51 Data Flow

# 51 Data Flow



٠

.

51 Data Flow

# **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.



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. INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

51-6

# 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 will 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 *waitfor* clauses; CONDITIONS into *waitfor* expressions that include event variables; and ACTIONS into statements and routines, also inside of *waitfor* clauses.

Then the translator creates a program 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.

```
STATE: sample1
CONDITIONS: TIME 1300
ACTIONS: ALARM
```

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

```
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 ();
}
```

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.

```
{
  extern fast event fevar time of day;
  extern volatile unsigned short crnt_time_of_day;
  task
  {
      main ()
         state_sample1:
         waitfor
         ł
            fevar_time_of_day && (crnt_time_of_day == 1300):
               sound alarm ();
         }
     }
  } 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  $c_test$ ).

```
extern fast_event fevar_time_of_day;
extern volatile unsigned short crnt_time_of_day;
task
{
  main ()
  ł
      state_sample1:
      waitfor
      {
         fevar_time_of_day && (crnt_time_of_day == 1300):
            sound_alarm ();
         }
      }
  }
} c_test;
task
{
   main ()
      state_task_list:
      {
         c_test ();
                                   This empty waitfor is automatically generated in any state
          waitfor
                                   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\_ctest.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\_ctest.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:

```
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  $layer_l_test$  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.



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) */
    }
    goio (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:

```
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:

```
static label current state;
state_alarm_on:
current_state = state_alarm_on_loop;
ł
   goto (current_state);
  state_alarm_on_loop:
   waitfor
   goto (current_state);
}
```



Figure 53-2 Basic C structure of a spreadsheet state.

If the user wants to redefine *current\_state*, he may do so in the nonloop state, in which case the loop (*waitfor*) state will be bypassed:

```
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:

```
static label current_state;
state alarm on:
current_state = state_alarm_on_loop;
ł
  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;
        }
    }
    goto (current_state);
```

Various hybrid versions are possible. Here is one:

```
STATE: alarm on
  CONDITIONS:
      keyboard_new_any_key && (keyboard_any_key ==
                                                           ')
  1
  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 \*/



Figure 53-3 The translator converts the Condition-and-Action "trigger" into a condition clause inside of a *waitfor* statement.

#### (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.

```
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.");
        }
    }
}
```

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

1

}

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

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:

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 *and*ing 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 *and*ed 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:

fevar\_gd\_bcc\_td:

{

ł

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 EIA 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:

```
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 *and*ed components is zero:

```
{
  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:

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."
{
  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.)

```
extern fast event keyboard_new_any_key;
extern volatile unsigned short keyboard_any_key;
task
   main()
   ł
      static label current state;
      state_alarm_on:
      current_state = state_alarm_on_loop;
         goto (current_state);
         state_alarm_on_loop:
      waitfor
         ł
            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 == ' '):

INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108



# 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_any_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 68k 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:

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.



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:

{
 extern fast\_event fevar\_eia\_changed;
}
LAYER: 1
 TEST: leads
 STATE: init
 CONDITIONS:
 {
 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.

An end and the control of a control of process control and the second control of a second control of the se

as si e di la contratazione del politare con contrata di contratazione di **l'eran scalta** del carinato a secono di ditta di la concontro di contrato e antaziana di ano di anta terre sono di contro di carona cola contro di contrato para contrato a antazi sana recontrato

#### international (23)

Ubstantable statemictory many mutation. For lensing different de la statementation and an an Europe 2 of the properties have functions. Actions (no fraccion of the track of forwels do not the advector for a size material for the relation of the definition of the state of the analysis of medifier scitterings. The integrational of a neuroscience of the lensities of second

# 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:

fevar\_time\_of\_day && (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.





# 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\_gd\_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 interrupts/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.

INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

# 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  $ll_transmit$  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  $ll_il_transmit$  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:

char \* m\_frame\_ptr; m\_frame\_ptr = (void\*)(((long)m\_lo\_ph\_il\_buff << 16) + 32);</pre>



PDU



# 55.2 Monitor Path vs. Receive Path

The variables  $m_lo_ph_prmtv$ ,  $m_lo_ph_il_buff$ , and  $m_lo_ph_sdu_offset$  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:  $lo_ph_prmtv$ ,  $lo_ph_il_buff$ ,  $lo_ph_sdu$ , 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  $m_lo_dl_sdu_offset$ . 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  $m_lo_dl_ibuff$  and  $m_lo_dl_sdu_offset$  to construct its packet trace.

The size of the data component in the buffer is stored in the variable  $m\_lo\_dl\_sdu\_size$ . This number will be 2 bytes smaller than the variable  $m\_lo\_ph\_sdu\_size$ .

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  $m_{\rm o}$  are updated automatically in order to drive the protocol traces. Receive-path variables such as  $lo_{\rm o}dl_{\rm o}rmtv$ ,  $lo_{\rm o}dl_{\rm o}l_{\rm o}dl_{\rm o}du$  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:

lo\_dl\_il\_buff = lo\_ph\_il\_buff; lo\_dl\_sdu = (lo\_ph\_sdu + 2); pdu\_ptr->data\_length = (pdu\_ptr->data\_length - 2); signal (lo\_dl\_prmtv);

The same updates of variables and the same signal would be generated if the user called a *send\_dl\_prmtv\_above* routine, as follows:

\_set\_maint\_buff\_bit (lo\_ph\_il\_buff, &l2\_relay\_baton); send\_dl\_prmtv\_above (lo\_ph\_il\_buff, l2\_relay\_baton, lo\_ph\_sdu + 2, pdu\_ptr->data\_length - 2, 0x45);

The send\_dl\_prmtv\_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\_prmtv\_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.

```
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: fox
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);
    l1_il_transmit(bufnum,baton,list_hd_offset,transmit_tag);
}
```

The transmit string will look like this on the INTERVIEW's data display:

토 N L L N THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG 0123456789 6

### (A) Segment Number

The *ll\_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 <u>get\_il\_msg\_buff</u> 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 <u>set\_maint\_buff\_bit</u> 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 <u>free\_il\_msg\_buff</u> routine) until the outstanding frame or packet has been acknowledged.

### (C) List-Header Offset

In addition to buffer number and baton number, the *ll\_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.



Figure 55-2 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 *l1\_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:

{
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};

```
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:
   ł
    _get_il_msg_buff(&bufnum,&l4_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, l4_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,&l3_baton);
    send_dl_prmtv_below(up_n_il_buff,13_baton,up_n_sdu,0,0x44,0);
   3
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,&l2_baton);
    send_ph_prmtv_below(up_dl_il_buff,l2_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:

l1\_il\_transmit(up\_dl\_il\_buff,l2\_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 *ll\_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 <u>open\_space\_in\_il\_buff</u> 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 <u>\_insert\_il\_buff\_list\_cnt</u> 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  $\frac{1}{5}$  and  $\frac{1}{5}$  or  $\frac{1}{5}$ . 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  $\frac{1}{5}$  and  $\frac{1}{5}$  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.

```
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] = {'$\vert ', '$\vert '};
}
STATE: look for stx
   CONDITIONS:
     fevar_rcvd_char_td && rcvd_char_td == '$x'
   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 == '<sup>E</sup>x') || (rcvd_char_td == '<sup>E</sup>b'))
{
    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(ii_buffer_number, &data_start_offset);
    _insert_il_buff_list(ii_buffer_number, data_start_offset, echo_string, length);
    l1_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.

```
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 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 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);
     }
```

56 C Basics

## 56 C Basics





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 reme to terminate the directive (no indication of the Return will appear on the screen). Type  $\$  (backslash) and press reme 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  $\blacksquare$ .)

## (A) #define

The *#define* directive gives you the convenience of replacing frequently referenced 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. *lusr/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. *lsys/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  $\Leftrightarrow$  or "" delimiters may be a relative pathname. The highest directory in the pathname must reside in the current directory or in one of the *linclude* directories. In response to an *#include* "disk\_io/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 *llib* 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 \_\_\_\_\_\_FRE\_\_\_\_\_, 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 **SPREADSHEET** 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 no incorporate them in code that will be compiled and saved as an LOBJ file. Although the code will compile, no search fro 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 0directive.

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 for 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.

#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
    {
       fevar_time_of_day && (crnt_time_of_day == 1300):
        {
         sound_alarm();
        3
    }
  }
} example_task;
example()
З
 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 fevar_time_of_day;
extern volatile unsigned short crnt_time_of_day;
example()
{
    waitfor
    {
        fevar_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:

```
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:

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:

```
{
  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 fffa, 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 equaling -6, b equaling 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 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.

vpe	char conversion	signed decimal conversion		unsigned decimal conversion		hex conversion	
150	(%c)	short (%d)	long (%ld)	shor%u)	long (%lu)	short (%x)	long (%lx)
				· · ·			
har <sup>1</sup>	ზ to Ж	0 to 255	-	0 to 255	-	0 to ff	- ,
signed char <sup>1</sup>	ზ to <i>Ж</i>	-128 to 127	-	0 to 127 and 65408 to 65535	-	0 to 7f and ff80 to ffff	-
Insigned char <sup>1</sup>	∿ to Ж	0 to 255	-	0 to 255	-	0 to ff	-
t	-	-32768 to 32767	-	0 to 65535	-	0 to ffff	-
igned int	-	-32768 to 32767	-	0 to 65535	-	0 to ffff	-
insigned int	-	-32768 to 32767	_	0 to 65535	_	0 to ffff	<b>–</b> 2
nort	-	-32768 to 32767	-	0 to 65535	-	0 to ffff	·
igned short	-	-32768 to 32767	-	0 to 65535	_	0 to ffff	-
insigned short	-	-32768 to 32767	-	0 to 65535	_	0 to ffff	-
ng	-	-	-2147483648 to 2147483647	-	0 to 4294967295	-	0 to fffffffff
igned long	_	_	-2147483648 to 2147483647	-	0 to 4294967295	-	0 to fffffffff
unsigned long	_	-	-2147483648 to 2147483647	, _	0 to 4294967295	-	0 to fffffffff

# Table 56-1Data Types: Ranges of Values Displayed and Printed

<sup>1</sup> 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:

```
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:

```
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:

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

Operator	Type of Operator	Associativity	
0	primary expression	left to right	
[]> ++	postfix	left to right	
++ sizeof & * + !	unary	right to left	
(type)	cast	left to right	
* / %	multiplicative	left to right	
+ -	additive	left to right	
<< >>	bitwise shift	left to right	
< > <= >=	relational	left to right	
== !=	equality	left to right	
&	bitwise AND	left to right	
^	bitwise exclusive OR	left to right	
	bitwise inclusive OR	left to right	
&&	logical AND	left to right	
11	logical OR	left to right	
?:	conditional	right to left	
= *= /= %= += -= <<= >>= &= ^=  =	assignment	right to left	
,	comma	left to right	

Table 56-2 Operator Precedence<sup>1</sup>

<sup>1</sup> 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:

STATE: right\_to\_left\_associativity
{
 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.

```
STATE: right_to_left_associativity
{
    int a = 3;
    3 = a;
    displayf ("%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:

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:

displayf ("%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:

char string\_array [] = "hello"; displayf (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:

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:

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:

```
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:

```
STATE: display_size_of_string
{
    char string_array [] = "hello";
    int compiler_count = sizeof(string_array);
    displayf ("%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

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

Non-literal	Meaning	ASCII character	Hex character
\a	bell	n gasta (* 1997). <b>B</b> L	0 <sub>7</sub>
<b>\b</b>	backspace	ang daa madha 🙀 sanaa siyaa siya	08
٨f	form feed	s en	°c
Nn	linefeed †	n standard an s	° <sub>A</sub>
$\sim 10^{-1} { m Vr}^{-1}$	carriage return		° <sub>D</sub>
$\mathrm{d} \mathrm{d} \mathrm{d} \mathrm{d} \mathrm{x} \mathrm{t} \mathrm{d} \mathrm{d} \mathrm{d} \mathrm{d} \mathrm{d} \mathrm{d} \mathrm{d} d$	horizontal tab	ses des séc <mark>i</mark> s contractéries	୍କ
<b>\v</b>	vertical tab	, baareeraa yy ba baareeraa	о <sub>в</sub>
$\mathbf{N}$	single quote	e de la construcción de la constru	2 <sub>7</sub>
\ <b>"</b>	double quote †	7	<sup>2</sup> 2
	backslash †		5 <sub>C</sub>
\###	octal representation	any ASCII character	°0 - <sup>F</sup> F
\x###	hex representation	any ASCII character	°0 - <sup>F</sup> F

Table 56-3 C String Non-Literals

† 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 "Nabc". You initialize an array as follows:

char string[] = "\x0abc";

The string will be stored as "+c" (hexadecimal characters  ${}^{6}_{B}{}^{6}_{3}$ ). 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 mex 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 <sup>5</sup>c<sup>5</sup>c.

## (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:

1. ANSI Document X3J11/86-098. Proposed American National Standard for Information Systems-Programming Language C.

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

- 2. Darnell, Peter A., and Margolis, Philip E. Software Engineering in C. New York: Springer-Verlag, 1988.
- 3. Harbison, Samuel P., and Steele, Guy L., Jr. C: A Reference Manual. 2d ed. Englewood Cliffs: Prentice-Hall 1987.
- 4. Kernighan, Brian W., and Ritchie, Dennis M. The C Programming Language. 2d ed. Englewood Cliffs: Prentice-Hall, 1988.

# **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 *displayf* 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.

```
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. {

}

```
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:

```
{
  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 CONDITIONS: 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:

{ waitfor ł 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.

