channel polarity, T1 Interface Control
screen, 49–27
Signaling bits, T1 Interface Control selection,
49–24
Signalling bits, G.703 transmissions
with CAS signalling with channel 16, 50–32
with CCS/CAS signalling with CRC–4, 50–34
Signalling Channel Control Part (SCCP), SS#7
Layer 3, 43–19
Signalling Link Selection (SLS), SS#7 Layer 3,
43–6, 43–10
Signalling type, G.703, field on Interface
Control menu, 50–23
Signed, C data type, 56–14
Single–channel testing, ISDN, 48–5
Size, trace column
LAPD, 39–8
SDLC, 35–10
SNA–SDLC, 36–6
X.25 Layer 2, 33–10
X.25 Layer 3, 34–14
Sizeof, C operator, 56–20, 71–13, 72–17,
73–13
SNA
fields in protocol trace, 36–7
LU 6.2, 36–3
sample Line Setup, 4–13
SNA/SDLC Frame Level Setup screen, 36–2
SNRM, send action, SDLC, 35–23
Source, field on Line Setup menu, 4–4, 11–4
Speaker, ISDN, ISDN Interface Setup selection,
48–10
Special–receive word, data display, 59–15
Speed
different speeds for TD and RD, 4–11
field on Line Setup menu, 4–10
optimizing high–speed performance, 2–22
selecting monitor and transmit speeds, 4–10
selecting record speed, 11–9
SRC, source, trace column
LAPD, 39–6
Q.931, 40–5
SDLC, 35–7
SS#7 Layer 2, 42–5
X.25 Layer 2, 33–7
X.25 Layer 3, 34–12

SREJ
monitor/receive condition
SDLC, 35–12
X.25 Layer 2, 33–12
address needed for Receive SREJ, 33–16
send action
SDLC, 35–23
X.25 Layer 2, 33–23
address required for Send SREJ, 33–23

SS#7
Layer 1, 41–3–41–6
compression of data, 41–4
Run-time display, 41–3
setup for testing, 41–3
Layer 2, 42–3–42–12
frame structure and values, 42–11
Run-time display, 42–3
setup for testing, 42–3
testing in emulate mode, 42–11
testing in monitor mode, 42–6
Layer 3, 43–3–43–22
ANSI format, 43–4
CCITT format, 43–4, 43–9
Circuit Identifier Code (CIC), 43–6, 43–10
Destination Point Code (DPC), 43–10
Integrated Services Digital Network (ISDN), 43–22
Message Signal Units (MSU’s), 43–6
incomplete, 43–7
structure and values, 43–12–43–22
Network Indicator, 43–9
Network Management (NETM) Headers, 43–18
Originating Point Code (OPC), 43–9
Service Indicators (SIO’s), 43–17
Signalling Channel Control Part (SCCP), 43–19
Signalling Link Selection (SLS), 43–6, 43–10
setting up, 43–3
Telephone User Part (TUP), 43–20–43–21
testing in Monitor mode, 43–7–43–22
US standard format, 43–9
sample Line Setup, 4–13
START/INCL, field on BCC Setup menu, 8–10
START/N/INCL, field on BCC Setup menu, 8–11, 8–14
STX, field on BCC Setup menu, 8–11
Start At Block, subfield on Disk Maintenance menu, 12–8

Start timeout, layer-independent action, 27–11
Start timer, layer-independent action, 27–10
Start up screen, 2–2
Start–stop, data setup, 4–10
Start–stop bit, voltage not affected by inverted
polarity, 4–12
Start/Incl, field on BCC Setup menu, 8–11
Start/N/Incl, field on BCC Setup menu, 8–12
States
in C, 53–2
introduction to concept, 20–3
Protocol Spreadsheet, programming block, 24–8
comments in, 24–13
traced along with layers and tests on Program Trace, 5–19
Static electricity
anti-static packing, J1–3
elimination, J1–3
Static Leads, field on Interface Control menu, 10–17
Statistics
graphics display
accessing via softkey, 18–3
printing, 14–15
identification of counters and timers, 17–4
postponed until after run, 17–6
tabular display
75 values displayed at one time, 17–3
accessing via softkey, 18–3
can scroll through 100 counters, timers, and
accumulators, 17–5
printing, 14–15
tabular menu
cursor movement, 17–4
two cursors, 17–4
Statistics menus, overview, 2–13
Statistics screen
G.703 BERT, 9–43
T1 BERT, 9–30
Statistics Type, field on Display Setup menu, 18–3
Stats, statistical softkey, linked to Statistics Type
field in Display Setup menu, 18–3
Status
four kinds of indicators for leads, 10–8
Status lines
division of Run-mode screen, 5–5
record/playback field, 4–5, 11–9
in BERT, 9–18
Status variables. See Nonevent variables
Stop At, field on Record Setup menu, 11–10
Stop Bits
  field on Line Setup menu, 21–6, 28–4
  subfield on Line Setup menu, 4–10
Stop timeout, layer-independent action, 27–11
Stream, copy of disk file used by disk I/O routines, 65–1
Strike-through’s, Protocol Spreadsheet, 2–20
String
  adjunct to send-frame action
    LAPD, 39–24
    SDLC, 35–26
    X.25 Layer 2, 33–27
  adjunct to send-packet action, X.25 Layer 3, 34–33, 34–38
  relation of string entry in Call Request to DATA field on Packet Level Setup screen, 34–33
  conversion specifier, 61–11
  Layer 1 send action, 33–27, 34–38, 35–26, 39–24
  location of IL buffer, 30–6
  monitor/receive condition
    always in quotation marks on Protocol Spreadsheet, 28–3
    Layer 1, 28–3, 28–4
  monitored or received, size limit, 28–3
  referenced in IL buffer, 30–6
  send action
    always in quotation marks on Protocol Spreadsheet, 28–8
    no practical size limit, 28–7
    valid characters, 28–7
    to be passed down with data primitive, 30–6
    used to initialize an array in C, 56–19, 57–13
    user-defined routine that matches string against line data, 58–6
Strings on Protocol Spreadsheet, 29–1
Strip, field on BCC Setup menu, 8–12
Structure, in C, 57–15
Suppress, field on Display Setup menu, 5–10, 5–13
Suppress not equal, logical equivalent of “display only”, 5–11
Suppress selected trace rows
  LAPD action, 39–29, 39–30
  l2_suppress, C variable, 77–8
  Q.931 action, 40–12, 40–13
  l3_suppress, C variable, 78–4
  SDLC action, 35–30, 35–32
  l2_suppress, C variable, 73–8
  SNA action, 36–3
  l2_suppress, C variable, 74–4
  SS#7 Layer 2 action, 42–10, 42–11
  l2_suppress, C variable, 79–4
  SS#7 Layer 3 action, 43–10, 43–11
  l3_suppress, C variable, 80–7
  X.25 Layer 2 action, 33–32, 33–33
  l2_suppress, C variable, 71–8
  X.25 Layer 3 action, 34–42, 34–43
  l3_suppress, C variable, 72–9
Sync Char
  field on Line Setup menu, 3–6, 9–9
  subfield on Line Setup menu, 4–7
Sync characters
  in half-duplex BERT, 9–7
  must be included in transmit string, 29–3
Sync length, Interface Control screen, 49–25
Sync loss time
  G.703, statistics display, 50–28
  T1 statistics display, 49–30
Sync losses
  G.703, statistics display, 50–28
  T1 statistics display, 49–29
Sync Pattern
  field on BERT Setup menu, 3–6, 9–9
  not applicable in pseudorandom full-duplex test, 9–11
  used for pattern sync in half-duplex pseudorandom test, 9–11
  versus Sync Chars on Line Setup menu, 9–9
  in fox or user-defined test, 9–9
Sync symbol, special symbol on data display, 4–7
Synchronization
  accidental synching, 4–9
  continuous search for sync (autosync), 4–8
  default patterns for standard codes, 4–7
  entering a one-character pattern, 4–7
  in-sync status message in BERT, 9–21
  searched for following Outsync action, 28–11
  when receivers do not search for sync, 4–9
Synchronization point, in half-duplex pseudorandom BERT, 9–11
Syntax errors, Protocol Spreadsheet, 2–16
/sys/fifty_hertz, file name, 1–8
System disk, boot-up, 2–3, 6–3

Index to Part I and Part II
T

T, type, field on Graphical Statistics menu, 18–5
T1, 49–1–49–17
aggregate data capture, 49–5
BERT, 9–23–9–34
automatic error injection rate, 9–29
bit errors, 9–31
block size, 9–29
blocks in error, 9–31
blocks received, 9–31
blocks sent, 9–31
channel mode, 9–27
degraded minutes, 9–31
error-free seconds, 9–31
failed seconds, 9–31
framed mode, 9–28
number of faults, 9–31
run-time function keys, 9–32
Setup screen, 9–27
Statistics screen, 9–30
setting up, 9–24
severely errored seconds, 9–31
test length, 9–29
test seconds, 9–31
unframed mode, 9–28
Bipolar violations, 49–26
bit-robbing
with D4 framing, 49–34
with ESF framing, 49–35
CRC check during sync, ESF framing,
Interface Control selection, 49–25
channel data capture, 49–4
clear-channel signaling
with D4 framing, 49–34
with ESF framing, 49–35
data displays, 49–6
drop-and-insert, 9–26, 49–5
eulation modes, 49–5, 49–7
FAS, 49–35
FEB Setup screen, 9–25
FPS (framing pattern sequence), in ESF transmissions,
field on Interface Control menu, 10–15
field on LAPD Frame Level Setup screen, 39–3
field on X.25 Frame Level Setup screen, 33–3
frame structures, D4 and ESF, 49–32
framing bits
D4, 49–33, 49–34
ESF, 49–35
recording of, 49–26
framing characteristics, 49–4

Interface Control screen, 49–2, 49–19
B8ZS Coding, 49–26
cable length, 49–21
cable type, 49–20
channel number, 49–24
check CRC during sync, 49–25
data path, 49–24
Framing mode, 49–23
Fs Bits, 49–24
frame data link, 49–24
framing (Ft) bits, 49–24
idle select, 49–22
line clock select, 49–22
record framing bits, 49–26
robbed bits, 49–24
sig channel polarity, 49–27
signal channel idle char, 49–27
signal channel number, 49–27
sync length, 49–25
Transmit mode, 49–21
yellow alarm, 49–24, 49–25
in-band signaling
with D4 framing, 49–34
with ESF framing, 49–35
Line Clock selection, 49–12
line conditions, statistics display, 49–31
monitor mode, 49–5
Primary Rate ISDN, 49–6
physical connectors, 49–6
record setup, 11–9
Sync procedure, D4 framing, Interface Control selection, 49–25
setting up menus for testing, 49–18
signaling, robbed-bit (in-band), 49–23
statistics display, 49–27
BPV’s received, 49–29
BPV-free seconds, 49–29
CRC–6 errors, 49–30
carrier losses, 49–31
ESF errors, 49–30
error-free seconds, 49–30
Frames received, 49–29
FT errors, 49–30
FT/FS errors, 49–30
OOF events, 49–30
sync loss time, 49–30
sync losses, 49–29
T1 line conditions, 49–31
test seconds, 49–29
superframing, 49–23
Test Interface Module, 49–2, 49–6, 1–17
signal direction, 49–10
Transmit mode, 9–26
test access points, 49–9
testing and layer protocols, 49–19
testing configurations, 49–11
transmission speeds, 49–3

Index to Part I and Part II
T1 BERT, testing modes, 9–27
T1 expired, emulate-mode condition
LAPD, 39–4, 39–16
X.25 Layer 2, 33–4, 33–18
T1 line conditions, T1 statistics display, 49–31
T1 statistics display, as alternate run–time display, 49–32
T1 timeout conditions under which timer expires, 33–4, 39–4
maximum and minimum values, 33–4, 39–4
T1STATS, T1 BERT, run–time function key, 9–34
T2, field on Interface Control menu, 10–14
T3, field on Interface Control menu, 10–14
T5, field on Interface Control menu, 10–16
T6, field on Interface Control menu, 10–16
Task
C keyword, 56–3
equivalent to spreadsheet Test, 53–1
placed at highest level of source code, 52–16
in linkable-object files, local to the file, 56–11
intercommunication between tasks via signal routine, 55–7
use routine in hook text to export tasks from
LOBJ files, 56–11
TE
ISDN terminal equipment, 48–7
state on ISDN line, 48–11
TEI
adjunct to monitor/receive–frame condition,
LAPD, 39–12
adjunct to send–frame action, LAPD, 39–22
trace column, LAPD, 39–6
Telephone User Part (TUP), SS#7 Layer 3,
43–20–43–21
Temperature, operating, 1–14
Termination, G.703, Interface Control selection,
50–19
Test, field on Display Setup menu, 5–19, 27–19
Test connectors
software control, 1–11
Test Interface Module, back panel,
1–10–1–16
TO DCE, 1–11
TO DTE, 1–11

Test Interface Module
G.703, 50–8
signal direction, 50–10
ISDN, 48–2, 48–6
installation, 1–14, 10–3
LED overlay, 1–5
installation, 1–15, 10–3
LED's, back panel, 1–11
RS–232
AUX outputs, 10–8
breakout panel, 10–5
effect of opened switch on screen and
LED display, 10–5
output jacks, 10–8
test points, 10–8
user–assigned input, 10–7
RS–449, 45–2
AUX outputs, 45–7
output jacks, 45–6
software control, 1–11
T1, 49–6
signal direction, 49–10
test connectors, 1–10–1–16
V.35, 44–2
AUX outputs, 44–7
output jacks, 44–6
X.21, 46–2
output jacks, 46–6
Test Length, field on BERT Setup menu, 9–14,
9–29, 9–42
Test points, on RS–485 TIM, 47–4
Test Seconds, BERT counter, 9–20
Test seconds
G.703, statistics display, 50–27
G.703 BERT statistics, 9–44
T1 BERT statistics, 9–31
T1 statistics display, 49–29
Tests
identified on Program Trace, 5–19
in C, 53–1
Protocol Spreadsheet, programming block,
24–8
comments in, 24–13
simultaneous, program design, 20–3–20–4
TIM
See also Test Interface Module
hardware architecture, J3–14
Tick Rate, field on Front–End Buffer Setup menu,
7–7, 18–7
Tick rate, 7–6
should agree with time “Unit” on Statistics
screen, 7–6

Index to Part I and Part II
Time, trace column
LAPD, 39-8
Q.931, 40-6
SDLC, 35-10
SNA-SDLC, 36-6
SS#7 Layer 2, 42-6
SS#7 Layer 3, 43-6
values may be wall time, ticks, or recorded
ticks, 33-10, 34-14, 35-10, 36-6, 39-8,
40-6, 42-6, 43-6
X.25 Layer 2, 33-10
X.25 Layer 3, 34-14

Time of day, layer-independent condition, 27-6

Time Ticks, field on Front-End Buffer Setup
screen, 7-6, 33-10, 34-14, 39-8, 40-6,
42-6, 43-6

Time ticks
effect on capacity of character buffer, 5-27
enabled/disabled on FEB Setup screen, 7-3
encodable in bit-image or character data, 7-3
gives most accurate timer readings, 27-10
playback, 2-18
of bit-image data, 2-18, 7-6
of character data, 2-17, 7-6
storage of, 1-12
stored in variable called l1_tick_count, 62-8
versus wall-clock timing measurements, 7-3,
7-6, 62-11

Time-of-day clock. See Date/Time Setup

Time/Date Setup, overview, 2-14

Timeout
ccondition, Trigger Setup menus, 21-10
field on Trigger Setup menu, 22-8
layer-independent action, 27-11
layer-independent condition, 27-4
maximum value, 27-11
program to increase maximum value, 27-12
restart (or start), 27-11
shared between spreadsheet and Trigger Setup
menus, 27-11

Timeout expired, SDLC condition, 35-4

Timer
accumulated, 27-15
action
Protocol Spreadsheet, 17-3
Trigger Setup menus, 17-3
identification postponed until after run, 17-6
identified by name on statistics screen, 17-4
layer-independent action, 27-10
printing line of tabular statistics for, 27-16

Timers, no values displayed, 2-21

TO DCE, test connector, 1-11

To Disk Number, subfield on Disk Maintenance
menu, 12-9

Trace
as component of custom protocol analysis,
5-20
as debugging tool, 5-19
compared to prompt, 5-19
layer-independent action, 27-18
layer-independent spreadsheet action, 5-19
versus prompt, 27-18

Trace buffer
correlation with character data, 5-26

Trace display
correlation with character data, 5-26

Trace display
correlation with character data, 5-26

Transitional condition, 27-2, 27-6, 28-1
C translation uses event variable, 54-3

Transitional/status condition, 27-2, 28-1, 28-5
C translation uses event or status variable,
54-3

Transmit
csample transmit program
BOP echo, 55-10
sync or async echo, 55-9
via REMOTE RS-232 port, 67-1

Transmit complete, Layer 1 condition, 28-6

Transmit mode
cG.703, Interface Control selection, 50-20
T1, Interface Control selection, 49-21

Transmit string
ccomplete version entered only at Layer 1,
29-3
does not appear on display, 2-21

Transmit tag, in header of IL buffer, 55-7

Trigger, condition-action grouping on Protocol
Spreadsheet, 27-1

Trigger conditions, EIA, fails to come true,
2-20

Trigger freeze. See Capture data to screen
(on/off)

Trigger Setup, variables shared with Protocol
Spreadsheet, 19-5

Trigger Setup menus, 21-3
Actions, 22-3-22-12
basic description of capabilities, 19-2
Conditions
Buffer Full, 21-11
combined with other Conditions, 21-4
Counter, 21-11
combined with other Conditions, 21-4
combining static and instantaneous, 21-4
EIA, 21-9
combined with other Conditions, 21-4
Flags, 21-10
combined with other Conditions, 21-4
Keyboard, 21-12
Receiver, 21-5
Timeout, 21-10
Xmit Complete, 21-10
menu selections
(Actions), 2
(Conditions), 2
overview, 2-12
transmit string, does not appear on screen, 2-21
Trigger Summary screen, 23-3
Triggers
active, 21-4
color of color display, 16-5—16-6
in C, 53-8
Trouble-shooting
data plus leads, failure of leads to transition, 2-21
data—plus—leads display, failure of leads to transition, 2-21
layers, passing data between, 2-21
overheating, 1-9, 2-22
Program key, unit unexpectedly enters Run mode, 2-20
Protocol Spreadsheet, unexplained
strike-through's, 2-20
Run mode, unit fails to enter, 2-20
timers, no values displayed, 2-21
transmit string, does not appear on screen, 2-21
trigger conditions, EIA, fails to come true, 2-20
Twisted pair, patch cords, 44-5, 45-5, 46-5
Two's complement, 56-14
TYPE, trace column
LAPD, 39-8
SDLC, 35-7
X.25 Layer 2, 33-7
X.25 Layer 3, 34-12
Type
field on BCC Setup menu, 8-11
field on Disk Maintenance menu, 11-4
field on Display Setup menu, 5-7
primitives, 30-4
subfield on Disk Maintenance menu, 12-8
Type conversion, automatic in some circumstances in C, 56-14
trig_flag, name of flag mask on Trigger Setup menus, 21-11, 22-6, 27-7
trig_timeout_1, name of timeout on Trigger Setup menus, 22-9
trig_timeout_2, name of timeout on Trigger Setup menus, 22-9
U
%u, C conversion specifier, converts char to short, 56-14
U, unit, field on Graphical Statistics menu, 18-7
U/A
LED, 1-11
RS-232 input jack, 10-7
RS-449 input jacks, 45-5
A and B, 45-5
A used for unbalanced patching, 45-5
V.35 input jacks, 44-5
A and B, 44-5
A used for unbalanced patching, 44-6
X.21 input jacks, 46-5
A and B, 46-5
monitored for on/off status, 32-7
UA, send action
SDLC, 35-23
X.25 Layer 2, 33-23
Undelete, editor command, 26-9
Unframed mode
G.703 BERT, 9-40
T1 BERT, 9-28
Unit, column on Tabular Statistics screen, 17-6
Unit of time
selection for printout of timer line, 27-17
selection on a statistics screen, 7-7
Unknown frame, receive condition
LAPD, 39-16
SDLC, 35-16
X.25 Layer 2, 33-17
Unknown packet, receive condition, X.25 Layer 3, 34-26
Unresolved reference, error message, 2-20
Unsigned, C data type, 56-14
Up Arrow key, 26-4
User disk, personality packages reside on, 6-3
User trace
#pragma tracebuf, 5–24, 61–26
buffer may be scrolled through in Freeze mode, 61–23
buffer size may be increased, 5–24, 61–26
C routines, 61–27
display mode, 5–24–5–26
messages written only via C routines, 61–23
newline nonliteral (\n) provides leading blank line, 61–27
seven buffers containing 4096 characters each, 61–23
seven of eight trace buffers are user–trace buffers, 61–23
structures, declared in trace_buf.h #include file, 61–23
trace_buf, C structure, 61–23
trace_buffer_header, C structure, 61–23
User-assigned BERT pattern, 9–7
/usr directory, filing system, 13–6
/usr/default
affect on Start Up screen, 2–2
boot-up menu configuration, 2–5
default program, 2–5
/usr/user_intrf
affect on Start Up screen, 2–2
creating a user interface, 2–4
may be given display line on stat screen(s), 27–15
Utilities menus, overview, 2–14
See also Separate entry under name of each menu