```
{
  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 *extern event*. Once the event variable has been declared, he can use the *signal* function to indicate that the event has occurred. Here is an example of an event variable called *check\_number* that causes the nonevent variable *number* to be checked—and sounds the alarm when the value of *number* satisfies the condition.

```
{
  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. If statements, for loops, while loops, and other conditional control structures may not be used in Conditions blocks (that is, in *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 *if* statement at the top of the State block, above any *waitfor* clauses, is how the translator converts ENTER\_STATE softkey conditions into C.)

- 2. Event variables are designed for use in Conditions blocks (*waitfor* clauses) only. It makes no sense to use an event variable in an *if* statement, *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 (*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.

{
 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  $f_{\rm R}$ .

char juliet = 'j';
}
STATE: display\_equivalent\_values
{
 displayf ("%o %d %x %X %c %#u ", juliet, juli

## 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 *and*ing this variable with the care mask for DTR will be as follows:

> 11100110 & <u>00010000</u> 00000000

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

11110110 & <u>00010000</u> 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 *and*ing 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:

```
{
  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:

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:

if ((current\_eia\_leads & 0x08) == 0) sound\_alarm();

This if statement will detect DTR on and DSR on:

if ((current\_eia\_leads & 0x18) == 0) sound\_alarm();

This if statement will detect DTR off and DSR on:

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;
}
STATE: display\_string
 {
 displays (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;
}
LAYER: 3
STATE: display_data
CONDITIONS: DTE DATA
ACTIONS:
first_data_address = ((long) m_lo_ph_il_buff << 16) + 37;
displays (first_data_address);
}

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).

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 *displayc* or *displayf* 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.

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.

```
{
  extern volatile unsigned short m_lo_ph_il_buff;
  char * th;
}
LAYER: 2
STATE: th_pointer
  CONDITIONS: DTE INFO
    ACTIONS:
    {
      th = (void *) (((long) m_lo_ph_il_buff << 16) + 34);
      if ((th[2] == 5) && (th[3] == 1))
      {
          ctl_capture_td (0x10);
            ctl_capture_rd (0x100);
            sound_alarm ();
      }
    }
}</pre>
```

# 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*:

```
{
    long keyrow;
}
STATE: assign_string_address_to_variable
{
    keyrow = "QWERTYUIOP";
}
```

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:

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

or

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:

displays ("QWERTYUIOP");

or

displays (keyrow);

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

char \* keyrow = "QWERTYUIOP";

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

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

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

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:

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:

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

Create a trigger to look for a line event (such as the event variable *dte\_packet*) that will initialize the pointer.

```
{
  extern volatile unsigned char * m_packet_ptr;
  char search_string [] = { 0x10, 0x04, 0x0b };
  extern event dte_packet;
}
LAYER: 3
STATE: match_packet_string
        CONDITIONS:
        {
            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.

```
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 ( $4 \times 12$ ). 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 maximum; 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*.

Here is a sample trigger that displays this variable whenever the operator presses T:

```
STATE: display_total_dce_good_bcc
CONDITIONS: KEYBOARD "Tt"
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

ł

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.

```
struct il_buffer
{
    unsigned short lock;
    unsigned short maintain_bits;
    unsigned short buffer_size;
    unsigned short transmit_tag;
    unsigned long char_buff_frame_start;
    unsigned long char_buff_frame_end;
    unsigned short tick_count_high;
    unsigned short tick_count_low;
    unsigned short tick_count_low;
    unsigned short toytes_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:

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 -> 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.

```
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_II_buffers
      CONDITIONS: DTE INFO
      ACTIONS:
      {
```

ł

}

}

buffer\_ptr = (void \*) ((long) m\_lo\_ph\_il\_buff <<16); pos\_cursor (0,0); displayf ("%02x", buffer\_ptr->data[4]); INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

# **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*:

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

A string of characters declared inside of double-quotation 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:

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 *l3\_window\_full* routine is declared automatically by the translator after the user has made a WINDOW FULL softkey entry. The synopsis for *l3\_window\_full* shows how it is declared:

extern unsigned char l3\_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:

if (13\_window\_full(path\_number) != 0) sound\_alarm();

Here is a simpler coding for the same test:

if (13\_window\_full(path\_number)) sound\_alarm();

This coding works for the same reason that if (1) sound\_alarm(); or if (10) 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.

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.

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

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

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:

```
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:

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.

```
{
  struct
   ł
     unsigned long event_id;
     unsigned short event_id_uid;
  timeout_prompt;
  void blank_prompt_line()
   ł
     pos_cursor(0,0);
                                                                   ");
      displays ("
  void temporary_prompt (string_pointer, seconds)
   char string_pointer [];
   char seconds;
   ł
      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.

```
ł
 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%s", binary_string);
 }
}
   STATE: binary
     CONDITIONS: { dce_packet }
     ACTIONS:
     ł
      display_binary (m_packet_ptr[2]);
```

ł

## (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 "5+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*).

```
ł
 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;
 ł
   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.

		and the second	
Туре	Variable	Value (hex/decimal	) Meaning
		n an	
Structure Nam	<u>e:</u> xmit_list		Structure of a transmit list for <i>11_transmit</i> routine. Declared as type <i>struct</i> . Reference member variables of the structure as follows: <i>xmit_list.string_length</i> .
unsigned char *	string		pointer to the location of the transmit string—the transmit string is declared separately
unsigned short	string_length	0-ffff10-65535	length of the transmit string

## Table 59-1 Transmit Structures

# 59.2 Variables

# (A) Monitoring Events

 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\_rd, fevar\_abort\_td, fevar\_parity\_rd, fevar\_parity\_td, 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.

extern fast_event       fevar_rcvd_char_rd         extern fast_event       fevar_rcvd_char_td         extern fast_event       fevar_gd_bcc_rd         extern fast_event       fevar_gd_bcc_td         extern fast_event       fevar_bd_bcc_rd         extern fast_event       fevar_bd_bcc_td         extern fast_event       fevar_bd_bcc_td         extern fast_event       fevar_abort_rd         extern fast_event       fevar_abort_td         extern fast_event       fevar_parity_rd         extern fast_event       fevar_parity_td	decimal) Meaning
extern fast_event fevar_gd_bcc_rd extern fast_event fevar_gd_bcc_td extern fast_event fevar_bd_bcc_rd extern fast_event fevar_bd_bcc_td extern fast_event fevar_abort_rd extern fast_event fevar_abort_td extern fast_event fevar_parity_rd extern fast_event fevar_parity_rd	True for each character received on RD. Line Setup configured for emulate or monitor mode.
extern fast_event fevar_gd_bcc_td extern fast_event fevar_bd_bcc_rd extern fast_event fevar_bd_bcc_td extern fast_event fevar_abort_rd extern fast_event fevar_abort_td extern fast_event fevar_parity_rd extern fast_event fevar_parity_rd	True for each character received on TD. Line Setup configured for emulate or monitor mode.
extern fast_event fevar_bd_bcc_rd extern fast_event fevar_bd_bcc_td extern fast_event fevar_abort_rd extern fast_event fevar_abort_td extern fast_event fevar_parity_rd extern fast_event fevar_parity_td	True when a good BCC is calculated for an RD block or frame. Line Setup configured for emulate or monitor mode.
extern fast_event       fevar_bd_bcc_td         extern fast_event       fevar_abort_rd         extern fast_event       fevar_abort_td         extern fast_event       fevar_parity_rd         extern fast_event       fevar_parity_td	True when a good BCC is calculated for a TD block or frame. Line Setup configured for emulate or monitor mode.
extern fast_event fevar_abort_rd extern fast_event fevar_abort_td extern fast_event fevar_parity_rd extern fast_event fevar_parity_td	True when a bad BCC is calculated for an RD block or frame. Line Setup configured for emulate or monitor mode.
extern fast_event fevar_abort_td extern fast_event fevar_parity_rd extern fast_event fevar_parity_td	True when a bad BCC is calculated for a TD block or frame. Line Setup configured for emulate or monitor mode.
extern fast_event fevar_parity_rd extern fast_event fevar_parity_td	True when an abort is detect in an RD frame. Line Setup configured for emulate or monitor mode.
extern fast_event fevar_parity_td	True when an abort is detect in a TD frame. Line Setup configured for emulate or monitor mode.
	True when a parity error is detected for an RD byte. Lir Setup configured for emulate monitor mode.
four free orror rd	True when a parity error is detected for a TD byte. Line Setup configured for emulate monitor mode.
extern fast_event fevar_frm_error_rd	True when an async framing error is detected for an RD byte. Line Setup configured emulate or monitor mode.
extern fast_event fevar_frm_error_td	True when an async framing error is detected for a TD by Line Setup configured for emulate or monitor mode.

Table 59-2Monitor/Transmit Variables

Туре	Variable	Value (hex/de	ecimal) Meaning
extern fast_event	fevar_xmit_cmplt	-	True when the INTERVIEW puts a transmission out onto the link. Line Setup configured for emulate mode only.
extern fast_event	fevar_rcv_buffer_full		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.
extern volatile unsigned short	rcv_buffer_full	0	not full full Line Setup configured for emulate or monitor mode.
extern unsigned short	rcvd_char_td		Most recent TD character is stored in this variable. Line Setup configured for emulate or monitor mode.
		0-ff/0-255 100/256 101/257 102/258 103/259	data character (lower byte in 16-blt data word in data buffer) good or bad BCC flag sync abort
extern unsigned short	rcvd_char_rd		Most recent RD character is stored in this variable. Line Setup configured for emulate or monitor mode.
		0-ff/0-255 100/256 101/257 102/258 103/259	data character (lower byte in 16-bit data word in data buffer) good or bad BCC flag sync abort
extern unsigned char	td_modifier		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.
		1 2 4 8 10/16 20/32 40/64	data—initial value (always included in value of td_modifier alternate code set underline (rd character) reverse image hexadecimal low intensity blink

# Table 59-2 (continued)

		and the second secon	
Туре	Variable	Value (hex/deci	imal) Meaning
extern unsigned char	rd_modifier		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.
		en y ser de la 194	data (always included in value of
			rd modifier)
		2	alternate code set
		4	underline (rd character)—initial value of rd_modifier
		8	reverse image
		10/16	hexadecimal
		20/32	low intensity
		40/64	blink
		80/128	strike-thru (parity error)

Table 59-2 (continued)

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  $rcvd\_char\_rd$  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 [S] (sync), [G] (good BCC), [D] (bad BCC), [D] (abort), and [E] (flag). See Table 59-2.

Notice on Table 59-2 that the value for good BCC and bad BCC is the same. Use *fevar\_gd\_bcc\_rd* and *fevar\_bd\_bcc\_rd* event variables to distinguish between good and bad BCC's (or data BCC's in DDCMP). Likewise, use *fevar\_gd\_bcc2\_rd* and *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 *rcvd\_char\_rd* and *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 *fevar\_abort\_rd* or *fevar\_abort\_td* to detect the interrupt. When this event variable transitions to true, it updates *rcvd\_char\_rd* (or *rcvd\_char\_td*) to indicate an abort.

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

CONDITIONS:

{

fevar\_rcvd\_char\_rd && (!(rcvd\_char\_rd & 0x100))

2. 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  $rd\_modifier$ . Table 59-2 shows how the various attributes are coded. The initial value of  $rd\_modifier$  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  $td\_modifier$ , therefore, is one.

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

When an RD data character is written to the character buffer, the value of  $rd\_modifier$  is written to the high byte of a two-byte data event-word. The data character, found in  $rcvd\_char\_rd$ , 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 0x4040. To turn blink off, it is 0x4000.

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 "FF((XXXXXXX0)) [XX]" ACTIONS: { ctl\_enhance\_td(0x4040); ctl\_enhance\_td(0x0808); }

```
CONDITIONS: DTE STRING "FF ((XXXXXXX0)) XXX "
ACTIONS:
{
ctl_enhance_td(0x4000);
ctl_enhance_td(0x0800);
}
```

# ctl\_enhance\_rd

## <u>Synopsis</u>

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 ENHANCE DCE action on the Protocol Spreadsheet.

#### **Inputs**

See ctl\_enhance\_td.

#### <u>Example</u>

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 "FF((XXXXXX0))区区"
ACTIONS:
{
    ctl_enhance_rd(0x4848);
}
CONDITIONS: DCE STRING "FF((XXXXXXX0))区区区"
ACTIONS:
    {
    ctl_enhance_rd(0x4800);
}
```

# ctl\_capture\_td

## <u>Synopsis</u>

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 mere 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 "FF((XXXXXX0)) KX 's X " ACTIONS: { ctl\_capture\_td(0x10); } CONDITIONS: KEYBOARD " " ACTIONS: { ctl\_capture\_td(0x00); }

# ctl\_capture\_rd

#### <u>Synopsis</u>

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 meter 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.

#### INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

```
LAYER: 1

STATE: find_cause

CONDITIONS: DCE STRING "FF((XXXXXXX0))) × *3×"

ACTIONS:

{

ctl_capture_rd(0x100);

}

CONDITIONS: KEYBOARD " "

ACTIONS:

{

ctl_capture_rd(0x00);

}
```

## 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.

## <u>Inputs</u>

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.

LAYER: 1

```
STATE: go_out_of_sync
CONDITIONS: DTE STRING "% "
ACTIONS:
{
outsync_action(0);
}
```

# (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."

# I1\_transmit

#### **Synopsis**

extern void l1\_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:

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

```
LAYER: 1
STATE: send_message
CONDITIONS: KEYBOARD " "
ACTIONS:
{
11_transmit(2, &send_string[0], 1);
}
```

# l1\_il\_transmit

#### <u>Synopsis</u>

extern void l1\_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.