V
V, value, field on Graphical Statistics menu, 18–6
V.35
circuits, monitoring by trigger, 44–7
Test Interface Module, 44–2, 1–9
DIP switches
balanced circuits, 44–5
unbalanced circuits, 44–5
test connector, 1–10
Video connectors
CRT/RGB, 1–9
RS–170 composite video, 1–10
View, File Maintenance, menu selection, 13–18
Void, C data type, return statement invalid with this type, 58–3
Voltage selection, back panel, 1–7

Voltagens
RS–232
detected by receivers, 10–9
generated by drivers, 10–9
generated by special output jacks, 10–9
indicated by UA–input LEDs, 10–7
RS–449
generated by special output jacks, 45–6
indicated by UA–input LEDs, 45–5
V.35
generated by special output jacks, 44–6
indicated by UA–input LEDs, 46–6
X.21
generated by special output jacks, 46–6
indicated by UA–input LEDs, 46–5

W
Wait for End Of Frame
condition dependent on Rcv Blk Chk, ON, 8–5
subfield on Trigger Setup menu, 8–5
Wait for End of Frame, subfield on Trigger Setup menu, 21–9
Wait for EOF (end of frame)
adjunct to String or One–of condition, Layer 1, 28–5
Layer 1 condition, 28–3
placed inside of state loop, 52–16
Wall clock
accurate to one millisecond, 7–6
drives the timers displayed on the stats results screen, 62–11
enabled when time ticks are disabled, 7–6
timings available via the
get_wall_time_286_ticks routine, 62–11
Warranties, E–3
WECO 310, T1 connectors, 49–7, 49–8
While, C statement, nonzero expressions always true inside of while statement, 58–2
Winchester hard disk
installing new system software, 2–7
storage capacity, 1–12
Window
cleared by Reset Ns action, 33–31, 35–30, 39–28

Index to Part I and Part II
empty, emulate-mode condition, 34-28, 35-18, 39-17
LAPD, translated into C, 77-8
SDLC, translated into C, 73-8
X.25 Layer 2, 33-18
translated into C, 71-8
X.25 Layer 3, translated into C, 72-13
full
effect on Send action, 33-23, 34-31, 35-22, 39-20
emulate-mode condition, 34-28, 35-18, 39-17
LAPD, translated into C, 77-8
SDLC, translated into C, 73-8
X.25 Layer 2, 33-18
translated into C, 71-8
X.25 Layer 3, translated into C, 72-12
not empty, emulate-mode condition, 34-28, 35-18, 39-17
LAPD, translated into C, 77-8
SDLC, translated into C, 73-8
X.25 Layer 2, 33-18
translated into C, 71-8
X.25 Layer 3, translated into C, 72-13
not full, emulate-mode condition, 34-28, 35-18, 39-17
LAPD, translated into C, 77-8
SDLC, translated into C, 73-8
X.25 Layer 2, 33-18
translated into C, 71-8
X.25 Layer 3, translated into C, 72-12
Window Size
field on LAPD Frame Level Setup screen, 39-3
field on SDLC Frame Level Setup screen, 35-3
field on SNA/SDLC Frame Level Setup screen, 36-3
field on X.25 Frame Level Setup screen, 33-3
field on X.25 Packet Level Setup screen, 34-3
Window size, 33-5, 34-4, 35-5, 39-5
Write, editor command, 13-7, 26-6
formatted, 26-6
unformatted, 26-7
Write Enable, File Maintenance, menu selection, 13-19
Write Protect, File Maintenance, menu selection, 13-20
Write protection, microfloppies, 1-5
while, C statement, 58-6

X
%\x, C conversion specifier, converts char to short, 56-14
X.200, CCITT recommendation, 20-5
X.21
call-setup phase, 32-4
clamping/unclamping data leads, 32-9
set_tcr_b, C routine, 70-8
invoking, 32-10
enter_call_phase, C routine, 70-9
plus, bell and sync idle, 32-9
x21_idle_action, C routine, 70-6
selectable as initial phase, 46-9
send action, 32-8
code and format, 32-8
x21_transmit_call, C routine, 70-7
data-plus-leads display, 46-6
data-transfer phase, 32-5
default initial phase, 46-9
invoking, 32-11
enter_data_phase, C routine, 70-9
selectable as initial phase, 46-9
send action, 32-7
Interface Control Menu screen, 46-8
Layer 1 package, 32-3
leads
controlling C and I, 32-10
monitoring C and I for true or valid status, 32-6
monitoring T and R for valid status, 32-5
sending from Layer 2, 32-5, 32-11
Test Interface Module, 46-2, 1-11
DIP switches, 46-5
X.21 bis, lead conversions, 32-4
X.25
diagram of frame fields, 33-8
diagram of packet fields, 34-10
sample Line Setup, 4-13
user program to convert protocol headers to hexadecimal, 33-35
user program to force data packets containing fox messages out onto the line from Layer 3, 34-45
user program to make Layer 2 "automatic" for higher layer, 33-36
X.25 Frame Level Setup screen, 33-2
X.25 Packet Level Setup screen, 34-2, 34-32
Xmit Complete
condition, Trigger Setup menus, 21-10
fevar_xmit_cmplt, C event, 59-5
Layer 1 condition, 28-6
Xmit Delay, field on Interface Control menu, 10-12, 10-16

Index to Part I and Part II
Xmit distant MF alarm, G.703, Interface Control selection, 50–22

Xmit Idle Char
  field on Line Setup menu, 3–6
  subfield on Line Setup menu, 4–9

Xmit remote alarm, G.703, field on Interface Control menu, 50–22

Xmit signalling all 1's, G.703, Interface Control selection, 50–22

XS–3
  default BCC parameters, 8–9
  default sync pattern, 4–7

hex–to–display conversion table, D2–3
keyboard–to–XS–3 conversion table, D1–5
SY characters inappropriate for, 4–7

Y
Yellow alarm, T1 transmissions, 49–24

Z
Zero, transmitting steady zero, set_tcr_b, C routine, 59–14

Index to Part I and Part II
Index B
C Structures, Variables, and Routines

A

Aux port I/O
  events, aux_change, 68-4
  routines
    set_aux_ctl_leads, 68-6
    set_aux_direction, 68-5
    set_aux_reg, 68-10
    write_aux, 68-7
variables
  curr_aux_value, 68-4
  prev_aux_value, 68-4

add_array_to_buff, data-display routine, defined, 59-18
add_event_to_buff, data-display routine, defined, 59-17
append_ill_buff_list_cnt, OSI layer-independent routine, 55-7
defined, 63-43
aux_change, aux port I/O event, 68-3, 68-10
defined, 68-4

counter_name_change, counter event, 54-2, 54-3, 62-1
defined, 62-3
counter_struct, counter structure, 57-15, 62-1, 62-6, 62-16
defined, 62-2
crnt_date_of_day, real-time clock variable, defined, 69-4
crnt_display_screen, status variable, 61-1, 66-1, 69-18
defined, 61-2
crnt_time_of_day, real-time clock variable, 52-1, 54-1, 57-3
defined, 69-4
crnt_tm, real-time clock structure, defined, 69-2
ctl_capture_rd, data-display routine, 57-12, 59-9
defined, 59-9
ctl_capture_td, data-display routine, 57-12
defined, 59-8
ctl_eia
  EIA routine, 60-4
defined, 60-3
  RS-485 application, 47-8
  X.21 routine, 70-5
defined, 70-4
cntl_enhance_rd, data-display routine, defined, 59-8
cntl_enhance_td, data-display routine, defined, 59-7
curr_aux_value, aux port I/O variable, 68-8, 68-10
defined, 68-4
current_col, Display Window variable, 61-4
defined, 61-5
current_eia_leads
  EIA variable, 57-8, 60-1, 60-2, 60-3
  defined, 60-2
  X.21 variable, 70-2
defined, 70-3
current_line, Display Window variable, 61-4
defined, 61-5

curr_aux_value, aux port I/O variable, 68-8, 68-10
defined, 68-4
current_col, Display Window variable, 61-4
defined, 61-5
current_eia_leads
  EIA variable, 57-8, 60-1, 60-2, 60-3
  defined, 60-2
  X.21 variable, 70-2
defined, 70-3
current_line, Display Window variable, 61-4
defined, 61-5

Index to C Structures, Variables, and Routines
Data display routines
   add_array_to_buff, 59-18
   add_event_to_buff, 59-17
   ctl_capture_rd, 59-9
   ctl_capture_td, 59-8
   ctl_enhance_rd, 59-8
   ctl_enhance_td, 59-7
variables
   rd_modifier, 59-5
   td_modifier, 59-4

DDCMP events
   fevar_bd_bcc_rd, 75-2
   fevar_bd_bcc_td, 75-2
   fevar_bd_bcc2_rd, 75-2
   fevar_bd_bcc2_td, 75-2
   fevar_gd_bcc_rd, 75-2
   fevar_gd_bcc_td, 75-2
   fevar_gd_bcc2_rd, 75-2
   fevar_gd_bcc2_td, 75-2

Disk I/O routines
   clearerr, 65-10
   fclose, 65-5
   feof, 65-7
   ferror, 65-8
   fflush, 65-6
   fgetc, 65-20
   fgets, 65-18
   fopen, 65-4
   fprintf, 65-28
   fputc, 65-26
   fputs, 65-25
   fread, 65-16
   fseek, 65-11
   fwrite, 65-23
   _get_file_type, 65-36
   lock, 65-14
   mkdir, 65-33
   remove, 65-32
   rename, 65-31
   rewind, 65-13
   _set_file_type, 65-34
   ungetc, 65-21
   unlock, 65-16

Display Window routines
   display_prompt, 61-21
   displayc, 61-9
   displayf, 61-9
   displays, 61-20
   pos_cursor, 61-21
   restore_cursor, 61-22
   sprintf, 61-13
structures, display_window_index_buffer, 61-8
variables
   current_col, 61-5
   current_line, 61-5
   display_window_buffer, 61-7
   window_color, 61-5
   window_modifier, 61-7
d_dce_frame, ISDN event, defined, 76-2
d_dte_frame, ISDN event, defined, 76-2
d_rcv_frame, ISDN event, defined, 76-2
dce_abort
   LAPD event, defined, 77-3
   SDLC event, defined, 73-3
   SS#7 Layer 2 event, defined, 79-2
   X.25 Layer 2 event, defined, 71-3
dce_bad_bcc
   LAPD event, defined, 77-3
   SDLC event, defined, 73-3
   SS#7 Layer 2 event, defined, 79-2
   X.25 Layer 2 event, defined, 71-3
dce_flags, SS#7 Layer 1 variable, defined, 79-5
dce_frame, 54-2
   LAPD event, defined, 77-3
   SDLC event, defined, 73-3
   SS#7 Layer 2 event, defined, 79-2
   X.25 Layer 2 event, defined, 71-3
dce_frames_suppressed, SS#7 Layer 1 variable, defined, 79-5
dce_good_bcc
   LAPD event, defined, 77-3
   SDLC event, defined, 73-3
   SS#7 Layer 2 event, defined, 79-2
   X.25 Layer 2 event, defined, 71-3
dce_packet
   Q.931 event, defined, 78-2
   SS#7 Layer 3 event, defined, 80-2
   X.25 Layer 3 event, defined, 72-3
display_binary, user-defined routine, 58-5
display_prompt, Display Window routine, 67-16
   defined, 61-21
display_screen_changed, status event, 61-1,
   66-1
   defined, 61-2
display_window_buffer, Display Window
   variable, 61-4, 61-35, 61-36
   defined, 61-7

Index to C Structures, Variables, and Routines
display_window_index_buffer, Display Window structure, 61-7, 61-35
defined, 61-8

displayc, Display Window routine, 61-1
defined, 61-9

defined, 61-9

displays, Display Window routine, 56-19, 65-12, 65-24
defined, 61-20

dte_abort
LAPD event, defined, 77-3
SDLC event, defined, 73-3
SS#7 Layer 2 event, defined, 79-2
X.25 Layer 2 event, defined, 71-3

dte_bad_bcc
LAPD event, defined, 77-3
SDLC event, defined, 73-3
SS#7 Layer 2 event, defined, 79-2
X.25 Layer 2 event, defined, 71-3

dte_flags, SS#7 Layer 1 variable, defined, 79-5

dte_frame
LAPD event, defined, 77-3
SDLC event, defined, 73-3
SS#7 Layer 2 event, defined, 79-2
X.25 Layer 2 event, defined, 71-3

dte_frames_suppressed, SS#7 Layer 1 variable,
defined, 79-5

dte_good_bcc
LAPD event, defined, 77-3
SDLC event, defined, 73-3
SS#7 Layer 2 event, defined, 79-2
X.25 Layer 2 event, defined, 71-3

dte_packet, 54-2
Q.931 event, defined, 78-2
SS#7 Layer 3 event, 80-1
defined, 80-2
X.25 Layer 3 event, defined, 72-3
__dup_il_buff_list, OSI layer-independent
routine, 63-43, 63-44
defined, 63-34
__dup_il_buff_list_start, OSI layer-independent
routine, 63-39, 63-42, 63-43
defined, 63-33

E

EIA
events, fevar_eia_changed, 60-2
routines, ctrl_eia, 60-3
RS-485 application, 47-8
variables
current_eia_leads, 60-2
previous_eia_leads, 60-2
enter_call_phase, X.21 routine, 70-9
defined, 70-9
ter_data_phase, X.21 routine, 70-10
defined, 70-9

F

Flag
events, flag_name_change, 69-4
structures, flag_struct, 69-2
defined, 65-5
feof, disk I/O routine, 65-4, 65-8
defined, 65-7
ferror, disk I/O routine, 65-4, 65-9
defined, 65-8
fevar_abort_rd, line event, 59-6
defined, 59-3
fevar_abort_td, line event, 59-6
defined, 59-3
fevar_bd_bcc
DDCMP event, defined, 75-2
line event, 59-2, 59-6
defined, 59-3
fevar_bd_bcc2
DDCMP event, defined, 75-2
line event, 59-2, 59-6
defined, 59-3
fevar_eia_changed
EIA event, 53-15, 54-2, 60-1, 60-3
defined, 60-2
X.21 event, 70-2
defined, 70-3
fevar_frm_error_rd, line event, defined, 59-3
fevar_frm_error_td, line event, defined, 59-3
fevar_gd_bcc
DDCMP event, defined, 75-2
line event, 54-2, 59-6
defined, 59-3

Index to C Structures, Variables, and Routines
fevar_gd_bcc_td
DDCMP event, defined, 75-2
line event, defined, 59-3
fevar_gd_bcc2_rd, DDCMP event, 59-6
defined, 75-2
fevar_gd_bcc2_td, DDCMP event, defined, 75-2
fevar_parity_rd, line event, defined, 59-3
fevar_parity_td, line event, defined, 59-3
fevar_rcv_buffer_full, line event, 59-2
defined, 59-3
fevar_rcvd_char_rd, line event, 55-1, 59-5
defined, 59-3
fevar_rcvd_char_td, line event, 54-2, 55-1, 59-5
defined, 59-3
fevar_time_of_day, real-time clock event, 52-1, 54-1, 54-2, 57-1, 57-3
defined, 69-4
fevar_xmit_cmplt, transmit event, 59-5
defined, 59-4
fflush, disk I/O routine, 65-1, 65-2, 65-3, 65-7
defined, 65-6
fgetc, disk I/O routine, 65-1, 65-2
defined, 65-20
getd, disk I/O routine, 65-1, 65-2, 65-19
defined, 65-18
flag_name_change, flag event, 54-2, 54-3
defined, 69-4
flag_struct, flag structure, defined, 69-2
defined, 65-4
fprintf, disk I/O routine, 65-1, 65-2, 65-31
defined, 65-28
fputc, disk I/O routine, 65-2, 65-11, 65-27
defined, 65-26
fgets, disk I/O routine, 65-2
defined, 65-25
frame_sent
LAPD event, defined, 77-4
SDLC event, defined, 73-4
X.25 Layer 2 event, defined, 71-4
fread, disk I/O routine, 65-1, 65-2, 65-9, 65-17, 65-36, 67-16
defined, 65-16
_free_il_msg_buff, OSI layer-independentroutine, 55-5, 63-38, 63-39
fseek, disk I/O routine, 65-2, 65-4, 65-12, 65-14
defined, 65-11
defined, 65-23
defined, 63-37

G
get_68k_phys_addr, stats-display routine, 58-1, 62-5, 62-7, 62-16
defined, 62-14
_get_file_type, disk I/O routine, 65-37
defined, 65-36
defined, 63-32
_get_il_msg_buff, OSI layer-independent
routine, 55-5, 55-8, 59-13, 63-36, 63-37, 63-41, 63-45, 71-13, 72-17, 73-13, 76-5, 77-13
defined, 63-31
get_wall_time_286_ticks, timer routine, 62-11, 62-12, 62-17
defined, 62-17
get_wall_time_ticks, timer routine, 62-17
defined, 62-16

I
Interrupt
events, signal_name, 69-4
routines, signal, 69-15
ISDN
events
d_dce_frame, 76-2
d_dte_frame, 76-2
d_rcv_frame, 76-2
routines
send_d_frame, 76-3
send_d_frame_il, 76-4
set_isdn_speaker_chan, 76-5
structures, xmit_list, 76-1
idle_action, transmit routine, defined, 59-13
il_buffer, OSI structure, 55-7, 57-16, 63-6, 63-7
defined, 63-9
il_list_header, OSI structure, 63-4
defined, 63-10

Index to C Structures, Variables, and Routines
il_list_node, OSI structure, 63-4  
defined, 63-11
index, string routine, 56-22, 69-11  
defined, 69-11
insert_il_buff_list_cnt  
OSI layer-independent routine, 55-5, 55-7,  
55-8, 59-13, 63-36, 63-39, 63-41,  
63-42, 63-46, 71-13, 72-17, 73-13,  
76-5, 77-13  
defined, 63-39
invalid_frame  
LAPD event, defined, 77-3  
SDLC event, defined, 73-3  
X.25 Layer 2 event, defined, 71-3
invalid_packet, X.25 Layer 3 event, defined,  
72-3  
OSI routine, 63-12

K

Keyboard  
events  
keyboard_new_any_key, 69-5  
keyboard_new_key, 69-4  
routines, send_key, 69-17  
structures, keyboard, 69-2  
variable, keyboard_any_key, 69-5
keyboard, keyboard structure, defined, 69-2
keyboard_any_key, keyboard variable, 53-9,  
53-13, 59-14, 69-18  
defined, 69-5
keyboard_new_any_key, keyboard event, 53-9,  
53-13, 53-14, 53-15, 57-2  
defined, 69-5
keyboard_new_key, keyboard event, 53-14,  
53-15, 54-2, 69-2  
defined, 69-4

dte_good_bcc, 77-3  
frame_sent, 77-3  
invalid_frame, 77-4  
l2_T1, 77-3  
nr_error, 77-3  
ns_error, 77-3  
rsvd_frame, 77-3
routines  
l2_give_data, 77-9  
resend_frame, 77-10  
reset_nr, 77-11  
reset_ns, 77-12
send_frame, 77-12
structures, send_frame_structure, 77-2
variables  
l2_current_window_edge, 77-5  
l2_enhance, 77-6  
l2_lower_window_edge, 77-5  
l2_resend_edge, 77-5  
l2_suppress, 77-6
l2_upper_window_edge, 77-5  
m_frame_addr_cr, 77-4  
m_frame_addr_sapi, 77-4  
m_frame_addr_tei, 77-4  
m_frame_bcc_type, 77-4  
m_frame_ctrl_byte_1, 77-4  
m_frame_nr, 77-4  
m_frame_ns, 77-4
m_frame_pfc, 77-4  
m_frame_type, 77-4
rcvd_frame_addr_cr, 77-4  
rsvd_frame_addr_sapi, 77-4  
rsvd_frame_addr_tei, 77-4  
rsvd_frame_bcc_type, 77-5  
rsvd_frame_bcc_seg, 77-5
rsvd_frame_ctrl_byte_1, 77-5  
rsvd_frame_nr, 77-5  
rsvd_frame_ns, 77-5  
rsvd_frame_pfc, 77-5  
rsvd_frame_sdu_offset, 77-5  
rsvd_frame_sdu_size, 77-5  
rsvd_frame_type, 77-4