<u>Inputs</u>

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 [] = "01\x0001004\x000((FOX))";
}
```

LAYER: 1 STATE: send\_message CONDITIONS: KEYBOARD " " ACTIONS: { \_get\_il\_msg\_buff(&il\_buffer\_number, &relay\_baton); \_start\_il\_buff\_list(il\_buffer\_number, &data\_start\_offset); \_insert\_il\_buff\_list\_cnt(il\_buffer\_number, data\_start\_offset, &message[0], (sizeof(message) - 1)); ll\_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 (PROTOCL) 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,  $F_F$ ,  $v_0$ , +, 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 *and*ed with the current TCR register to turn the current values of bits 3 and 4 (counting 1-8 from the right) to zero. This mask is always 0xf3.

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

#### Example

ł

}

This program will generate a 250-millisecond break when the operator presses the **w** 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.

Туре	Mask (hex)	Input (hex)	Meaning
data	· · · · ·		data-event word:
	0100	0100	the low byte contains data
	0500	add 0100 to the following:	td/rd indicator:
		0000 0400	td character rd character (underlined)
	ff00	add modified value of td/rd indicator to one (or a combi– nation) of the following:	<u>enhancements</u> : † (enhancements apply to data indicated in low byte)
		0000 0200 0800 1000 2000	normal alternate code set reverse image hexadecimal low intensity
		4000 8000	blink strike-thru (parity error on character
special receive			special receive-event word:
	8300	0200 8200	special receive-event word reserved
•	8700	add 0200 to the following:	td/rd indicator:
		0000 0400	td character rd character (underlined)
	bf00	add modified value of td/rd indicator to one of the following:	<u>special event</u> : (symbols for these events overlay the data indicated in low byte)
		0800 1000 1800 2000 2800 3000 3800	good CRC bad CRC abort flag sync bad CRC2 (DDCMP) good CRC2 (DDCMP)
reserved	0700	0400	reserved
reserved	0f00	0800	reserved

## Table 59-3 Character Buffer 16-Bit Word

INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

† 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 "

if(rcvd\_char\_td == 0xc2)

}

add\_event\_to\_buff (((short)td\_modifier << 8) + rcvd\_char\_td); ctl\_capture\_td(0x00);

) CONDITIONS: DTE STRING "EF" ACTIONS: CAPTURE DTE OFF

```
CONDITIONS: DCE STRING "EFF"

ACTIONS:

{

if(rcvd_char_rd == 0xc2)

{

add_event_to_buff (((short)rd_modifier << 8) + rcvd_char_rd);

ctl_capture_rd(0x00);

}

CONDITIONS: DCE STRING "EFF"

ACTIONS: CAPTURE DCE OFF
```

## add\_array\_to\_buff

#### <u>Synopsis</u>

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.

#### <u>Returns</u>

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 "FF"
        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 "X"
        ACTIONS:
        {
         dte\_array [2] = (0x0100 + rcvd\_char\_td);
         lcn = ((unsigned int)rcvd_char_td & 0x0f) << 8;</pre>
        }
        NEXT_STATE: Icn
     STATE: Icn
        CONDITIONS: DTE STRING "IN"
        ACTIONS:
        {
         dte_array [3] = (0x0100 + rcvd_char_td);
         lcn += rcvd_char_td;
         if(lcn == 0x0004)
           {
            add_array_to_buff(dte_array, 4);
            ctl_capture_td(0x00);
            current_state = state_eof;
           }
         else
            current_state = state_address;
         break;
        }
     STATE: eof
        CONDITIONS: DTE STRING "
        ACTIONS: CAPTURE DTE OFF
        NEXT_STATE: address
```

INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108
# **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:** in 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.

Туре	Variable	Value (hex/de	ecimal) Meaning
extern fast_event	fevar_eia_changed		True when the status changes for an EIA lead (non-data). Line Setup configured for emulate or monitor mode.
			<u>RS-232/V.35</u> : (RS-449)
extern const volatile unsigned short	current_eia_leads	4 8 10/16 20/32 40/64 80/128 200/512	RI (IC) DSR (DM) DTR (TR) CD (RR) CTS (CS) RTS (RS) UA
			A value in this list, when anded (&) with current_eia_leads, equals zero if the lead is on. Example: STATE: rts_on { if ((current_eia_leads & 0x80) == 0) sound_alarm(); }
			<i>Note:</i> This variable will store EIA status if (1) internal or external clock is supplied and (2) EIA leads are enabled on FEB Setup. Line Setup configured for emulate or monitor mode.
extern const volatile unsigned short	previous_ela_leads		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.

Table 60-1 EIA Variables

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:

```
{
  (fevar_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

<u>Synopsis</u>

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 0xff 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 DOE**. 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

Туре	Variable	Value	(hex/decim	nal)	Meaning
extern fast_event	display_screen_changed			lisplay-se when Run changed. crnt_disp prev_disp crnt_disp Line Setu	n Run-mode creen is changed, or //Freeze mode is Value in /ay_screen is stored in /ay_screen, and /ay-screen is updated. p configured for or monitor mode.
extern unsigned short	crnt_display_screen		(	low byte Init is in	current display screen ) and indicates whethe Run mode or Freeze
			c		gh byte). Line Setup d for emulate or node.
			c	display-so	creen
		0 1 2	5	no display single-line dual-line	e data data
		3 4 11/:	17 1	dual-line abular st	
		12/3 21/3 31/4 41/9	33 i 19 i	graphic s Display W Program	/indow
		42/0 43/0 44/0	56 I 57 I	_ayer 2 F _ayer 3 F	Protocol Trace Protocol Trace Protocol Trace
		45/0 46/7 47/7	70 I 71 I	_ayer 6 F _ayer 7 F	Protocol Trace Protocol Trace Protocol Trace
		51/8 52/8 53/8 54/8	32 I 33 I	Jser Tra Jser Tra Jser Tra Jser Tra	ce 2 ce 3
		55/0 55/0 56/0 57/0	35 I 36 I	User Tra User Tra User Tra	ce 5 ce 6
		61/9 62/9	97	TIM pack	age standard stats
					eze mode (bit 9)
		100 0		Freeze n Run mod	

Туре	Variable	Value (hex/de	cimal) Meaning
extern unsigned short	prev_display_screen	a da servica da servic A servica da servica da A servica da	Contains previous display screer (low byte) and indicates whether unit was in Run mode or Freeze mode (high byte). Line Setup configured for emulate or monitor mode.
			display-screen
		0 1 2 3 4	no display single-line data dual-line data single-line data with leads 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 User Trace 2
		52/82 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
		ULIUU	Run/Freeze mode (bit 9)
			numi reeze mode (bit 9)
		100/256	Freeze mode
		0	Run mode

## Table 61-1 (continued)

## 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  $\textcircled{\ }$  and  $\textcircled{\ }$  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*.

Туре	Variable	Value (hex/de	ecimal) Meaning
extern unsigned short	current_line	0-10/0-16	Contains the current row number of the cursor position i the Display Window. Line Setu configured for emulate or monitor mode.
extern unsigned short	current_col	0-3110-63	Contains the current column number of the cursor position i the Display Window. Line Setu configured for emulate or monitor mode.
extern unsigned short	window_color		<ul> <li>Two-byte variable. Current color selections are indicated i the low byte. Current enhancements are indicated in the high byte. Written to by % conversions. Attributes are copied into data words in Display Window. Line Setup configured for emulate or monitor mode.</li> <li>Isolate bits of interest via bitwise anding (&amp;) of mask wit variable. Compare result to value column. For example, underline attribute mask = 0x100. Therefore window_colo &amp; 0x100 (underline on). Line Setup configured for emulate of monitor mode.</li> </ul>
			monitor mode. background color mask = 7 (b 1-3):
		0 1 2 3 4 5	black blue green cyan red magenta
		6 7	yellow white foreground color mask = 0x38 (bits 4–6):
		0 8 10/16 18/24 20/32 28/40 30/48 38/56	black blue green cyan red magenta yellow white

Table 61-2 Display Window Variables

Туре	Variable	Value (hex/de	cimal)	Meaning
	(window_color continued	0	color b	link mask = 0x40 (bit 7):
		0 40/64	no blink blink	
			color st (bit 8):	trike-thru mask = 0x80
		0 80/128	no strik strike-t	
			overline	e mask = 0x100 (blt 9):
		0 100/256	no over overline	
			blank n	nask = 0x200 (bit 10):
		0 200/512	no biani biank	k
			underlii 11):	ne mask = 0x400 (bit
		0 400/1024	no unde underlir	
			reverse 12):	e image mask = 0x800 (b
		0 800/2048		erse image e image
			hex ma	ask = 0x1000 (bit 13):
		0 1000/4096	no hex hex	
			low inte 14):	ensity mask = 0x2000 (bi
		0 2000/8192	no low low inte	intensity ensity (RS-170 output)
				hrome blink mask = ) (bit 15):
		0 4000/16384		nochrome blink hrome blink
				hrome strike-thru mask (bit 16):
		0 8000/32768	no mo monoc	nochrome strike-thru hrome strike-thru

## Table 61-2 (continued)

.

Variable	Value (hex/d	ecimal) Meaning
window_modifier		Contains the current modifiers. Line Setup configured for emulate or monitor mode.
		font mask = 7 (bits 1-3):
	0	ASCII
	1	special graphic character set (refer to Table 61-5)
	2	primary font-code selected on Line Setup
	3	alternate font—current implementation is for call-setup phase in X.21 (ASCII)
	7	hexadecimal
display_window_buffe	r [1088]	Array of 32-bit words that mak 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.
	window_modifier	window_modifier 0 1 2 3

Table 61-2 (continued)

## (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.

Display Window Buffer Structures					
Туре	Variable	Value (hex/decimal)	Meaning		
	n an				
Structure Name:	display_window	v_index_buffer [17]	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 <i>displayf</i> routine changes line 3, <i>display_window_index_buffer[3].mpm</i> is automatically incremented. The CPM detects the difference between		
			display_window_index_buffer [3].mpm and display_window_index_buffer [3].cpm and updates line 3 in the Display Window. Declared as type extern struct.		
			You must increment an <i>mpm</i> variable manually when you write <i>directly</i> (not via a <i>displayf</i> routine) to the Display Window.		
unsigned char	mpm	0-ff/0-255	When the MPM updates a line in the display-window buffer, this variable is incremented.		
unsigned char	cpm	0-ff/0-255	The CPM checks the value of this variable against the value of <i>mpm</i> . If they are different, the value in <i>mpm</i> is copied into <i>cpm</i> . The updated line in MPM is then presented on the		
			display-window screen.		

Table 61-3 Display Window Buffer Structures

## (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, *displayf*, 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 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');	а
displayc(65);	Α
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 ("%#u", 258); yields °1°2. For c and s conversions, if the argument contains a newline character, it is displayed as '\F.

- 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 1 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 1 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

С

s

p

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.

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

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.)

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 +. 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 *stracef* routine, you must use %b rather than %s to display it. (To display the information in an array written to via *sprintf*, use %s.)

m

The argument is a *long* integer that indicates attributes to be assigned to subsequent characters. Attributes stay "on" until they are specifically turned "off" with another %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 *window\_color* and *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 *long*. Append an "L" at the end of the hexadecimal code specifying attributes to indicate the value is a *long*.

**NOTE:** If you are specifying an attribute in a lower-order byte of the *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, 0x200000L is converted to 0x00200000L. 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 %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.

## <u>Returns</u>

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).

Bit	Mask (hex)†	Input (hex)††	Meaning
allokara (k. K		an an airte an	<i>Modifier</i> attributes, font for example, are contained in the low byte of the 32-bit word.
1-3	000000 <u>07</u> L		Font:
		000000 <u>00</u> L 000000 <u>01</u> L 00000002L	ASCII special graphic character set (refer to Table 61-5) primary font—code selected on Line
		000000 <u>03</u> L 000000 <u>07</u> L	Setup alternate font—current implementation is for call-setup phase in X.21 (ASCII) hexadecimal
4	000000 <u>08</u> L	The second s	<u>Special character indicator:</u> (used in trace buffer only; should not be altered by user)
	ett strangen som en store en som e	000000 <u>00</u> L 000000 <u>08</u> L	only value in <i>modifier</i> in trace buffer header Character is not displayable but contains control info used internally by the trace logic. When a "\n" is
			included in a <i>tracef</i> routine, for example, a new line is generated, but nothing is displayed on the trace screen. The <i>tracef</i> routine automatically sets this bit before the 32-bit word is written into
5-8	000000f0L	0000000L	<i>trace_buf.array</i> . unused, but should be zero
9-16	00000 <u>ff</u> 00L	00000 <u>00</u> 00L	<i>Character data</i> is contained in the second byte of the <i>long</i> word. Input should be 00 in all % <i>m</i> conversions.

Table 61-4 Display Window/Trace Buffer 32-Bit Data Word

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:

display window buffer[position] = ((display\_window\_buffer[position] & (-attribute-mask)) | 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:

l2\_trbuf.array[20] = ((trace\_buf.array[20] & (~0x00000007L)) | 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.

Bit	Mask (hex)	Input (hex)	Meaning
			<i>Color</i> is contained in the third byte of the <i>long</i> . Combine color attributes via hexadecimal addition.
17-19	00 <u>07</u> 0000L		Background color:
		00 <u>00</u> 0000L 00 <u>0</u> 10000L 00 <u>0</u> 20000L 00 <u>03</u> 0000L 00 <u>04</u> 0000L 00 <u>05</u> 0000L 00 <u>05</u> 0000L 00 <u>07</u> 0000L	black blue green cyan red magenta yellow white
20-22	00 <u>38</u> 0000L		Foreground color:
		00 <u>00</u> 0000L 00 <u>08</u> 0000L 00 <u>10</u> 0000L 00 <u>18</u> 0000L 00 <u>20</u> 0000L 00 <u>28</u> 0000L 00 <u>38</u> 0000L 00 <u>38</u> 0000L	black blue green cyan red magenta yellow white
23	00 <u>40</u> 0000L		Color blink:
		00 <u>00</u> 0000L 00 <u>40</u> 0000L	no blink blink
24	00 <u>80</u> 0000L		Color strike-thru:
		00 <u>00</u> 0000L 00 <u>80</u> 0000L	no strike-thru strike-thru <i>Enhance</i> attributes, underlining for example, are contained in the high
			byte of the <i>long</i> . Combine enhancements via hexadecimal addition.
25	<u>01</u> 000000L		<u>Overline</u> : (for monochrome and color)
		<u>00</u> 000000L <u>01</u> 000000L	no overline overline
26	0200000L		Blank:
		0000000L 02000000L	monochrome display, color display monochrome no display, color display
27	<u>04</u> 000000L		<u>Underline</u> : (for monochrome and color)
		<u>00</u> 000000L <u>04</u> 000000L	no underline underline

## Table 61-4 (continued)

Bit	Mask (hex)	Input (hex)	Meaning
<b>28</b> . a get er artikulter	<u>08</u> 000000L		Monochrome reverse image:
		<u>00</u> 000000L <u>08</u> 000000L	no reverse image reverse image
29	<u>10</u> 00000L		Hex:
		<u>00</u> 000000L <u>10</u> 000000L	no hex hex
30	2000000L		Monochrome low intensity:
		0000000L 20000000L	no low intensity low intensity (RS-170 interface)
31	<u>40</u> 000000L		Monochrome blink:
		<u>00</u> 000000L <u>40</u> 000000L	no blink blink
32	8000000L		Monochrome strike-thru:
		<u>00</u> 000000L <u>80</u> 000000L	no strike-thru strike-thru

## Table 61-4 (continued)

-

Real and

(1) Contract A sector Contracts and prime to the sector constrained of the sector sector (sector) = (Contracts) = (Contracts) and (Contract) = (Contracts) = (Contracts)

> , 1918) Akonan ana ana kaona ana kaona

n ortek timar o skalatak - zakar olihikuy mavedita oleh na disbilik o shilaa okiga a

> ak oligi. Nanjme sektorova a sektorova

> > an a subtra 15. An ang Pangan

> > > 61–16

.

Display	Input (hex/decimal)	Display	Input (hex/decimal)
	0	٦	1a/26
L	1	1	1b/27
	2		1c/28
	3		1d/29
»	4		1e/30
«	5	<b>L</b>	1f/31
	6		20/32
5	7	F	21/33
$\mathbf{\Sigma}$	8	*	22/34
F	9		23/35
A	a/10		24/36
G	b/11		25/37
B	c/12		26/38
	d,11/13,17		27/39
2	e/14	8	28/40
<ul> <li>         ・・・         ・・・         ・・・</li></ul>	f/15		29/41
	10/16		2a/42
1	12/18	~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~	2b/43
ſ	13/19		2c/44
가는 수요. 같은 관람을 통해하는 것이다.	14/20		2d/45
	15/21		2e/46
B	16/22	***	2f/47
	17/23		30/48
in a fille and the second s	18/24	(space)	31/49
	19/25		

 Table 61-5

 Special Graphic Character Set†

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

Display Input (hex/decimal) Display Input (hex/decimal)						
ologi (1997) New Alexandra Yorand Alexandra (1997)	80/128	n la prime come al se construction de la compaction de la compaction de la compaction de la compaction de la co La compaction de la compaction de la compaction de la compaction	9a/154			
第68.5 m 1 ■	81/129	<del>サ</del>	9b/155			
r <sup>tada</sup> at	82/130	Ð	9c/156			
	83/131	ス	9d/157			
in the state of the state of t	84/132	セ	9e/158			
1999 - 1999 	85/133	Y	9f/159			
<b>Э</b>	86/134	5	a0/160			
T and the	87/135	÷	a1/161			
<b>-</b>	88/136	<u>ب</u>	a2/162			
<b>?</b>	89/137	Ŧ	a3/163			
r <sup>alla</sup> l	8a/138	د. الاستقلال	a4/164			
<b>オ</b>	8b/139	7	a5/165			
ייין איז	8c/140	=	a6/166			
_	8d/141	R	a7/167			
	8e/142	*	a8/168			
ייין איז	8f/143	J	a9/169			
<b>—</b> <sub>19</sub> 19 - 19	90/144	Ji and	aa/170			
<b>ም</b> የአለት የሚ	91/145		ab/171			
$\mathbf{r}^{\mathrm{part}}$	92/146	つの作品	ac/172			
<b>ウ</b> (1) (1)	93/147		ad/173			
I	94/148	<b></b>	ae/174			
* *	95/149	੨	af/175			
カ	96/150		b0/176			
+	97/151	4	b1/177			
л	98/152	*	b2/178			
<b>ታ</b> <sup>1056</sup> በ የትንድ የ	99/153	ene el frèsiq <b>e</b> r estres	e 11 e protecti ( <b>b3/179</b> e protecti			

Table 61-5 (continued)

61–18

Display	Input (hex/decimal)	Display	Input (hex/decimal)
4	b4/180	Ä	ce/206
2	b5/181	A	cf/207
	b6/182	É	d0/208
ラ	b7/183	æ	d1/209
n an the 1999 - Bryan Daniel an an Antonia an Antonia 1997 - Antonia Daniel an Antonia an Antonia	b8/184	£	d2/210
JU	b9/185	Ô	d3/211
L	ba/186	ö	d4/212
	bb/187	ò	d5/213
ס	bc/188	û	d6/214
	bd/189	ù	d7/215
	be/190	ÿ	d8/216
	bf/191	Ö	d9/217
Ç	c0/192	Ü	da/218
ü	c1/193	¢	db/219
é	c2/194	`£	dc/220
â	c3/195	ß	dd/221
ä in teoremisen ander der der der der der der der der der	c4/196	Ŗ	de/222
à	c5/197	ſ	df/223
a	c6/198	á	e0/224
Ç	c7/199	í	e1/225
ê	c8/200	ó	e2/226
ë	c9/201	ú	e3/227
è	ca/202	ñ	e4/228
ï	cb/203	Ñ	e5/229
î	cc/204	<b>a</b>	e6/230
ì	cd/205		e7/231

Table 61-5 (continued)

61-19

Display Input (hex/decimal)		Display Input (hex/decimal)		
ذ	e8/232	ing a state of the	ed/237	
-	e9/233	••	ee/238	
<b>-</b> <sup>1.20</sup> .00	ea/234	S	ef/239	
1/2	eb/235		f0/240	
1/4	ec/236			

Table 61-5 (continued)

## displays

#### <u>Synopsis</u>

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*.

#### <u>Inputs</u>

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.

### <u>Example</u>

The following entry

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:

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  $\$ . 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.

### <u>Returns</u>

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.

#### <u>Example</u>

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. { unsigned 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

 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 11\_trbuf;

The names of all the trace buffers are listed in Table 61-6.

#### Variable Value (hex/decimal) Meaning Type Structure Name: trace\_buffer\_header 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/sys/include/trace\_buf.h. Written to by %m conversion specifier. Because it is an extern structure, values of component variables should not be altered directly by the user. In some instances, e.g., altering array size, the result could be a crash. 0-fff10-4095 end of data within the buffer. Maximum value is unsigned short logical\_end one less than the array size. 0 trace buffer is not full logical\_end\_wrap\_count unsigned short trace buffer is full. As new lines are written to non-zero the end of the trace buffer, lines at the beginning of the buffer are removed. Special-character indicator bit and bit 8 must be modifier unsigned char zero. For other specific values and their meanings, see Table 61-4. 0-ff/0-255 For specific values and their meanings, see unsigned char color Table 61-4. For specific values and their meanings, see unsigned char enhance 0-ff/0-255 Table 61-4. prevents two processes from writing to the same unsigned short write lock 0-fffff/0-65535 buffer at the same time. Should not be altered by user or future access to the trace buffers may be locked out. size of buffer; at present only one value array\_size 1000/4096 unsigned short number of characters in last line in buffer 0-3f/0-63 unsigned char line size 0 reserved for future use unsigned char spare Structure of a trace buffer. Declared as type Structure Name: trace\_buf extern struct. Declared automatically if a softkey-entered TRACE action is taken. Declaration contained in file named HRD/sys/include/trace\_buf.h.

## Table 61-6 **Trace Buffer Structures**

structure of the trace-buffer header described above

array of data words in the buffer

struct trace\_buffer\_header

unsigned long

hdr

array [4096]

Туре	Variable	Value (hex/decimal)	Meaning
Structure Name:	prog_trbuf	1997 - 2013) 가슴여다: 2011 가 다음 - 644 다음 - 644 다음 - 1995 - 1997 다음 - 1997 - 1997 다음 - 1997 - 1997	Structure of the Program Trace buffer, an instance of the <i>trace_buf</i> structure declared in file named <i>HRDIsyslincludeItrace_buf.h</i> . Declared as type <i>extern struct trace_buf</i> . Declared automatically if a softkey-entered TRACE action is taken. Writing to this buffer makes the Run-mode PROG TR softkey appear.
struct trace_buffer_he	eader hdr		structure of the trace-buffer header described above
unsigned long ar	ray [4096]		array of data words in the buffer
<u>Structure Name:</u>	l1_trbuf		Structure of one of seven user trace buffers, an instance of the <i>trace_buf</i> structure declared in file named <i>HRD/sys/include/trace_buf.h</i> . Declared as type extern struct trace_buf. Writing to this buffer causes the Run-mode TRACE 1 softkey appear.
struct trace_buffer_h	eader hdr		structure of the trace-buffer header described above
unsigned long a	rray [4096]		array of data words in the buffer
<u>Structure Name:</u>	l2_trbuf	사망(사망) - 프랑아(아이) - 라양(아이) - 나아(아이) - 나아(아이) - 아이아(아이) - 아이아(아이) - 아이아(아이) - 아이아(아이아) - 아이어(아이아) - 아이아(아이아) - 아이어(아이아) - 아이어(아이어) - 아이어(아이어(아이어) - 아이어(아이어(아이어) - 아이어(아이어(아이어) - 아이어(아이어(아이어) - 아이어(아이어(아이어) - 아이어(아이어(아이어) - 아이어(아이어(아이어(아이어) - 아이어(아이어(아이어) - 아이어(아이어(아이어(아이어(아이어) - 아이어(아이어(아이어(아이어(아이어(아이어(아이어(아이어(아이어(아이어(	Structure of one of seven user trace buffers, an instance of the <i>trace_buf</i> structure declared in file named <i>HRD/sys/include/trace_buf.h</i> . Declared as type extern struct trace_buf. Writing to this buffer causes the Run-mode TRACE 2 softkey appear.
struct trace_buffer_h	eader hdr		structure of the trace-buffer header described above
unsigned long a	rray [4096]		array of data words in the buffer
<u>Structure Name:</u>	l3_trbuf		Structure of one of seven user trace buffers, an instance of the <i>trace_buf</i> structure declared in file named <i>HRD/sys/include/trace_buf.h</i> . Declared as type extern struct trace_buf. Writing to this buffer causes the Run-mode TRACE 3 softkey appear.
struct trace_buffer_h	eader hdr		structure of the trace-buffer header described above
unsigned long a	rray [4096]		array of data words in the buffer
Structure Name:		nan an 1945 - Angelan 1965 - Angelan Angelan 1965 - Angelan	Structure of one of seven user trace buffers, ar 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.
struct trace_buffer_h	neader hdr		structure of the trace-buffer header described above
unsigned long a	urray [4096]		array of data words in the buffer

61-25

Туре	Variable	Value	(hex/decimal)	1997). 	Meaning	CCC 
<u>Structure Name</u>	ne ful roet er fin	을 많 이 2003 이 1000		instance of file named Declared Writing to	of one of seven user to of the trace_buf struct d HRD/sys/include/trac as type extern struct to this buffer causes the softkey appear.	ure declared in e_buf.h. trace_buf.
struct trace_buffe	r_header hdr			structure above	of the trace-buffer he	ader described
unsigned long	array [4096]			array of d	lata words in the buffe	r <sub>gani bengimu</sub>
Structure Name	स्टब्स् क्रिक्ट स्टिक्स स्टब्स् इत्या अन्द्रियद्वार दिवेल अक्षेत्र चार्वप्रस्तु स्टिक्स्			instance of file named Declared Writing to	of one of seven user to of the trace_buf struct d HRD/sys/include/trac as type extern struct to this buffer causes the softkey appear.	ure declared in ce_buf.h. trace_buf.
struct trace_buffe	r_header hdr			structure above	of the trace-buffer he	ader described
unsigned long	array [4096]	ang barb		array of c	data words in the buffe	r ord tenpleta
Structure Nam	ningag karakatan Seri Kolon (Ko Kolang Kong Kong Kolon (Kong K geng karakatan (Kong Kolon (Kong Kang Kong Karakatan (Kong Kong Kong Kong Kong Kong Kong Kong			instance of file named Declared Writing to	of one of seven user of the <i>trace_buf</i> struct d <i>HRD/sys/include/trac</i> as type <i>extern struct</i> i o this buffer causes the softkey appear.	ure declared in ce_buf.h. trace_buf.
struct trace_buffe		n di yana shari Birki birki bir		structure above	of the trace-buffer he	ader described
unsigned long	array [4096]			array of o	data words in the buffe	er gandelanssetter u

#### Table 61-6 (continued)

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 SIZE. . . . 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, "\nThis message will not generate a leading blank line.");

#### tracec

#### **Synopsis**

extern void tracec(trace\_buffer\_ptr, character); extern struct trace\_buf \* trace\_buffer\_ptr; const 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

ł

}

In this instance, output will be written to the Program Trace screen.

#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,65); tracec(&prog\_trbuf, '\n'); tracec(&prog\_trbuf,0x65);

> tracec(&prog trbuf, '\n'); tracec(&prog\_trbuf,065);

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

а 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 trace\_buf.hdr.modifier, trace\_buf.hdr.color, and 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.

#### <u>Returns</u>

The *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.

#### LAYER: 3

}