L

LAPD  
events  
bcc_error, 77-3  
dce_abort, 77-3  
dce_bad_bcc, 77-3  
dce_frame, 77-3  
dce_good_bcc, 77-3  
dte_abort, 77-3  
dte_bad_bcc, 77-3  
dte_frame, 77-3  
fevar_abort_rd, 59-3  
fevar_abort_td, 59-3  
fevar_bd_bcc_rd, 59-3  
fevar_bd_bcc_td, 59-3  
fevar_frm_error_rd, 59-3  
fevar_frm_error_td, 59-3  
fevar_gd_bcc_rd, 59-3  
fevar_gd_bcc_td, 59-3  
fevar_parity_rd, 59-3  
fevar_parity_td, 59-3  
fevar_rcv_buffers_full, 59-3

Index to C Structures, Variables, and Routines
Index to C Structures, Variables, and Routines
SS#7 Layer 3 variable, defined, 80-6
X.25 Layer 3 variable, 72-9, 80-7
defined, 72-7

13_tick_count, OSI Layer 3 variable, defined, 63-19

13_trbuf, trace buffer structure, defined, 61-25

13_window_empty, X.25 Layer 3 routine, defined, 72-13

13_window_full, X.25 Layer 3 routine, 58-2
defined, 72-12

13pp_trbuff, Protocol Trace variable, defined, 61-39

13pp_trbuff_ctl, Protocol Trace structure, defined, 61-42

13pp_trbuff_end, Protocol Trace variable, defined, 61-39

14_tick_count, OSI Layer 4 variable, defined, 63-22

14_trbuf, trace buffer structure, defined, 61-25

15_tick_count, OSI Layer 5 variable, defined, 63-25

15_trbuf, trace buffer structure, defined, 61-26

16_tick_count, OSI Layer 6 variable, defined, 63-28

16_trbuf, trace buffer structure, defined, 61-26

17_tick_count, OSI Layer 7 variable, defined, 63-30

17_trbuf, trace buffer structure, defined, 61-26

lo_dl_il_buff, OSI Layer 3 variable, 55-3,
63-50
defined, 63-18

lo_dl_pdu_seg, OSI Layer 3 variable, defined, 63-17

lo_dl_prmtv, OSI Layer 3 event, 55-3
defined, 63-17

lo_dl_prmtv_code, OSI Layer 3 variable, 63-47
defined, 63-17

lo_dl_prmtv_path, OSI Layer 3 variable, 63-47,
63-51
defined, 63-18

lo_dl_sdu, OSI Layer 3 variable, 55-3, 63-51
defined, 63-18

lo_n_il_buff, OSI Layer 4 variable, 63-54,
63-56
defined, 63-21

lo_n_pdu_seg, OSI Layer 4 variable, defined, 63-20

lo_n_prmtv, OSI Layer 4 event, defined, 63-20

lo_n_prmtv_code, OSI Layer 4 variable, 63-51
defined, 63-20

lo_n_prmtv_path, OSI Layer 4 variable, 63-55
defined, 63-21

lo_n_sdu, OSI Layer 4 variable, 63-55
defined, 63-21

lo_p_il_buff, OSI Layer 7 variable, defined, 63-30

lo_p_pdu_seg, OSI Layer 7 variable, defined, 63-29

lo_p_prmtv, OSI Layer 7 event, defined, 63-29

lo_p_prmtv_code, OSI Layer 7 variable, 63-63
defined, 63-29

lo_p_prmtv_path, OSI Layer 7 variable, defined, 63-29

lo_p_sdu, OSI Layer 7 variable, defined, 63-30

lo_ph_il_buff, OSI Layer 2 variable, 55-3,
63-46
defined, 63-15

lo_ph_pdu_seg, OSI Layer 2 variable, defined, 63-14

lo_ph_prmtv, OSI Layer 2 event, 55-3
defined, 63-14

lo_ph_prmtv_code, OSI Layer 2 variable, 63-45
defined, 63-14

lo_ph_prmtv_path, OSI Layer 2 variable,
defined, 63-15

lo_ph_sdu, OSI Layer 2 variable, 55-3, 63-46,
63-47
defined, 63-15

lo_s_il_buff, OSI Layer 6 variable, 63-62
defined, 63-27

lo_s_pdu_seg, OSI Layer 6 variable, defined, 63-26

lo_s_prmtv, OSI Layer 6 event, defined, 63-26

lo_s_prmtv_code, OSI Layer 6 variable, defined,
63-26

lo_s_prmtv_path, OSI Layer 6 variable, 63-63
defined, 63-27

lo_s_sdu, OSI Layer 6 variable, 63-63
defined, 63-27

lo_t_il_buff, OSI Layer 5 variable, 63-58
defined, 63-24
Index to C Structures, Variables, and Routines
m_lo_n prmtv_path, OSI Layer 4 variable, defined, 63-21
m_lo_n_sdu_offset, OSI Layer 4 variable, 63-21
m_lo_n_sdu_size, OSI Layer 4 variable, 63-56 defined, 63-21
m_lo_p il buff, OSI Layer 7 variable, defined, 63-30
m_lo_p pdu_seg, OSI Layer 7 variable, defined, 63-29
m_lo_p prmtv, OSI Layer 7 event, defined, 63-29
m_lo_p prmtv_code, OSI Layer 7 variable, 63-64 defined, 63-29
m_lo_p prmtv_path, OSI Layer 7 variable, defined, 63-30
m_lo_p sdu_offset, OSI Layer 7 variable, defined, 63-30
m_lo_p sdu_size, OSI Layer 7 variable, defined, 63-30
m_lo_ph il buff
OSI Layer 2 variable, 55-1, 55-2, 55-3, 63-48 defined, 63-15
OSI layer 2 variable, 57-10, 57-12, 57-17
m_lo_ph pdu_seg, OSI Layer 2 variable, defined, 63-14
m_lo_ph prmtv
OSI Layer 2 event, 55-1, 55-3 defined, 63-15
OSI layer 2 event, 57-10 signalled by DDCMP package, 37-2
m_lo_ph prmtv_code, OSI Layer 2 variable, defined, 63-14
m_lo_ph prmtv_path, OSI Layer 2 variable, defined, 63-14
m_lo_ph_sdu_offset, OSI Layer 2 variable, 55-1, 63-48 defined, 63-15
m_lo_ph_sdu_size, OSI Layer 2 variable, 55-3, 63-48 defined, 63-15
m_lo_s il buff, OSI Layer 6 variable, 63-64 defined, 63-27
m_lo_s pdu_seg, OSI Layer 6 variable, defined, 63-26
m_lo_s prmtv, OSI Layer 6 event, defined, 63-26
m_lo_s prmtv_code, OSI Layer 6 variable, 63-60 defined, 63-26
m_lo_s prmtv_path, OSI Layer 6 variable, defined, 63-27
m_lo_s sdu_offset, OSI Layer 6 variable, 63-64 defined, 63-27
m_lo_s_sdu_size, OSI Layer 6 variable, 63-64 defined, 63-27
m_lo_t il buff, OSI Layer 5 variable, 63-60 defined, 63-24
m_lo_t pdu_seg, OSI Layer 5 variable, defined, 63-23
m_lo_t prmtv, OSI Layer 5 event, defined, 63-23
m_lo_t prmtv_code, OSI Layer 5 variable, 63-56 defined, 63-23
m_lo_t prmtv_path, OSI Layer 5 variable, defined, 63-24
m_lo_t sdu_offset, OSI Layer 5 variable, 63-60 defined, 63-24
m_lo_t_sdu_size, OSI Layer 5 variable, 63-60 defined, 63-24
m_message_type, Q.931 variable, defined, 78-3
m_message_type_defined, Q.931 variable, defined, 78-2
m_packet_bcc_type, Q.931 variable, defined, 78-2
m_packet_buff_seg, X.25 Layer 3 variable, 72-8 defined, 72-5
m_packet_cause, X.25 Layer 3 variable, defined, 72-4
m_packet_d, X.25 Layer 3 variable, defined, 72-3
m_packet_daf, SNA variable, defined, 74-2
m_packet_def, SNA variable, defined, 74-2
m_packet_diag_code, X.25 Layer 3 variable, defined, 72-4
m_packet_dsa f, SNA variable, defined, 74-2
m_packet_fi, SNA variable, defined, 74-3
m_packet_fid_type, SNA variable, 74-1 defined, 74-2

Index to C Structures, Variables, and Routines
Index to C Structures, Variables, and Routines

m_packet_info_length, X.25 Layer 3 variable, defined, 72-5
m_packet_info_offset, X.25 Layer 3 variable, defined, 72-5
m_packet_info_ptr
X.25 Layer 2 variable, 58-6
X.25 Layer 3 variable, 58-1, 61-29, 61-30, 72-8
defined, 72-6
m_packet_info_seg, X.25 Layer 3 variable, defined, 72-5
m_packet_lcn, X.25 Layer 3 variable, 61-29, 61-30
defined, 72-3
m_packet_lcngrp, X.25 Layer 3 variable, defined, 72-3
m_packet_length
SNA variable, defined, 74-2
X.25 Layer 3 variable, 72-8
defined, 72-5
m_packet_lsid, SNA variable, defined, 74-2
m_packet_m, X.25 Layer 3 variable, defined, 72-3
m_packet_oaf, SNA variable, defined, 74-2
m_packet_oeof, SNA variable, defined, 74-2
m_packet_osaf, SNA variable, defined, 74-2
m_packet_pr, X.25 Layer 3 variable, defined, 72-3
m_packet_ps, X.25 Layer 3 variable, defined, 72-3
m_packet_ptr, X.25 Layer 3 variable, 57-14 defined, 72-6
m_packet_q, X.25 Layer 3 variable, defined, 72-3
m_packet_rrri, SNA variable, defined, 74-3
m_packet_rtti, SNA variable, defined, 74-3
m_packet_ru_category, SNA variable, defined, 74-3
m_packet_sdi, SNA variable, 74-4 defined, 74-3
m_packet_sdu_offset, X.25 Layer 3 variable, 72-8 defined, 72-5
m_packet_type, X.25 Layer 3 variable, defined, 72-4
m_packet_type_byte, X.25 Layer 3 variable, defined, 72-4
m_prot_disc, Q.931 variable, defined, 78-2
m_ptr_to_call_ref, Q.931 variable, 78-4 defined, 78-3
m_ptr_to_info_element, Q.931 variable, 78-4 defined, 78-3
m_sio_ni, SS#7 Layer 3 variable, defined, 80-2
m_sio_priority, SS#7 Layer 3 variable, defined, 80-2
m_sio_si, SS#7 Layer 3 variable, 80-1 defined, 80-2
m_so0, SS#7 Layer 2 variable, 79-4 defined, 79-3
m_unit_type, SS#7 Layer 2 variable, 79-3 defined, 79-2
mkdir, disk I/O routine, defined, 65-33
mpm_info, status structure, defined, 66-5

N
nr_error
LAPD event, defined, 77-3
SDLC event, defined, 73-3
X.25 Layer 2 event, defined, 71-3

ns_error
LAPD event, defined, 77-3
SDLC event, defined, 73-3
X.25 Layer 2 event, defined, 71-3

O
OSI
events
lo_dl_prmtv, 63-17
lo_n_prmtv, 63-20
lo_p_prmtv, 63-29
lo_ph_prmtv, 63-14
lo_s_prmtv, 63-26
lo_t_prmtv, 63-23
m_lo_dl_prmtv, 63-17
m_lo_n_prmtv, 63-20
m_lo_p_prmtv, 63-29
m_lo_ph_prmtv, 63-14
m_lo_s_prmtv, 63-26
m_lo_t_prmtv, 63-23
up_dl_prmtv, 63-14
up_n_prtmv, 63-17
up_p_prtmv, 63-26
up_s_prtmv, 63-23
up_t_prtmv, 63-20
routines, 63-30
  _append_il_buff_list_cnt, 63-43
  _dup_il_buff_list, 63-34
  _dup_il_buff_list_start, 63-33
  _free_il_msg_buff, 63-37
  _get_il_msg_buff, 63-31
  _insert_il_buff_list_cnt, 63-39
  _open_space_in_il_buff, 63-35
  _set_maint_buff_bit, 63-37
  _start_il_buff_list, 63-32
send_dl_prtmv_above, 63-46
send_dl_prtmv_below, 63-53
send_m_dl_prtmv_above, 63-48
send_m_n_prtmv_above, 63-52
send_m_p_prtmv_above, 63-64
send_m_s_prtmv_above, 63-60
send_m_t_prtmv_above, 63-56
send_n_prtmv_above, 63-50
send_n_prtmv_below, 63-57
send_p_prtmv_above, 63-62
send_p_prtmv_below, 63-66
send_ph_prtmv_below, 63-49
send_ph_to_above, 63-44
send_s_prtmv_above, 63-58
send_s_prtmv_below, 63-65
send_t_prtmv_above, 63-54
send_t_prtmv_below, 63-61
structures
  il_buffer, 63-9
  il_list_header, 63-10
  il_list_node, 63-11
  pdu, 63-8
variables, 63-12-63-30
  l2_tick_count, 63-16
  l3_tick_count, 63-19
  l4_tick_count, 63-22
  l5_tick_count, 63-25
  l6_tick_count, 63-28
  l7_tick_count, 63-30
  lo_dl_il_buff, 63-18
  lo_dl_pdu_seg, 63-17
  lo_dl_prtmv_code, 63-17
  lo_dl_prtmv_path, 63-18
  lo_dl_sdu, 63-18
  lo_n_il_buff, 63-21
  lo_n_pdu_seg, 63-20
  lo_n_prtmv_code, 63-20
  lo_n_prtmv_path, 63-21
  lo_n_sdu, 63-21
  lo_p_il_buff, 63-30
  lo_p_pdu_seg, 63-29
  lo_p_prtmv_code, 63-29
  lo_p_prtmv_path, 63-29
  lo_p_sdu, 63-30
  lo_ph_il_buff, 63-15
  lo_ph_msg_buff, 63-37
  lo_ph_pdu_list, 63-39
  lo_ph_pdu_list_start, 63-33
  lo_ph_pdu_offset, 63-30
  lo_ph_pdu_size, 63-30
  lo_ph_prtmv_code, 63-14
  lo_ph_prtmv_path, 63-15
  lo_ph_sdu, 63-15
  lo_s_il_buff, 63-27
  lo_s_pdu_list, 63-26
  lo_s_pdu_list_start, 63-23
  lo_s_pdu_offset, 63-24
  lo_s_pdu_size, 63-24
  lo_s_prtmv_code, 63-26
  lo_s_prtmv_path, 63-27
  lo_s_sdu, 63-27
  lo_t_il_buff, 63-24
  lo_t_pdu_list, 63-23
  lo_t_pdu_list_start, 63-20
  lo_t_pdu_offset, 63-21
  lo_t_pdu_size, 63-21
  lo_t_prtmv_code, 63-23
  lo_t_prtmv_path, 63-24
  lo_t_sdu, 63-24
  m_lo_dl_il_buff, 63-18
  m_lo_dl_pdu_list, 63-17
  m_lo_dl_pdu_list_start, 63-14
  m_lo_dl_pdu_offset, 63-18
  m_lo_dl_pdu_size, 63-18
  m_lo_n_il_buff, 63-21
  m_lo_n_pdu_list, 63-20
  m_lo_n_pdu_list_start, 63-18
  m_lo_n_pdu_offset, 63-21
  m_lo_n_pdu_size, 63-21
  m_lo_p_pdu_list, 63-29
  m_lo_p_pdu_list_start, 63-14
  m_lo_p_pdu_offset, 63-29
  m_lo_p_pdu_size, 63-29
  m_lo_p_prtmv_code, 63-29
  m_lo_p_prtmv_path, 63-30
  m_lo_p_pdu_list, 63-30
  m_lo_p_pdu_list_start, 63-30
  m_lo_p_pdu_offset, 63-30
  m_lo_p_pdu_size, 63-30
  m_lo_ph_il_buff, 63-15
  m_lo_ph_pdu_list, 63-14
  m_lo_ph_pdu_list_start, 63-15
  m_lo_ph_pdu_offset, 63-15
  m_lo_ph_pdu_size, 63-15
  m_lo_s_il_buff, 63-27
  m_lo_s_pdu_list, 63-26
  m_lo_s_pdu_list_start, 63-27
  m_lo_s_pdu_offset, 63-27
  m_lo_s_pdu_size, 63-27
  m_lo_t_il_buff, 63-24
  m_lo_t_pdu_list, 63-23
  m_lo_t_pdu_list_start, 63-21
  m_lo_t_pdu_offset, 63-24
  m_lo_t_pdu_size, 63-24
  m_ph_pdu_list, 63-13
  m_ph_pdu_list_start, 63-13
  m_ph_pdu_offset, 63-13
  m_ph_pdu_size, 63-13
  m_ph_prtmv_code, 63-14
  m_ph_prtmv_path, 63-15
  m_ph_sdu, 63-15

| Structures, Variables, and Routines

Index to C Structures, Variables, and Routines
P

Playback routines
  start_rcrd_play, 65-3
  suspend_rcrd_play, 65-3

Primitives
OSI routines
  Layer 1, 63-44
  Layer 2, 63-46–63-50
  Layer 3, 63-50
  Layer 4, 63-54–63-58
  Layer 5, 63-58–63-62
  Layer 6, 63-62
  Layer 7, 63-66–63-67
  layer-independent, 63-31–63-44
  See also IL buffer

Print routines
  printf, 64-1, 64-4
defined, 64-4
  prints, 64-9

_1open_space_in_il_buff, OSI layer-independent
  routine, 55-9, 63-36, 63-39, 63-42, 63-44, 63-45
  defined, 63-35

outsync_action, line routine, defined, 59-10

set_print_header, 64-8
  sprintf, 64-7
  structures, _print_buffer, 64-2

Program chaining, routines, load_program, 69-12

Protocol Trace
  structures
    l2pp_trbuff_ctl, 61-41
    l3pp_trbuff_ctl, 61-42
  variables
    l2pp_trbuff, 61-39
    l2pp_trbuff_end, 61-39
    l3pp_trbuff, 61-39
    l3pp_trbuff_end, 61-39

packet_sent, X.25 Layer 3 event, defined, 72-3

pdu, OSI structure, 55-4, 63-47, 63-51, 63-55, 63-59
  defined, 63-8

ph_prmtv_type, OSI Layer 1 variable, 63-49
  defined, 63-13

  defined, 61-21

pr_error, X.25 Layer 3 event, defined, 72-3

prev_aux_value, aux port I/O variable, 68-10
  defined, 68-4

prev_date_of_day, real-time clock variable, defined, 69-4

prev_display_screen, status variable, 61-1, 66-1
  defined, 61-3

prev_time_of_day, real-time clock variable, defined, 69-4

prev_tm, real-time clock structure, defined, 69-2

previous_eia_leads
  EIA variable, 60-1, 60-3
  defined, 60-2
  X.21 variable, 70-2
  defined, 70-3

  _print_buffer, print structure, 64-1
  defined, 64-2
  used to check status of print buffer, 64-1

printc, print routine, 64-1, 64-4
  defined, 64-4

printf, print routine, 64-1, 64-6
  defined, 64-4

prints, print routine, 56-22, 64-1
  defined, 64-9

Index to C Structures, Variables, and Routines
probuf, trace buffer structure, defined, 61-25
ps_error, X.25 Layer 3 event, defined, 72-3

Q
Q.931
events
dce_packet, 78-2
dte_packet, 78-2
variables
13_enhance, 78-3
13_suppress, 78-3
m_call_ref_flag, 78-2
m_call_ref_len, 78-3
m_info_element_len, 78-3
m_message_type, 78-3
m_message_type_defined, 78-2
m_packet_bcc_type, 78-2
m_prot_disc, 78-2
m_ptr_to_call_ref, 78-3
m_ptr_to_info_element, 78-3

R
Real-time clock
events, fevar_time_of_day, 69-4
structures
crnt_tm, 69-2
prev_tm, 69-2
tm, 69-2
variables
crnt_date_of_day, 69-4
crnt_time_of_day, 69-4
prev_date_of_day, 69-4
prev_time_of_day, 69-4
Record
routines
start_rcrd_play, 65-3, 69-16
suspend_rcrd_play, 65-3, 69-17
Remote port I/O
events
rmt_break, 67-3
rmt_input_almost_empty, 67-3
rmt_input_almost_full, 67-3
rmt_input_empty, 67-3
rmt_input_not_empty, 67-3
rmt_input_overflow, 67-3
rmt_output_empty, 67-3

rcv_buffer_full, line variable, 59-2
defined, 59-4
rcvd_char_rd, line variable, 55-1, 59-5, 59-6, 59-18
defined, 59-4
rcvd_char_td, line variable, 55-1, 59-5, 59-6,
59-17, 59-19
defined, 59-4
rcvd_device_path, X.25 Layer 3 variable, 72-9
defined, 72-7
rcvd_frame
LAPD event, defined, 77-3
SDLC event, defined, 73-3
X.25 Layer 2 event, defined, 71-3
rcvd_frame_addr
SDLC variable, defined, 73-4
X.25 Layer 2 variable, defined, 71-4
rcvd_frame_addr_cr, LAPD variable, defined,
77-4
rcvd_frame_addr_sapi, LAPD variable, defined,
77-4
rcvd_frame_addr_tei, LAPD variable, defined,
77-4
rcvd_frame_bcc_type
LAPD variable, defined, 77-5
SDLC variable, defined, 73-5
X.25 Layer 2 variable, defined, 71-4
rcvd_frame_buff_seg
LAPD variable, 77-8
defined, 77-5