```
#include <trace buf.h>
extern struct trace buf 13 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:
   ł
      pad_ctrl_msg = m_packet_info_ptr[0];
tracef (&l3_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 (&l3_trbuf, "%mDCE%m LCN:%.3x PAD MSG:%.2x\n", 0x04000000L,
          0x00000000L, m_packet lcn, pad ctrl msg);
   3
```

#### stracef

#### **Synopsis**

extern void stracef(array\_ptr, string\_ptr);
unsigned long array\_ptr;
const char \* string\_ptr;

#### Description

The stracef routine is similar to the tracef routine, except that stracef writes output to a variable, while tracef writes output to a trace screen. The output is under control of the string pointed to by 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 stracef routine returns when the end of the format string is encountered. The stracef routine differs from sprintf in that it generates an array of longs, whereas sprintf generates an array of chars. When the stracef array is written to a trace buffer (or to the Display Window) it carries its predefined attributes along with it. An 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 *modifier* attribute-byte.

#### 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 *long*. By allocating 32 bits for each element, the array may accommodate attributes assigned via the %*m* conversion specifier. Attributes comprise 24 bits of the *long*. The preferred form of the declaration is *unsigned long name [100]*. The size and name of the array are user-determined.

For the second parameter, see the *displayf* routine.

### <u>Example</u>

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 *tracef*. Note that attributes that are turned on in an *stracef* array do not need to be turned off after the array has been brought, via the %b conversion specifier, into a *tracef* format string.

LAYER: 3

```
#include <trace_buf.h>
   extern struct trace buf 13 trbuf;
   extern unsigned char * m_packet_info_ptr;
  extern unsigned short m_packet_lcn;
  unsigned char pad_ctrl_msg;
   unsigned long source[4];
}
   STATE: packet_type
     CONDITIONS: DTE DATA Q= 1
     ACTIONS:
        stracef (source, "%s", "DTE");
     NEXT_STATE: pad_msg_trace
     CONDITIONS: DCE DATA Q= 1
     ACTIONS:
        stracef (source, "%m%s", 0x04000000L, "DCE");
      NEXT_STATE: pad_msg_trace
   STATE: pad msg_trace
      CONDITIONS: ENTER STATE
      ACTIONS:
      ł
         pad_ctrl_msg = m_packet_info_ptr[0];
tracef (&l3_trbuf, "%b LCN:%.3x PAD MSG:%.2x\n", source, m_packet_lcn,
           pad_ctrl_msg);
      NEXT_STATE: packet_type
```

### 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

#include <trace buf.h> extern struct trace\_buf l1\_trbuf;

```
LAYER: 1
```

ł

}

STATE: any CONDITIONS: KEYBOARD " " ACTIONS: { traces(&l1\_trbuf, "End of test."); }

produces the following output on the TRACE 1 trace screen:

End of test.

The following coding produces the same output:

}

#include <trace\_buf.h> extern struct trace\_buf l1\_trbuf; LAYER: 1 STATE: any CONDITIONS: KEYBOARD " " ACTIONS: { const char \* string ptr; string\_ptr = "End of test."; traces (&l1\_trbuf, string\_ptr); }

## 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);

territ e se se adh suabour galleo y avenisi osili -

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 *displayf* 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.





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 l1_trbuf;
}
STATE: apply_attribute_to_header
CONDITIONS: ENTER_STATE
ACTIONS:
{
    tracef (&l1_trbuf, "%mThis data is underlined.", 0x0400000L);
}
```

Only the header for the TRACE 1 display is affected by this %m 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.

```
CONDITIONS: KEYBOARD " "
ACTIONS:
{
display_window_buffer[64] = ((display_window_buffer[64] & -0x0400000L) |
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.

Spec	ifier	Argument type	Conversion Type	
%	6 <b>b</b>	integer-array pointer	array of <i>long</i> integers. 2nd byte of each <i>long</i> 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.	
%	61	integer	signed decimal representing 15-bit value	
%	6C	unsigned character	unsigned character	
%	6#c	unsigned character	newline character displayed as $\succ$ rather than acted on	
%	6d	integer	signed decimal representing 15-bit value	
9	6ld	integer	signed decimal representing 31-bit value	
9	6H (1997) - 1997	character-array pointer	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.)	
9	6m	integer	long integer not displayed or printed, but written to attribute header-variable for Displa Window or for one of the trace buffers	
9	60	integer	unsigned octal representing 16-bit value	
9	610	integer	unsigned octal representing 32-bit value	
9	6#o	integer	unsigned octal representing 16-bit value, preceded by 0	
9	6#lo	integer	unsigned octal representing 32-bit value, preceded by 0	
9	6p	integer	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 a offset.	
9	%S	character-array pointer	array of characters beginning at pointer and ending at null terminator or at array-length precision, whichever occurs first	
9	%#s	character-array pointer	newline character displayed as '+ rather tha acted on	
ç	%u	integer	unsigned decimal representing 16-bit value	
c	%lu	integer	unsigned decimal representing 32-bit value	
ç	%#u	integer	hex characters (example: <sup>B</sup> F <sup>E</sup> 5 ) representi 16-bit value	
	%#lu	Integer	hex characters (example: <sup>B</sup> F <sup>E</sup> 5 <sup>3</sup> 0 <sup>1</sup> 3)	

## Table 61-7 **Conversion Specifiers**

Specifier	Argum	nent type	Conversion Type
%x	integer	с	unsigned hexadecimal (lower-case letters) representing 16-bit value
%lx	integer		unsigned hexadecimal (lower-case letters) representing 32-bit value
%#x	integer		unsigned hexadecimal (lower-case letters) representing 16-bit value, preceded by 0x
%#Ix	integer		unsigned hexadecimal (lower-case letters) representing 32-bit value, preceded by 0x
%X	integer		unsigned hexadecimal (upper-case letters representing 16-bit value
%IX	integer		unsigned hexadecimal (upper-case letters representing 32-bit value
%#X	integer		unsigned hexadecimal (upper-case letters representing 16-bit value, preceded by 0x
%#IX	integer		unsigned hexadecimal (upper-case letters representing 32-bit value, preceded by 0x
%\n	none		displays an ۶
%%	none		displays a %

Table 61-7 (continued)

# 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.

Туре	Variable Value (he:	x/decimal) Meaning
extern unsigned char	l2pp_trbuff	First data byte in the Layer 2 Protocol Trace buffer. Addres of this variable—segment number plus offset—will indicate the <i>physical</i> location of the first data byte, two bytes from the beginning of the buffer. Line Setup configured for emulate o monitor mode.
extern unsigned long	l2pp_trbuff_end	First byte in the two-byte traile 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 <i>physica</i> 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.
extern unsigned char	l3pp_trbuff	First data byte in the Layer 3 Protocol Trace buffer. Address of this variable—segment number plus offset—will indicat the <i>physical</i> location of the firs data byte, two bytes from the beginning of the buffer. Line Setup configured for emulate of monitor mode.
extern unsigned long	l3pp_trbuff_end	First byte in the two-byte traile 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 <i>physica</i> 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.

Table 61-8Protocol Trace Buffer Variables

.

#### (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 13\_trbuff to identify the physical address of a logical location.

The example below uses *#define* preprocessor directives for determining beginning and ending offsets in the Layer 3 Protocol Trace buffer. When *get\_l3pp\_value\_end* is encountered in a program, for example, each of the two "end" offset-variables is cast into a *long* and, if necessary, shifted left to its appropriate position in an offset. Then the two variables are added together.

#define get\_l3pp\_value\_begin
(((unsigned long) (l3pp\_trbuff\_ctl.begin\_off\_hi) << 8) +
((unsigned long) (l3pp\_trbuff\_ctl.begin \_off\_lo)))</pre>

#define get\_l3pp\_value\_end
(((unsigned long)(l3pp\_trbuff\_ctl.end\_off\_hi) << 8) +
((unsigned long)(l3pp\_trbuff\_ctl.end\_off\_lo)))</pre>

When the ending offset, in this example, is added to the address of *l3\_trbuff*, the result is the address of the *logical* end in the buffer:

unsigned long end\_address; end\_address = &l3\_trbuff + get\_l3pp\_value\_end;

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.

#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\_off\_hi) << 8) + ((unsigned long) (l3pp\_trbuff\_ctl.begin\_off\_lo)))

#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\_off\_hi) << 8) + ((unsigned long)(l3pp\_trbuff\_ctl.end\_off\_lo)))

unsigned long end, begin, count; end = get\_13pp\_value\_end; begin = get\_13pp\_value\_begin; count = end - begin;

Туре	Variable	Value (hex/decimal)	Meaning
na su companya da su			
Structure Nam	<u>ne:</u> lpp_trbuff_ctl		Declared as type <i>struct</i> . The variables contained in this structure monitor logical location in a Protocol Trace buffer. Reference structure variables as follows: <i>lpp_trbuff_ctl.begin_off_hi</i> .
unsigned char	begin_off_hi	0-ff/0-255	High byte of a 2-byte offset from the <i>physical</i> beginning of the Protocol Trace buffer to a <i>logical</i> beginning in the buffer. Range of the two-byte offset is 2 through hexadecimal FFFE.
unsigned char	begin_off_lo	0-ff10-255	Low byte of a 2-byte offset from the <i>physical</i> beginning of the Protocol Trace buffer to a <i>logical</i> beginning in the buffer. Range of the two-byte offset is 2 through hexadecimal FFFE.
unsigned char	begin_wrap_hi	0-ff/0-255	High byte of a 2-byte count of the number of times a <i>logical</i> beginning has wrapped through the Protocol Trace buffer.
unsigned char	begin_wrap_lo	0-ff/0-255	Low byte of a 2-byte count of the number of times a <i>logical</i> 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.
unsigned char	end_off_hi	0-ff10-255	High byte of a 2-byte offset from the <i>physical</i> beginning of the Protocol Trace buffer to a <i>logical</i> end in the buffer. Range of the two-byte offset is 2 through hexadecimal FFFE.
unsigned char	end_off_lo	0-ff10-255	Low byte of a 2-byte offset from the <i>physical</i> beginning of the Protocol Trace buffer to a <i>logical</i> end in the buffer. Range of the two-byte offset is 2 through hexadecimal FFFE.
unsigned char	end_wrap_hi	0-ff10-255	High byte of a 2-byte count of the number of times a <i>logical</i> end has wrapped through the Protocol Trace buffer.
unsigned char	end_wrap_lo	0-ff10-255	Low byte of a 2-byte count of the number of times a <i>logical</i> end has wrapped through the Protocol Trace buffer. It will have a value of ze only once. Once the count reaches hexadecim FFFF, it will wrap to one.
Structure Nar	<u>ne:</u> l2pp_trbuff_ctl		An instance of the <i>lpp_trbuff_ctl</i> structure, declared as type <i>extern struct lpp_trbuff_ctl</i> . The variables contained in this structure monito logical location in the Layer 2 Protocol Trace buffer. Has the same structure as <i>lpp_trbuff_ctl</i> . Reference structure variables a follows: <i>l2pp_trbuff_ctl.begin_off_h</i> .

## Table 61-9 Protocol Trace Buffer Structures

61-41

INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

Table 61-9 (continued	Table	61-9 (	continued	1)
-----------------------	-------	--------	-----------	----

Туре	Variable	Value (hex/decimal)	Meaning
Structure Name:	i3pp_trbuff_ctl		An instance of the <i>lpp_trbuff_ctl</i> structure, declared as type <i>extern struct lpp_trbuff_ctl</i> . The variables contained in this structure monitor logical location in the Layer 3 Protocol Trace
			buffer. Has the same structure as <i>lpp_trbuff_ctl</i> . Reference structure variables as follows: <i>l3pp_trbuff_ctl.begin_off_h</i> .

# (C) Routines

There are no routines associated exclusively with Protocol Traces.

# 62 Counters, Timers, and Accumulators

## 62.1 Counters

ł

The translator declares the following structure for counters that are entered as softkey tokens on the Protocol Spreadsheet:

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,-0u1};

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*.

### INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

Туре	Variable	Meaning
Structure Name:	counter_struct	Structure of a counter. Declared as type <i>struct</i> . 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.
unsigned long	current	This value of the counter is acted on directly by program actions.
unsigned long	last	Last sampled value; displayed on the tabular statistics screen.
unsigned long	maximum	Maximum value of all samples; displayed on the tabular statistics screen.
unsigned long	minimum	Minimum value of all samples; displayed on the tabular statistics screen. Should be initialized as ~0ul.
unsigned short	sample_count	Number of samples.
unsigned long	total_high	High four bytes of an eight-byte counter total.
unsigned short	total_low_low	Low two bytes of an eight-byte counter total. This two-byte variable counts to 65,535.
unsigned short	total_low_high	Bytes 3 and 4 of an eight-byte counter total.
unsigned short	out_of_range	Number is out of range, either incremented beyond the range or decremented below 0; should not be factored into averages.
unsigned short	changed	For future use.
unsigned long	prev	When converting a counter action to C, the translator compares <i>prev</i> with <i>current</i> to determine whether counter has changed. Then <i>prev</i> is updated to <i>current</i> and <i>counter_name_change</i> is signaled.
unsigned long	old	When converting a counter condition to C, the translator compares <i>old</i> with <i>current</i> to determine whether true condition is new (transitional). After program logic has examiner counter, <i>old</i> is updated to <i>prev</i> .

## Table 62-1 Counter Structures

Here, for example, is the C translation of the simple action COUNTER example SET 5.

```
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);
}
```

Туре	Variable	Meaning
extern event	counter_ <i>name</i> _change	True when the named counter is incremented, or
		set to new value. This event will not be triggered unless a
		spreadsheet condition names the counter. Line Setup
	an a	configured for emulate or monitor mode.

Table 62-2 Counter Variables

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:

```
counter_name.last = counter_name.current;
if (counter_name.current > counter_name.maximum)
 Ł
   counter_name.maximum = counter_name.current;
 3
if (counter_name.current < counter_name.minimum)
 ł
   counter name.minimum = counter name.current;
 }
counter_name.sample_count++;
Ł
 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).