Index to C Structures, Variables, and Routines
SDLC variable, 73-8 defined, 73-5
X.25 Layer 2 variable, 71-8 defined, 71-5

rcvd_frame_ctrl_byte_1
LAPD variable, defined, 77-5
SDLC variable, defined, 73-4
X.25 Layer 2 variable, defined, 71-4

rcvd_frame_nr
LAPD variable, defined, 77-5
SDLC variable, defined, 73-5
X.25 Layer 2 variable, defined, 71-4

rcvd_frame_ns
LAPD variable, defined, 77-5
SDLC variable, defined, 73-5
X.25 Layer 2 variable, defined, 71-5

rcvd_frame_pf
LAPD variable, defined, 77-5
SDLC variable, defined, 73-4
X.25 Layer 2 variable, defined, 71-4

rcvd_frame_sdu_offset
LAPD variable, defined, 77-8
SDLC variable, defined, 73-8
X.25 Layer 2 variable, defined, 71-8
defined, 71-5

rcvd_frame_sdu_size
LAPD variable, defined, 77-8
SDLC variable, defined, 73-8
X.25 Layer 2 variable, defined, 71-8
defined, 71-5

rcvd_frame_type
LAPD variable, defined, 77-4
SDLC variable, defined, 73-4
X.25 Layer 2 variable, defined, 71-4

rcvd_packet, X.25 Layer 3 event, defined, 72-3
rcvd_packet_ptr, X.25 Layer 3 variable, defined, 72-6
rcvd_packet_type, X.25 Layer 3 variable, defined, 72-5

rcvd_pkt_cause, X.25 Layer 3 variable, defined, 72-4
rcvd_pkt_d, X.25 Layer 3 variable, defined, 72-4
rcvd_pkt_diag, X.25 Layer 3 variable, defined, 72-4
rcvd_pkt_info_length, X.25 Layer 3 variable, defined, 72-9
rcvd_pkt_info_offset, X.25 Layer 3 variable, defined, 72-6
rcvd_pkt_info_ptr, X.25 Layer 3 variable, defined, 72-6
rcvd_pkt_info_seg, X.25 Layer 3 variable, defined, 72-6
rcvd_pkt_info_type, X.25 Layer 3 variable, defined, 72-6
rcvd_pkt二者, X.25 Layer 3 variable, defined, 72-6
rcvd_pkt_length, X.25 Layer 3 variable, defined, 72-8
rcvd_pkt_m, X.25 Layer 3 variable, defined, 72-4
rcvd_pkt_pr, X.25 Layer 3 variable, defined, 72-4
rcvd_pkt_ps, X.25 Layer 3 variable, defined, 72-4
rcvd_pkt_q, X.25 Layer 3 variable, defined, 72-4
rcvd_pkt_sdu_offset, X.25 Layer 3 variable, defined, 72-8
rcvd_pkt_type_byte, X.25 Layer 3 variable, defined, 72-5
rd_modifier, data-display variable, 59-7, 59-18 defined, 59-5
remove, disk I/O routine, 65-3, 65-33 defined, 65-32
rename, disk I/O routine, 65-3, 65-32 defined, 65-31

resend_frame
LAPD routine, defined, 77-10
SDLC routine, defined, 73-10
X.25 Layer 2 routine, defined, 71-10
resend_packet, X.25 Layer 3 routine, defined, 72-14

reset_nr
LAPD routine, defined, 77-11
SDLC routine, defined, 73-11
X.25 Layer 2 routine, defined, 71-11
reset_ns
LAPD routine, defined, 77-12
SDLC routine, defined, 73-12
X.25 Layer 2 routine, defined, 71-11
reset_pr_ps, X.25 Layer 3 routine, defined, 72-15
restore_cursor, Display Window routine, 61-1, 61-8, 61-22
defined, 61-22
rewind, disk I/O routine, 65-2, 65-4
defined, 65-13
rh_ptr, SNA variable, defined, 74-3
rindex, string routine, 56-22
defined, 69-12
rmt_break, remote port I/O event, defined, 67-3
rmt_flush_i, remote port I/O routine, 67-15
defined, 67-8
rmt_flushi, remote port I/O routine, 67-9
defined, 67-22
rmt_flusho, remote port I/O routine, 67-17
defined, 67-16
rmt_get_baud_rate, remote port I/O routine, 67-25
defined, 67-25
rmt_get_bits, remote port I/O routine, 67-26
defined, 67-25
rmt_get_mode, remote port I/O routine, 67-27
defined, 67-26
rmt_get_parity, remote port I/O routine, 67-26
defined, 67-26
rmt_getc, remote port I/O routine, 67-5, 67-9, 67-19
defined, 67-4
rmtgetc, remote port I/O routine, 67-6
defined, 67-5
rmt_gets, remote port I/O routine, defined, 67-6
rmt_input_almost_empty, remote port I/O event, 67-2
defined, 67-3
rmt_input_almost_full, remote port I/O event, 67-2
defined, 67-3
rmt_input_empty, remote port I/O event, 67-2
defined, 67-3
rmt_input_not_empty, remote port I/O event, 67-2
defined, 67-3
rmt_input_overflow, remote port I/O event, 67-2
defined, 67-3
rmt_lock, remote port I/O routine, 67-10
defined, 67-9
rmt_output_empty, remote port I/O event, defined, 67-3
rmt_putb, remote port I/O routine, 67-16
defined, 67-14
rmtputc, remote port I/O routine, 67-12, 67-17
defined, 67-11
rmt_puts, remote port I/O routine, 67-14
defined, 67-13
rmt_resumeo, remote port I/O routine, 67-19
defined, 67-18
rmt_send_break, remote port I/O routine, 67-20
defined, 67-19
rmt_set_baud_rate, remote port I/O routine, 67-22
defined, 67-21
rmt_set_bits, remote port I/O routine, 67-22
defined, 67-22
rmt_set_mode, remote port I/O routine, 67-25
defined, 67-24
rmt_set_parity, remote port I/O routine, 67-23
defined, 67-23
rmt_suspendo, remote port I/O routine, 67-18
defined, 67-17
rmt_unlock, remote port I/O routine, 67-11
defined, 67-10
ru_ptr, SNA variable, defined, 74-3

S
SDLC
events
bcc_error, 73-3
dce_abort, 73-3
dce_bad_bcc, 73-3
dce_frame, 73-3
dce_good_bcc, 73-3
dte_abort, 73-3
dte_bad_bcc, 73-3
dte_frame, 73-3
dte_good_bcc, 73-3
frame_sent, 73-4
invalid_frame, 73-3
l2_T1, 73-3
nr_error, 73-3
ns_error, 73-3
rcvd_frame, 73-3

Index to C Structures, Variables, and Routines
Index to C Structures, Variables, and Routines
String
routines
index, 69–11
rindex, 69–12

send_d_frame
ISDN routine, defined, 76–3
transmit routine, 76–1

send_d_frame_il, ISDN routine, defined, 76–4

send_dl_prmvt_above, OSI Layer 2 routine,
55–4
defined, 63–46

send_dl_prmvt_below, OSI Layer 3 routine,
55–8, 63–42
defined, 63–53

send_frame
LAPD routine, defined, 77–12
SDLC routine, defined, 73–12
SNA routine, 74–5
X.25 Layer 2 routine, defined, 71–12

send_frame_structure
LAPD structure, 77–13
defined, 77–2
SDLC structure, 73–13
defined, 73–2
SNA Layer 2 structure, 74–1
X.25 Layer 2 structure, 71–13
defined, 71–2

send_key, keyboard routine, 3–11, 5–22; 61–4
defined, 69–17

send_m_dl_prmvt_above, OSI Layer 2 routine,
defined, 63–48

send_m_n_prmvt_above, OSI Layer 3 routine,
defined, 63–52

send_m_p_prmvt_above, OSI Layer 6 routine,
defined, 63–64

send_m_s_prmvt_above, OSI Layer 5 routine,
defined, 63–60

send_m_t_prmvt_above, OSI Layer 4 routine,
defined, 63–56

send_n_prmvt_above, OSI Layer 3 routine,
defined, 63–50

send_n_prmvt_below, OSI Layer 4 routine,
55–7, 55–8, 63–41, 63–58
defined, 63–57

send_p_prmvt_above, OSI Layer 6 routine,
defined, 63–62

send_p_prmvt_below, OSI Layer 7 routine,
defined, 63–66

send_packet, X.25 Layer 3 routine, defined, 72–16

send_packet_structure, X.25 Layer 3 structure,
72–17
defined, 72–2

send_ph_prmvt_below, OSI Layer 2 routine,
55–8, 63–39, 63–43, 63–44
defined, 63–49

send_ph_to_above, OSI Layer 1 routine, 63–46
defined, 63–44

send_s_prmvt_above, OSI Layer 5 routine,
defined, 63–58

send_s_prmvt_below, OSI Layer 6 routine,
defined, 63–65

send_stat_message, stats–display routine, 62–15,
62–16
defined, 62–15

send_t_prmvt_above, OSI Layer 4 routine,
defined, 63–54

send_t_prmvt_below, OSI Layer 5 routine,
63–41
defined, 63–61

set_auxCtl_leads, aux port I/O routine, 68–6,
68–8, 68–9
defined, 68–6

set_aux_direction, aux port I/O routine, 68–5,
68–8, 68–9
defined, 68–5

set_aux_reg, aux port I/O routine, 68–11
defined, 68–10

_set_file_type, disk I/O routine, 65–35
defined, 65–34

set_isdn_speaker_chan, ISDN routine, 76–2
defined, 76–5

_set_maint_buff_bit, OSI layer–independent
type, 55–4, 55–5, 63–39, 63–41, 63–42,
63–43, 63–44, 63–46, 63–48, 63–50,
63–51, 63–52, 63–54, 63–56, 63–57, 63–59
defined, 63–37

set_print_header, print routine, 64–1, 64–9
defined, 64–8

set_tcr_b
transmit routine, defined, 59–14
X.21 routine, 70–8
defined, 70–8

Index to C Structures, Variables, and Routines
signal, interrupt routine, 57-6
  defined, 69-15
signal_name, interrupt event, defined, 69-4
sound_alarm, alarm routine, defined, 69-16
sprintf
  Display Window routine, defined, 61-13
  print routine, 61-13, 64-1, 64-7
  defined, 64-7
  used to specify precision for tracef, 67-16
_start_il_buf_list, OSI layer-independent routine, 55-5, 55-8, 59-13, 63-36, 63-41, 63-45, 71-13, 72-17, 73-13, 76-5, 77-13
start_of_run_date, timer variable, defined, 62-10
start_of_run_time, timer variable, defined, 62-10
start_rccrd_play
  playback routine, 65-3
  record routine, 14-6; 65-3
  defined, 69-16
stat_msg, stats-display structure, 62-15
  defined, 62-5
stracef, trace buffer routine, 61-13, 61-30
  defined, 61-29
strcmp, user-defined routine, 58-6
suspend_rccrd_play
  playback routine, 65-3
  record routine, 65-3
  defined, 69-17

T

Timeout
  events, timeout_name_expired, 69-4
  routines
    timeout_restart_action, 69-8
    timeout_stop_action, 69-10
  structures, timeout, 69-3
Timer
  routines
    convert_tick_count, 62-18
    get_wall_time_286_ticks, 62-17
    get_wall_time_ticks, 62-16
    structures, timer_struct, 62-9
  variables
    l1_tick_count, 62-10
    start_of_run_date, 62-10
    start_of_run_time, 62-10
Trace buffer
  routines
    stracef, 61-29
    tracec, 61-27
    tracef, 61-28
    traces, 61-31
  structures
    l1_trbuf, 61-25
    l2_trbuf, 61-25
    l3_trbuf, 61-25
    l4_trbuf, 61-25
    l5_trbuf, 61-26
    l6_trbuf, 61-26
    l7_trbuf, 61-26
    prog_trbuf, 61-25
    trace_buf, 61-24
    trace_buffer_header, 61-24
  Transmission
  events, fevar_xmit_cmplt, 59-4
  routines
    idle_action, 59-13
    l1_il_transmit, 55-4, 59-12
    l1_transmit, 59-11
    set_tcr_b, 59-14
  structures, xmit_list, 59-1
  td_modifier, data-display variable, 59-7, 59-17
  defined, 59-4
temporary_prompt, user-defined routine, 58-4
th_ptr, SNA variable, defined, 74-3
timeout, timeout structure, 58-5
  defined, 69-3
timeout_name_expired, timeout event, defined, 69-4
timeout_restart_action, timeout routine, 58-4, 69-9
  defined, 69-8
timeout_stop_action, timeout routine, 69-10
  defined, 69-10
timer_struct, timer structure, 62-8
  defined, 62-9
tm, real-time clock structure, defined, 69-2
trace_buf, trace buffer structure, 61-23, 61-29,
  61-30, 61-35
  defined, 61-24
trace_buffer_header, trace buffer structure, 61-23, 61-32
  defined, 61-24
tracec, trace buffer routine, 61-1, 61-4, 61-28
  defined, 61-27
tracef, trace buffer routine, 61-1, 61-4, 61-27,
  61-29, 61-32, 61-34, 61-35, 65-18
  defined, 61-28

Index to C Structures, Variables, and Routines
traces, trace buffer routine, 56–22, 61–1, 61–4
defined, 61–31

U
ungetc, disk I/O routine, 65–1, 65–2, 65–4, 65–22
defined, 65–21
unit_config, status structure, 11–7; 66–1
defined, 66–3
unit_setup, status structure, 66–1
defined, 66–2
unlock
disk I/O routine, 65–4
defined, 65–16
lock routine, defined, 69–14
up_dl_il_buff, OSI Layer 2 variable, 63–38, 63–42, 63–44, 63–50
defined, 63–16
up_dl_pdu_seg, OSI Layer 2 variable, defined, 63–15
up_dl_prmtv, OSI Layer 2 event, defined, 63–14
up_dl_prmtv_code, OSI Layer 2 variable, 63–53
defined, 63–16
up_dl_prmtv_path, OSI Layer 2 variable, 63–49
defined, 63–16
up_dl_sdu, OSI Layer 2 variable, 63–42, 63–44, 63–50
defined, 63–16
up_n_il_buff, OSI Layer 3 variable, 55–7, 63–5, 63–8, 63–9, 63–10, 63–11, 63–42, 63–54
defined, 63–19
up_n_pdu_seg, OSI Layer 3 variable, 63–8
defined, 63–19
up_n_prmtv, OSI Layer 3 event, 55–7
defined, 63–17
up_n_prmtv_code, OSI Layer 3 variable, 63–8, 63–57
defined, 63–19
up_n_prmtv_path, OSI Layer 3 variable, 63–8, 63–53
defined, 63–19
up_n_sdu, OSI Layer 3 variable, 55–7, 63–8, 63–10, 63–42, 63–54
defined, 63–19
up_p_il_buff, OSI Layer 6 variable, 63–65, 63–66
defined, 63–28
up_p_pdu_seg, OSI Layer 6 variable, defined, 63–27
up_p_prmtv, OSI Layer 6 event, defined, 63–26
up_p_prmtv_code, OSI Layer 6 variable, 63–67
defined, 63–28
up_p_prmtv_path, OSI Layer 6 variable, 63–65
defined, 63–28
up_p_sdu, OSI Layer 6 variable, 63–65, 63–66
defined, 63–28
up_s_il_buff, OSI Layer 5 variable, 63–61, 63–62
defined, 63–25
up_s_pdu_seg, OSI Layer 5 variable, defined, 63–24
up_s_prmtv, OSI Layer 5 event, defined, 63–23
up_s_prmtv_code, OSI Layer 5 variable, 63–65
defined, 63–25
up_s_prmtv_path, OSI Layer 5 variable, 63–61
defined, 63–25
up_s_sdu, OSI Layer 5 variable, 63–61, 63–62
defined, 63–25
up_t_il_buff, OSI Layer 4 variable, 63–41, 63–57
defined, 63–22
up_t_pdu_seg, OSI Layer 4 variable, defined, 63–22
up_t_prmtv, OSI Layer 4 event, defined, 63–20
up_t_prmtv_code, OSI Layer 4 variable, 63–61
defined, 63–22
up_t_prmtv_path, OSI Layer 4 variable, 63–57
defined, 63–22
up_t_sdu, OSI Layer 4 variable, 63–41, 63–57
defined, 63–22

W
window_color, Display Window variable, 61–4, 61–32, 61–33, 61–35
defined, 61–5
windowModifier, Display Window variable, 61–4, 61–32, 61–35
defined, 61–7
write_aux, aux port I/O routine, 68–8, 68–9
defined, 68–7

Index to C Structures, Variables, and Routines
X

X.21
events, fevar_eia_changed, 70-3
routines
ctl_eia, 70-4
enter_call_phase, 70-9
enter_data_phase, 70-9
set_tcr_b, 70-8
x21_idle_action, 70-6
x21_transmit_call, 70-7
structures, xmit_list, 70-1
variables
current_eia_leads, 70-3
previous_eia_leads, 70-3

X.25 Layer 2
events
bcc_error, 71-3
dce_abort, 71-3
dce_bad_bcc, 71-3
dce_frame, 71-3
dce_good_bcc, 71-3
dte_abort, 71-3
dte_bad_bcc, 71-3
dte_frame, 71-3
dte_good_bcc, 71-3
frame_sent, 71-4
invalid_frame, 71-3
l2_T1, 71-3
nr_error, 71-3
ns_error, 71-3
rcvd_frame, 71-3
routines
l2_give_data, 71-9
resend_frame, 71-10
reset_nr, 71-11
reset_ns, 71-11
send_frame, 71-12
structures, send_frame_structure, 71-2
variables
l2_current_window_edge, 71-5
l2_enhance, 71-5
l2_lower_window_edge, 71-5
l2_resend_edge, 71-5
l2_suppress, 71-5
l2_upper_window_edge, 71-5
m_frame_addr, 71-4
m_frame_bcc_type, 71-4
m_frame_cntrl_byte_1, 71-4
m_frame_pf, 71-4
m_frame_type, 71-4
rcvd_frame_addr, 71-4
rcvd_frame_bcc_type, 71-4
rcvd_frame_buff_seg, 71-5
rcvd_frame_cntrl_byte_1, 71-4

rcvd_frame_nr, 71-4
rcvd_frame_ns, 71-5
rcvd_frame_pf, 71-4
rcvd_frame_sdu_offset, 71-5
rcvd_frame_sdu_size, 71-5
rcvd_frame_type, 71-4

X.25 Layer 3
events
dce_packet, 72-3
dte_packet, 72-3
invalid_packet, 72-3
packet_sent, 72-3
pr_error, 72-3
ps_error, 72-3
rcvd_packet, 72-3
routines
l3_clear_path, 72-11
l3_give_data, 72-10
l3_more_to_resend, 72-11
l3_window_empty, 72-13
l3_window_full, 72-12
resend_packet, 72-14
reset_pr_ps, 72-15
send_packet, 72-16
structures, send_packet_structure, 72-2
variables
l3_enhance, 72-7
l3_suppress, 72-7
m_packet_buff_seg, 72-5
m_packet_cause, 72-4
m_packet_d, 72-3
m_packet_diag_code, 72-4
m_packet_info_length, 72-5
m_packet_info_offset, 72-5
m_packet_info_ptr, 72-6
m_packet_info_seg, 72-5
m_packet_lcn, 72-3
m_packet_lcn_grp, 72-3
m_packet_length, 72-5
m_packet_m, 72-3
m_packet_pr, 72-3
m_packet_ps, 72-3
m_packet_ptr, 72-6
m_packet_q, 72-3
m_packet_sdu_offset, 72-5
m_packet_type, 72-4
m_packet_type_byte, 72-4
rcvd_device_path, 72-7
rcvd_packet_ptr, 72-6
rcvd_packet_type, 72-5
rcvd_pkt_buff_seg, 72-5
rcvd_pkt_cause, 72-4
rcvd_pkt_d, 72-4
rcvd_pkt_diag, 72-4
rcvd_pkt_info_length, 72-6
rcvd_pkt_info_offset, 72-6

Index to C Structures, Variables, and Routines
rcvd_pkt_info_ptr, 72-6
rcvd_pkt_info_seg, 72-6
rcvd_pkt_lcn, 72-4
rcvd_pkt_length, 72-6
rcvd_pkt_m, 72-4
rcvd_pkt_pr, 72-4
rcvd_pkt_ps, 72-4
rcvd_pkt_q, 72-4
rcvd_pkt_sdu_offset, 72-6
rcvd_pkt_type_byte, 72-5

x21_idle_action, X.21 routine, defined, 70-6
x21_transmit_call, X.21 routine, 70-1, 70-7
defined, 70-7

xmit_list
ISDN structure, 76-3
defined, 76-1
transmit structure, 59-11
defined, 59-1
X.21 structure, 70-6, 70-7
defined, 70-1