Туре	Variable	Value (hex/decimal)	Meaning
<u>Structure Nam</u>	<u>ie:</u> stat_msg		Structure of a stat message. A stat message is sent once for each named counter, timer, or accumulator. Declared as type struct. Declare 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.
unsigned short	op_type	0a00/2560	Register statistics objects from the MPM to the CPM. Other values and meanings for future use
unsigned short	type	0 0100/256 0200/512	accumulator counter timer
unsigned long	object_name		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("name_of_counter");
unsigned long	object_address		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: <i>name</i> .object_address = get_68k_phys_addr(&counter_ <i>name_of_counter</i> )

Table 62-3Counter, Timer, and Accumulator Structures

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.

```
{
  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\_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 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.object_name = get_68k_phys_addr("key");
    stat_msg.object_address = get_68k_phys_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

ł

The translator declares the following structure for timers that are entered as softkey tokens on the Protocol Spreadsheet:

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-triggering 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.

Туре	Variable	Value	(hex/decimal)	Meaning
an a fairt an				
<u>Structure Name:</u>	timer_struct			Structure of a timer. Declared as type <i>struct</i> . Declared automatically if a program timer is used. Program timers assigned to structure as follows: struct timer_struct timer_ <i>name</i> . Reference a structure variable as follows: timer_ <i>name</i> .current.
unsigned long	current			Current value of timer, not updated while timer i running. Values are in microseconds rounded t tick-unit on FEB Setup screen.
unsigned long	last			Value of last sample; displayed on the tabular statistics screen.
unsigned long	maximum			Maximum value of all samples; displayed on the tabular statistics screen.
unsigned long	minimum			Minimum value of all samples; displayed on the tabular statistics screen. Should be initialized a -0ul.
unsigned short	sample_count		•	Number of samples.
unsigned long	total_high			High four bytes of an eight-byte timer total.
unsigned short	total_low_low			Low two bytes of an eight-byte timer total.
unsigned short	total_low_high			Bytes 3 and 4 of an eight-byte timer total.
unsigned long	start_tick_value			Tick-count in microseconds when timer was
				started, restarted, or continued. For line-related conditions at Layer 1, this value is stored in <i>l1_tick_count</i> ; for non-line conditions, use get_wall_time_286_ticks routine.
unsigned short	running		<ul> <li>An Experiencial State - C</li> <li>An Experimental State - C</li> <li>An Ex</li></ul>	Stopped. This variable is polled and a zero stop the timer from incrementing and sets the curre value to <i>timer_name.current</i> (understood as microseconds).
			-0	Running. All 1's in this variable causes the time to increment, showing a value that equals (wall-time ticks - <i>timer_name.start_tick_value</i> ) <i>timer_name.current</i> .
unsigned short	changed		~0	For future use.

## Table 62-4 Timer Structures

62-9

Туре	Variable	Meaning
extern unsigned long	l1_tick_count	This variable counts ticks from the start of Run mode. Tick=sec, msec, etc., depending on FEB setup. Subtract early value from later value to create a timer. ACTIONS: { displayf ("%Id msecs ", (I1_tick_count - timer_name.start_tick_value));)
		Add to start_of_run_time to determine more precise current time for time-stamping events. Line Setup configured for emulate or monitor mode.
extern unsigned long	start_of_run_date	Date when Run mode entered. Byte 1 (low byte) indicates day byte 2 stores month; and bytes 3 and 4 indicates year. May be used to time-stamp events. See also <i>start_of_run_time</i> . Line Setup configured for emulate or monitor mode.
extern unsigned long	start_of_run_time	Time when Run mode entered. Byte 1 (low byte) indicates seconds; byte 2 stores minutes and byte 3 indicates hours. May be used to time-stamp events. See also start_of_run_date and 11_tick_count. † Line Setup configured for emulate or monitor mode.

#### Table 62-5 Timer Variables

```
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: %.2d: %.2d: %.2d\n",
(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:

unsigned long temp; convert\_tick\_count (l1\_tick\_count, &temp); timer\_name.current = 0; timer\_name.start\_tick\_value = temp; timer\_name.running = -0;

The convert\_tick\_count routine converts  $ll_tick_count$  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:

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:

```
if (timer_name.running != 0)
{
    unsigned long temp;
    convert_tick_count (l1_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.

62 Counters, Timers, and Accumulators

struct accumulator\_struct

ł

{

}

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; };

struct accumulator\_struct accumulator\_name={0,0,-0u1};

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.

#### INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

Туре	Variable	Meaning
Structure Name	accumulator_struct	Structure of an accumulator. Declared as type <i>struct</i> . 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.
unsigned long	last	Value of last sample; displayed on the tabular statistics screen.
unsigned long	maximum	Maximum value of all samples; displayed on the tabular statistics screen.
unsigned long	minimum	Minimum value of all samples; displayed on the tabular statistics screen. Should be initialized as ~0ul.
unsigned short	sample_count	Number of samples.
unsigned long	total_high	High four bytes of an eight-byte accumulator total.
unsigned short	total_low_low	Low two bytes of an eight-byte accumulator total.
unsigned short	total_low_high	Bytes 3 and 4 of an eight-byte accumulator total
unsigned short	changed	For future use.

### Table 62-6 Accumulator Structures

# 62.4 Routines

## get\_68k\_phys\_addr

#### <u>Synopsis</u>

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.

#### <u>Inputs</u>

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

#### <u>Synopsis</u>

extern void send\_stat\_message(struct\_stat\_msg\_ptr); 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.

#### <u>Inputs</u>

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;
```

```
struct counter_structure
 {
  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_structure counter_dte_info = {0, 0, 0, -0ul};
extern unsigned long get_68k_phys_addr();
}
LAYER: 2
     STATE:send_stat_message
        CONDITIONS: ENTER STATE
        ACTIONS:
        ł
         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

#### <u>Synopsis</u>

```
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 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.

62 Counters, Timers, and Accumulators

Example

```
{
    unsigned long ticks;
}
LAYER: 2
    STATE: get_ticks
    CONDITIONS: KEYBOARD * "
    ACTIONS:
    {
        get_wall_time_ticks(&ticks);
    }
}
```

## get\_wall\_time\_286\_ticks

#### <u>Synopsis</u>

extern void get\_wall\_time\_286\_ticks(ticks\_286\_format\_ptr); 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: while 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

```
{
unsigned long ticks_286;
}
LAYER: 3
CONDITIONS: KEYBOARD " "
ACTIONS:
{
    get_wall_time_286_ticks(&ticks_286);
    displayf ("%lu", ticks_286);
}
```

#### convert\_tick\_count

#### <u>Synopsis</u>

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

ł

}

```
extern unsigned long l1_tick_count;
unsigned long converted_ticks;
LAYER: 1
     STATE: convert_ticks
        CONDITIONS: RECEIVE GOOD BCC
        ACTIONS:
        ł
         convert_tick_count(l1_tick_count, &converted_ticks);
         displayf ("%lu", converted_ticks);
        }
```



• .





Figure 63-1 Primitive Data Unit and sample Pointer-List Buffer being passed down the layers.

# 63 OSI

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).



PDU



## (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.

 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:

(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
```

ł

{

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\_low; unsigned short tick\_count\_low; unsigned short available\_space\_offset; unsigned short bytes\_remaining; unsigned long bcc\_indicator; unsigned char data [4064]; };

- }
- 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\_lo\_dl\_il\_buff <<16);</pre>

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.

Τa	able	63-1
OSI	Stru	ictures

Туре	Variable	Value (hex/decimal)	Meaning
<u>Structure Nam</u>	<u>ne:</u> pdu		Structure of an OSI primitive data unit (PDU). Declared as type <i>struct</i> . Use this structure as follows Declare the entire structure. Make a pointer to a PDU by shifting $m_{-}\log dl_{-}pdu_{-}seg$ (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 << 16). Reference a structure-pointer variable as follows: pdu_pointer->primitive_code.
unsigned char	primitive_code		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 <i>m</i> _lo_ <i>dl</i> _prmtv_code (or up_ <i>n</i> _prmtv_code).
unsigned char	path	0-8	Path number, both directions. The value of this variable is also stored in external variable <i>m</i> _lo_ <i>dl</i> _prmtv_path (or up_ <i>n</i> _prmtv_path).
unsigned long	parameter		For future use. At present, under user control.
unsigned short	relay_baton		Maintain bit passed with an interlayer-message buffer, both directions. Zero in this variable identifies maintain bit.
unsigned short	il_buffer_number		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_ $n_il_buff$ ).
unsigned char	buffer_contents	0	Contains data-character buffer type. Must be used for buffer being passed up.
		1	Contains pointer-list buffer type. May be used for buffers being passed up, but is currently used primarily for buffers being passed down.
unsigned short	data_start_offset		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).
unsigned short	data_length		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.
Туре	Variable	Value (hex/decimal)	Meaning
----------------	-----------------------	-----------------------	---
Structure Nam	e: il_buffer		Structure of an interlayer-message buffer, both directions. Declared as type <i>struct</i> . Use this structure as follows. Declare the entire structure. Make a pointer to an il_buffer by shifting <i>m</i> _lo_dl_il_buff (or up_n_il_buff) 16 bits to the left:
			<pre>iI_buffer_pointer = (void *) ((long) (lo_dl_il_buff &lt;&lt; 1 Then convert this pointer to a pointer to an il_buffe structure: struct il_buffer * il_buffer_pointer. Reference a structure-pointer variable as follows: il_buffer_pointer-&gt;tick_count_low.</pre>
unsigned short	lock	0	Internal variable which prevents structure from being updated by more than one program at the same time.
unsigned short	maintain_bits		Two-byte variable which provides the status of the maintain bits. A bit with a value of 1 is in use.
unsigned short	buffer_size	1000/4096	Currently, the only value.
unsigned short	transmit_tag		Bits 1-3 define bcc indication:
		0 1 2 3 4	no bcc good bcc bad bcc abort half bad bcc (DDCMP)
			Bits 4-8 for future use.
unsigned short	receive_tag		Bits 1-3 define bcc indication: -
		0 1 2 3 4	no bcc good bcc bad bcc abort half bad bcc (DDCMP)
			Bit 4 identifies side of the line:
		0 1	td rd
			Bit 5-message buffer overflow:
		0 1	frame fits in buffer frame too large for the buffer
			Bits 6-8 for future use.
unsigned long	char_buff_frame_start		Location in the character buffer of the start of the buffered data.
unsigned long	char_buff_frame_end		Location in the character buffer of the end of the buffered data.

Table 63-1 (continued)

(il\_buffer structure continued on next page)

.

Туре	Variable	Value (hex/dec	imal) Meaning
l_buffer (cont	inued)		
insigned short	tick_count_high		Value of internal variable that counts the number of times <i>I1_tick_count</i> has reached its maximum value. Together, the three <i>II_buffer</i> tick-count variables preserve at each layer the original time when the end of the data (BCC) was clocked int the buffer.
insigned short	tick_count_mid		16 high-order bits of 32-bit <i>l1_tick_count</i> .
nsigned short	tick_count_low		16 low-order bits of 32-bit I1_tick_count.
nsigned short	available_space_offset		Offset to the next available space in the interlayer-message buffer.
insigned short	bytes_remaining		Available number of bytes remaining in the buffe
nsigned long	bcc_indicator	0	reserved
insigned char	data [4064]		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.
Structure Nam	<u>ne:</u> 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 <i>struct</i> . Use this structure as follows. Declare the entire structure. Make a pointer to an il_list_header by shifting up_n_il_buff (or $m_lo_dl_il_buff$ ) 16 bits to the left and adding the data_start_offset from the PDU structure (also
			stored as external variable up_n_sdu or m_lo_di_sdu_offset): II_list_header_pointer = (void *)(((long)up_n_II_buff) << 16) + up_n_sdu) Then convert this pointer into a pointer to an II_list_header structure: struct II_list_header * II_list_header_pointer. Reference a structure-pointer variable as follow II_list_header_pointer->last_node_offset.
unsigned short	first_node_offset		<pre>stored as external variable up_n_sdu or m_lo_dl_sdu_offset): ii_list_header_pointer = (void *)(((long)up_n_il_buff) &lt;&lt; 16) + up_n_sdu) Then convert this pointer into a pointer to an ii_list_header structure: struct il_list_header * il_list_header_pointer. Reference a structure-pointer variable as follow il_list_header_pointer-&gt;last_node_offset. Offset from the beginning of the buffer to the first text node in the buffer. Varies according t the layer at which the buffer is located. At Lay</pre>
unsigned short unsigned short	first_node_offset last_node_offset		<pre>stored as external variable up_n_sdu or m_lo_dl_sdu_offset): ii_list_header_pointer = (void *) (((long)up_n_ii_buff) &lt;&lt; 16) + up_n_sdu) Then convert this pointer into a pointer to an ii_list_header structure: struct ii_list_header * ii_list_header_pointer. Reference a structure-pointer variable as follow ii_list_header_pointer-&gt;last_node_offset.</pre> Offset from the beginning of the buffer to the first text node in the buffer. Varies according t the layer at which the buffer is located. At Lay 2, the offset would be to different starting node

# Table 63-1 (continued)

Туре	Variable	Value (hex/decimal)	Meaning
Structure Name	a. il list node		Structure of text nodes in an interlayer-message
			buffer. Only for primitives sent down from the layer above. Declared as type <i>struct</i> . 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_d/_il_buff) 16 bits to
			the left and adding the first node_offset (or last_node_offset) from the li_list_header structure: li_list_node_pointer =
			(void *)(((long)up n   l buff << 16) +
			II_list_header_pointer->first_node_offset). Point to the next node as follows:
			next node pointer = (il list node pointer +
			ll_list_node_pointer->next_node_offset).
unsigned char *	data_pointer		Pointer to the data in a text node.
unsigned short	data_length		Length of the data in a text node.
unsigned short	next_node_offset		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
n an an Anna an Anna Anna An Anna Anna A			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.

## Table 63-1 (continued)

•

# 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 <u>insert\_il\_buff\_list\_cnt</u> 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.

Туре	Variable	Value (hex/	decimal) Meaning
extern volatile unsigned char	ph_prmtv_type	20/32	ph activate req
		21/33	ph activate ind
		22/34	ph activate resp
		23/35	ph activate conf
		24/36	ph data req
		25/37	ph data ind
		2a/42	ph reset req
		2b/43	ph reset ind
		2c/44	ph reset resp
	•	2d/45	ph reset conf
		2e/46	ph deactivate req
		2f/47	ph deactivate ind
		30/48	ph debug req
		31/49	ph debug ind
		33/51	ph error report ind
		34/52	ph xmit req
		35/53	ph set idle req
		38/56	ph mgt facility req
		39/57	ph mgt facility ind
			OSI primitive code for primitives
			moving between Layers 1 and 2
			Line Setup configured for
			emulate mode only.

Table 63-2 Layer 1 OSI Variables

INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

lo_ph_prmtv m_lo_ph_prmtv		True when an OSI primitive is received at Layer 2 from Layer 1. Line Setup configured for emulate mode only.
m_lo_ph_prmtv		
		True when an OSI primitive is received at Layer 2 from Layer 1. Line Setup configured for emulate or monitor mode.
up_dl_prmtv		True when an OSI primitive is received at Layer 2 from Layer 3. Line Setup configured for emulate mode only.
lo_ph_pdu_seg		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.
m_lo_ph_pdu_seg		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 of monitor mode.
lo_ph_prmtv_code	21/33 23/35 25/37 2b/43 2d/45 2f/47 31/49 33/51 39/57	ph activate ind ph activate conf ph data ind ph reset ind ph reset conf ph deactivate ind ph debug ind ph error report ind 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 25/37	td ph data ind rd ph data ind OSI primitive code received at Layer 2 in a PDU from Layer 1
	lo_ph_pdu_seg m_lo_ph_pdu_seg lo_ph_prmtv_code	lo_ph_pdu_seg m_lo_ph_pdu_seg lo_ph_prmtv_code 21/33 23/35 25/37 2b/43 2d/45 2f/47 31/49 33/51 39/57 m_lo_ph_prmtv_code 24/36

Table 63-3 Layer 2 OSI Variables

Туре	Variable	Value (hex/de	ecimal) M	eaning
xtern volatile const unsigned char	lo_ph_prmtv_path	0-8	2 in a PDU f	received at Layer rom Layer 1. Line ured for emulate
ktern volatile const unsigned char	m_lo_ph_prmtv_path	0-8	2 in a PDU f	r received at Laye rom Layer 1. Line ured for emulate o le.
ttern volatile unsigned short	lo_ph_il_buff		iAPX-286 se received at from Layer number can pointer by si	uffer number (an gment number) Layer 2 in a PDU I. This segment be converted to a nifting it left 16 bits onfigured for de only.
ktern volatile unsigned short	m_lo_ph_il_buff		IAPX-286 se received at 1 from Layer number can pointer by sl Line Setup c	affer number (an gment number) Layer 2 in a PDU 1. This segment be converted to a hifting it left 16 bit configured for nonitor mode.
ktern volatile unsigned short	lo_ph_sdu		Layer 2 from to where the	ive received at n Layer 1, the offs e service data unit e Setup configured mode only.
xtern volatile unsigned short	m_lo_ph_sdu_offset		Layer 2 from to where the begins. Line	tive received at n Layer 1, the offs service data unit s Setup configured or monitor mode.
xtern volatile unsigned short	m_lo_ph_sdu_size		an interlayed displayed as 2 trace scro Layer 2 from data length	service data unit in r-message buffer, s SIZE on the Laye sen. Received at m Layer 1. Same in a PDU. Line gured for emulate de
extern volatile unsigned short	up_dl_pdu_seg		OSI primitiv iAPX-286 s received at 3. This set be converte shifting it le	e data unit (PDU) egment number Layer 2 from Lay gment number can ed to a pointer by ft 16 bits. Line gured for emulate

Table 63-3 (continued)

Туре	Variable	Value (hex/	decimal) Meaning
extern volatile const unsigned char	up_dl_prmtv_code	40/64 42/66 44/68 48/72 4a/74 4c/76 4e/78 50/80 52/82 58/88	di conn req di conn resp di data req di expd data req di reset req di reset resp di disconn req di debug req di unit data req di mgt facility req
			OSI primitive code received at Layer 2 in a PDU from Layer 3 Line Setup configured for emulate mode only.
extern volatile const unsigned char	up_dl_prmtv_path	0-8	Path number received at Layer 2 in a PDU from Layer 3. Line Setup configured for emulate mode only.
extern volatile unsigned short	up_dl_ll_buff		Interlayer-buffer number (an iAPX-286 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.
extern volatile unsigned short	up_dl_sdu		Offset to the start (header node) of the service data unit i an interlayer-message buffer. Received at Layer 2 from Laye 3. Same as <i>data_start_offset</i> i a PDU. Line Setup configured for emulate mode only.
extern unsigned long	l2_tick_count		32-bit <i>I1_tick_count</i> stored in header of most recent IL buffe 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.

Table 63-3 (continued)

		Ta	ble	63-4		
L	ayer	3	OSI	Varia	bles	

extern event	lo di prmtv		True when an OSI primitive is
			received at Layer 3 from Layer
			2. Line Setup configured for
			emulate mode only.
extern event	m_lo_dl_prmtv		True when an OSI primitive is
			received at Layer 3 from Laye
		-	2. Line Setup configured for emulate or monitor mode.
extern event	up_n_prmtv		True when an OSI primitive is
			received at Layer 3 from Laye
			4. Line Setup configured for
			emulate mode only.
extern volatile unsigned short	lo_dl_pdu_seg		OSI primitive data unit (PDU)
			iAPX-286 segment number received at Layer 3 from Laye
			2. This segment number can
			be converted to a pointer by
			shifting it left 16 bits. Line
			Setup configured for emulate
			mode only.
extern volatile unsigned short	m_lo_dl_pdu_seg		OSI primitive data unit (PDU)
			IAPX-286 segment number
			received at Layer 3 from Laye 2. This segment number can
			be converted to a pointer by
			shifting it left 16 bits. Line
			Setup configured for emulate
			monitor mode.
extern volatile const unsigned char	lo_dl_prmtv_code	41/65	di conn ind
		43/67 45/69	di conn conf di data ind
		49/73	di expd data ind
		4b/75	di reset ind
		4d/77	di reset conf
		4f/79	di disconn ind
		51/81 53/83	di debug ind di unit data ind
		55/85	di error report ind
		59/89	di mgt facility ind
			OCI primitive and reactived a
			OSI primitive code received a Layer 3 in a PDU from Layer
			Line Setup configured for
			emulate mode only.

Туре	Variable	Value (hex/	decimal) Meaning
extern volatile const unsigned char	m_lo_dl_prmtv_code	44/68 45/69 48/72 49/73 54/84 55/85	td di data ind rd di data ind td di expd data ind rd di expd data ind td di unit data ind rd di unit data ind
			OSI primitive code received at Layer 3 in a PDU from Layer 2 Line Setup configured for emulate or monitor mode.
extern volatile const unsigned char	lo_dl_prmtv_path	0-8	Path number received at Layer 3 in a PDU from Layer 2. Line Setup configured for emulate mode only.
extern volatile const unsigned char	m_lo_dl_prmtv_path	0-8	Path number received at Layer 3 in a PDU from Layer 2. Line Setup configured for emulate of monitor mode.
extern volatile unsigned short	lo_dl_il_buff		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.
extern volatile unsigned short	m_lo_dl_il_buff -		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.
extern volatile unsigned short	lo_dl_sdu		In OSI primitive received at Layer 3 from Layer 2, the offs to where the service data unit begins. Line Setup configured for emulate mode only.
extern volatile unsigned short	m_lo_dl_sdu_offset		In OSI primitive received at Layer 3 from Layer 2, the offs to where the service data unit begins. Line Setup configured for emulate or monitor mode.
extern volatile unsigned short	m_lo_dl_sdu_size		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 data_length in a PDU. Line Setup configured for emulate of monitor mode.

# Table 63-4 (continued)

63 OSI

Туре	Variable	Value (hex/c	decimal) Meaning
extern volatile unsigned short	up_n_pdu_seg		OSI primitive data unit (PDU) IAPX-286 segment number received at Layer 3 from Laye 4. This segment number can be converted to a pointer by shifting it left 16 bits. Line Setup configured for emulate mode only.
extern volatile const unsigned char	up_n_prmtv_code	60/96 62/98 64/100 66/102 68/104 6a/106 6c/108 6e/110 70/112 72/114 74/116 76/118 78/120	n conn req n conn resp n data req n data ack req n expd data req n reset reg n reset resp n disconn req n debug req n unit data req n qual data ack req n gual data ack req n mgt facility req
			OSI primitive code received at Layer 3 in a PDU from Layer Line Setup configured for emulate mode only.
extern volatile const unsigned char	up_n_prmtv_path	0-8	Path number received at Laye 3 in a PDU from Layer 4. Lin Setup configured for emulate mode only.
extern volatile unsigned short	up_n_li_buff		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 bit Line Setup configured for emulate mode only.
extern volatile unsigned short	up_n_sdu		Offset to the start (header node) of the service data unit an interlayer-message buffer. Received at Layer 3 from Lay 4. Same as <i>data_start_offset</i> a PDU. Line Setup configured for emulate mode only.
extern unsigned long	I3_tick_count		32-bit 11_tick_count stored in header of most recent IL buff 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.

Table 63-4 (continued)

INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

Туре	Variable	Value (hex/decimal)	Meaning
extern event	lo_n_prmtv	rece 3.	when an OSI primitive is lived at Layer 4 from Layer Line Setup configured for late mode only.
extern event	m_lo_n_prmtv	rece 3.	e when an OSI primitive is lived at Layer 4 from Layer Line Setup configured for late or monitor mode.
extern event	up_t_prmtv	rece 5.	when an OSI primitive is pived at Layer 4 from Layer Line Setup configured for late mode only.
extern volatile unsigned short	lo_n_pdu_seg	iAP) rece	primitive data unit (PDU) (-286 segment number lived at Layer 4 from Layer
		be c shift Setu	This segment number can converted to a pointer by ing it left 16 bits. Line up configured for emulate le only.
extern volatile unsigned short	m_lo_n_pdu_seg	IAP) rece 3. be c shift Setu	primitive data unit (PDU) <-286 segment number bived at Layer 4 from Layer This segment number can converted to a pointer by ting it left 16 bits. Line up configured for emulate on hitor mode.
extern volatile const unsigned char	lo_n_prmtv_code	63/99 n c 65/101 n d 67/103 n d 69/105 n e 6b/107 n r 6d/109 n r 6d/109 n r 6f/111 n d 71/113 n d 73/115 n u 75/117 n q 77/119 n q 79/121 n r 7a/122 n e OS Lay	onn ind onn conf ata ind ata ack ind kpd data ind sset ind sset ind set conf sconn ind ebug ind nit data ind ual data ind ual data ack ind ngt facility ind rror report ind i primitive code received at er 4 in a PDU from Layer 3 a Setup configured for

# Table 63-5 Layer 4 OSI Variables

# Table 63-5 (continued)

Туре	Variable	Value (hex/c	decimal) Meaning
extern volatile const unsigned char	m_lo_n_prmtv_code	64/100 65/101 68/102 69/103 74/116 75/117	td n data ind rd n data ind td n expd data ind rd n expd data ind td n unit data ind rd n unit data ind
			OSI primitive code received at Layer 4 in a PDU from Layer 3 Line Setup configured for emulate or monitor mode.
extern volatile const unsigned char	lo_n_prmtv_path	0-8	Path number received at Layer 4 in a PDU from Layer 3. Line Setup configured for emulate mode only.
extern volatile const unsigned char	m_lo_n_prmtv_path	0-8	Path number received at Layer 4 in a PDU from Layer 3. Line Setup configured for emulate o monitor mode.
extern volatile unsigned short	lo_n_il_buff		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.
extern volatile unsigned short	m_lo_n_ll_buff		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 or monitor mode.
extern volatile unsigned short	lo_n_sdu	• * · · · · · · · · · · · · · · · · · ·	In OSI primitive received at Layer 4 from Layer 3, the offer to where the service data unit begins. Line Setup configured for emulate mode only.
extern volatile unsigned short	m_lo_n_sdu_offset		In OSI primitive received at Layer 4 from Layer 3, the offs to where the service data unit begins. Line Setup configured for emulate or monitor mode.
extern volatile unsigned short	m_lo_n_sdu_size		Size of the service data unit ir an interlayer-message buffer. Received at Layer 4 from Lay 3. Same as <i>data_length</i> in a PDU. Line Setup configured f emulate or monitor mode.

# INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

Туре	Variable	Value (hex/decin	nal) Meaning
extern volatile unsigned short	up_t_pdu_seg	i. r . t	DSI primitive data unit (PDU) APX-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.
extern volatile const unsigned char	up_t_prmtv_code	82/130 t 84/132 t 88/136 t 8e/142 t 90/144 t 92/146 t	conn req conn resp data req expd data req disconn req debug req unit data req mgt facility req
		L	DSI primitive code received at _ayer 4 in a PDU from Layer 5. _ine Setup configured for emulate mode only.
extern volatile const unsigned char	up_t_prmtv_path	2	Path number received at Layer 4 in a PDU from Layer 5. Line Setup configured for emulate mode only.
extern volatile unsigned short	up_t_ll_buff		Interlayer-buffer number (an APX-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.
extern volatile unsigned short	up_t_sdu		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 <i>data_start_offset</i> in a PDU. Line Setup configured for emulate mode only.
extern unsigned long	I4_tick_count		32-bit <i>I1_tick_count</i> stored in header of most recent IL buffe 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.

# Table 63-5 (continued)

.

Туре	Variable	Value (hex/c	lecimal) Meaning
extern event	lo_t_prmtv		True when an OSI primitive is received at Layer 5 from Layer 4. Line Setup configured for emulate mode only.
extern event	m_lo_t_prmtv		True when an OSI primitive is received at Layer 5 from Layer 4. Line Setup configured for emulate or monitor mode.
extern event	up_s_prmtv		True when an OSI primitive is received at Layer 5 from Laye 6. Line Setup configured for emulate mode only.
extern volatile unsigned short	lo_t_pdu_seg		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.
extern volatile unsigned short	m_lo_t_pdu_seg		OSI primitive data unit (PDU) IAPX-286 segment number received at Layer 5 from Laye 4. This segment number can be converted to a pointer by shifting it left 16 bits. Line Setup configured for emulate of monitor mode.
extern volatile const unsigned char	lo_t_prmtv_code	81/129 83/131 85/133 89/137 8f/143 91/145 93/147 95/149 99/153	t conn ind t conn conf t data ind t expd data ind t disconn ind t debug ind t unit data ind t error report ind t mgt facility ind OSI primitive code received at Layer 5 in a PDU from Layer 4
extern volatile const unsigned char	m_lo_t_prmtv_code	84/132 85/133 88/136 89/137 94/148 95/149	emulate mode only. td t data ind rd t data ind rd t expd data ind rd t expd data ind td t unit data ind rd t unit data ind OSI primitive code received a Layer 5 in a PDU from Layer

# Table 63-6 Layer 5 OSI Variables

63-23

Туре	Variable	Value	(hex/decimal)	Meaning
extern volatile const unsigned char	lo_t_prmtv_path	0-8	5 in a	number received at Layer PDU from Layer 4. Line configured for emulate only.
extern volatile const unsigned char	m_lo_t_prmtv_path	0-8	5 in a Setup	number received at Layer PDU from Layer 4. Line configured for emulate or or mode.
extern volatile unsigned short	lo_t_il_buff		iAPX- receiv from l numbe pointe Line S	ayer-buffer number (an 286 segment number) ed at Layer 5 in a PDU Layer 4. This segment er can be converted to a r by shifting it left 16 bits. Setup configured for te mode only.
extern volatile unsigned short	m_lo_t_il_buff		IAPX- receiv from I numbo pointe Line S	yer-buffer number (an 286 segment number) ed at Layer 5 in a PDU Layer 4. This segment er can be converted to a ar by shifting it left 16 bits. Setup configured for te or monitor mode.
extern volatile unsigned short	lo_t_sdu		Layer to wh begins	I primitive received at 5 from Layer 4, the offset ere the service data unit s. Line Setup configured nulate mode only.
extern volatile unsigned short	m_lo_t_sdu_offset		Layer to wh begin:	I primitive received at 5 from Layer 4, the offse ere the service data unit s. Line Setup configured nulate or monitor mode.
extern volatile unsigned short	m_lo_t_sdu_size		an int Recei 4. Si PDU.	of the service data unit in erlayer-message buffer. ved at Layer 5 from Layer ame as <i>data_length</i> in a Line Setup configured for ate or monitor mode.
extern volatile unsigned short	up_s_pdu_seg		IAPX- recei 6. T be co shiftir Setup	primitive data unit (PDU) -286 segment number ved at Layer 5 from Layer his segment number can onverted to a pointer by ng it left 16 bits. Line o configured for emulate a only.

# Table 63-6 (continued)

Туре	Variable	Value (hex/	decimal) Meaning
xtern volatile const unsigned char	up_s_prmtv_code	a0/160	s conn reg
<b>--</b>		a2/162	s conn resp
		a4/164	s data reg
		a8/168	s expd data req
		ac/172	s release req
		ae/174	s release resp
		b0/176	s debug reg
		b2/178	s unit data reg
		b8/184	s mgt facility req
			OSI primitive code received at Layer 5 in a PDU from Layer 6 Line Setup configured for emulate mode only.
extern volatile const unsigned char	up_s_prmtv_path	0-8	Path number received at Laye 5 in a PDU from Layer 6. Line Setup configured for emulate mode only.
extern volatile unsigned short	up_s_il_buff		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 bit Line Setup configured for emulate mode only.
extern volatile unsigned short	up_s_sdu		Offset to the start (header node) of the service data unit an interlayer-message buffer.
			Received at Layer 5 from Lay 6. Same as data_start_offset
			a PDU. Line Setup configured
	and a second		for emulate mode only.
extern unsigned long	15_tick_count		32-bit <i>I1_tick_count</i> stored in header of most recent IL buff 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.

Table 63-6 (continued)

Туре	Variable	Value (hex/decir	mal) Meaning
extern event	lo_s_prmtv		True when an OSI primitive is received at Layer 6 from Layer 5. Line Setup configured for emulate mode only.
extern event	m_lo_s_prmtv		True when an OSI primitive is received at Layer 6 from Laye 5. Line Setup configured for emulate or monitor mode.
extern event	up_p_prmtv		True when an OSI primitive is received at Layer 6 from Laye 7. Line Setup configured for emulate mode only.
extern volatile unsigned short	lo_s_pdu_seg	· · ·	OSI primitive data unit (PDU) iAPX-286 segment number received at Layer 6 from Laye 5. This segment number can be converted to a pointer by shifting it left 16 bits. Line Setup configured for emulate mode only.
extern volatile unsigned short	m_lo_s_pdu_seg		OSI primitive data unit (PDU) iAPX-286 segment number received at Layer 6 from Laye 5. This segment number can be converted to a pointer by shifting it left 16 bits. Line Setup configured for emulate of monitor mode.
extern volatile const unsigned char	lo_s_prmtv_code	a1/161 a3/163 a5/165 a9/169 ad/173 af/175 b1/177 b3/179 b5/181 b9/185	s conn ind s conn conf s data ind s expd data ind s release ind s release conf s debug ind s unit data ind s error report ind s mgt facility ind OSI primitive code received at Layer 6 in a PDU from Layer Line Setup configured for
extern volatile const unsigned char	m_lo_s_prmtv_code	a4/164 a5/165 a8/168 a9/169 b4/180 b5/181	td s data ind rd s data ind rd s data ind rd s expd data ind rd s expd data ind rd s unit data ind rd s unit data ind rd s unit data ind OSI primitive code received at Layer 6 in a PDU from Layer Line Setup configured for emulate or monitor mode.

# Table 63-7 Layer 6 OSI Variables

Туре	Variable	Value	(hex/decimal)	Meaning
extern volatile const unsigned char	lo_s_prmtv_path	0-8	6 in a l	umber received at Layer PDU from Layer 5. Line configured for emulate only.
extern volatile const unsigned char	m_lo_s_prmtv_path	0–8	6 in a l Setup	umber received at Layer PDU from Layer 5. Line configured for emulate o r mode.
extern volatile unsigned short	lo_s_il_buff		iAPX-2 receive from L numbe pointer Line Se	ver-buffer number (an 86 segment number) ed at Layer 6 in a PDU ayer 5. This segment r can be converted to a by shifting it left 16 bits stup configured for e mode only.
extern volatile unsigned short	m_lo_s_il_buff		Interlay IAPX-2 receive from L numbe pointer Line Se	ver-buffer number (an 86 segment number) ad at Layer 6 in a PDU ayer 5. This segment r can be converted to a by shifting it left 16 bits atup configured for e or monitor mode.
extern volatile unsigned short	lo_s_sdu		Layer to whe begins	primitive received at 6 from Layer 5, the offs re the service data unit . Line Setup configured ulate mode only.
extern volatile unsigned short	m_lo_s_sdu_offset		Layer to whe begins	primitive received at 6 from Layer 5, the offs re the service data unit . Line Setup configured pulate or monitor mode.
extern volatile unsigned short	m_lo_s_sdu_size		an inte Receiv 5. Sa PDU.	f the service data unit in erlayer-message buffer. ed at Layer 6 from Laye me as <i>data_length</i> in a Line Setup configured f te or monitor mode.
extern volatile unsigned short	up_p_pdu_seg		IAPX-/ receiv 7. Th be co shiftin	rimitive data unit (PDU) 286 segment number ed at Layer 6 from Laye is segment number can nverted to a pointer by g it left 16 bits. Line configured for emulate only.

Table 63-7 (continued)

See.

	Variable	Value (hex/d	lecimal) Meaning
extern volatile const unsigned char	up_p_prmtv_code	c0/192 c2/194	p conn req p conn resp
		c4/196 c8/200 cc/204	p data req p expd data req p release req
	•	ce/206 d0/208 d2/210 d8/216	p release resp p debug req p unit data req p mgt facility req
			OSI primitive code received at Layer 6 from Layer 7 in a PDU. Line Setup configured for emulate mode only.
extern volatile const unsigned char	up_p_prmtv_path	0-8	Path number received at Layer 6 from Layer 7 in a PDU. Line Setup configured for emulate mode only.
extern volatile unsigned short	up_p_il_buff		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 Line Setup configured for emulate mode only.
extern volatile unsigned short	up_p_sdu		Offset to the start (header node) of the service data unit i an interlayer-message buffer. Received at Layer 6 from Laye 7. Same as data_start_offset in a PDU. Line Setup configured for emulate mode only.
extern unsigned long	I6_tick_count		32-bit <i>I1_tick_count</i> stored in header of most recent IL buffer passed up to Layer 6. 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.

# Table 63-7 (continued)

63-28

Туре	Variable	Value (hex/dec	mal) Meaning
extern event	lo_p_prmtv	ana ang ang ang ang ang ang ang ang ang	True when an OSI primitive is received at Layer 7 from Layer 6. Line Setup configured for emulate mode only.
extern event	m_lo_p_prmtv		True when an OSI primitive is received at Layer 7 from Layer 6. Line Setup configured for emulate or monitor mode.
extern volatile unsigned short	lo_p_pdu_seg		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.
extern volatile unsigned short	m_lo_p_pdu_seg		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 of monitor mode.
extern volatile const unsigned char	lo_p_prmtv_code	c1/193 c3/195 c5/197 c9/201 cd/205 cf/207 d1/209 d3/211 d5/213 d9/217	p conn ind p conn conf p data ind p expd data ind p release ind p release conf p debug ind p unit data ind p unit data ind p error report ind p mgt facility ind OSI primitive code received at Layer 7 in a PDU from Layer 6 Line Setup configured for emulate mode only.
extern volatile const unsigned char	m_lo_p_prmtv_code	c4/196 c5/197 c8/200 c9/201 d4/212 d5/213	td p data ind rd p data ind td p expd data ind rd p expd data ind td p unit data ind rd p unit data ind
			OSI primitive code received at Layer 7 in a PDU from Layer 6 Line Setup configured for emulate or monitor mode.
extern volatile const unsigned char	lo_p_prmtv_path	0–8	Path number received at Laye 7 in a PDU from Layer 6. Line Setup configured for emulate mode only.

Table 63-8 Layer 7 OSI Variables

Туре	Variable	Value	(hex/deci	mal)	Meaning
extern volatile const unsigned char	m_lo_p_prmtv_path	0-8		7 in a Setup	umber received at Layer PDU from Layer 6. Line configured for emulate or or mode.
extern volatile unsigned short	lo_p_ii_buff			iAPX-2 receive from L numbe pointer Line S	yer-buffer number (an 286 segment number) ed at Layer 7 in a PDU ayer 6. This segment or can be converted to a r by shifting it left 16 bits. etup configured for te mode only.
extern volatile unsigned short	m_lo_p_ll_buff			iAPX-2 receive from L numbe pointer Line S	yer-buffer number (an 286 segment number) ed at Layer 7 in a PDU ayer 6. This segment or can be converted to a r by shifting it left 16 bits. etup configured for te or monitor mode.
extern volatile unsigned short	lo_p_sdu			Layer to whe begins	primitive received at 7 from Layer 6, the offset ere the service data unit Line Setup configured hulate mode only.
extern volatile unsigned short	m_lo_p_sdu_offset			Layer to whe begins	primitive received at 7 from Layer 6, the offset ere the service data unit 5. Line Setup configured nulate or monitor mode.
extern volatile unsigned short	m_lo_p_sdu_size			an inte Receiv 6. Sa PDU.	f the service data unit in erlayer-message buffer. ved at Layer 7 from Layer ume as <i>data_length</i> in a Line Setup configured for te or monitor mode.
extern unsigned long	17_tick_count			heade passe Prese origina the da into the config	: 11_tick_count stored in or of most recent IL buffer d up to Layer 7. rves at each layer the al time when the end of ata (BCC) was clocked he buffer. Line Setup gured for emulate or or mode.

## Table 63-8 (continued)

# 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.

## (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

### <u>Synopsis</u>

extern void \_get\_il\_msg\_buff(buffer\_number\_ptr, maintain\_bit\_ptr); unsigned short \* buffer\_number\_ptr; unsigned short \* maintain\_bit\_ptr;

### Description

The <u>get\_il\_msg\_buff</u> 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;
```

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

#### <u>Synopsis</u>

extern void \_start\_il\_buff\_list(il\_buffer\_number, start\_offset\_ptr); unsigned short il\_buffer\_number; unsigned short \* start\_offset\_ptr;

### Description

The <u>start\_il\_buff\_list</u> 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);</pre>

### <u>Example</u>

ł

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.

unsigned short il\_buffer\_number; unsigned short relay\_baton; unsigned short data\_start\_offset; }

```
STATE: start_a_list
CONDITIONS: KEYBOARD " "
ACTIONS:
{
    get_il_msg_buff(&il_buffer_number, &relay_baton);
    _start_il_buff_list(il_buffer_number, &data_start_offset);
```

/\* See \_insert\_il\_buff\_list\_cnt routine on how information is inserted in the buffer. \*/
}

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 <u>dup\_il\_buff\_list\_start</u> 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.

<u>Example</u>

Duplicate the header node of a buffer passed down from Layer 3.

and the second secon	
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 "ፂ 导ዄ((FOX))	
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 inform	mation is inserted in the buffer.
}	

## \_dup\_il\_buff\_list

### **Synopsis**

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.

extern volatile unsigned short up\_dl\_il\_buff; extern volatile unsigned short up\_dl\_sdu; unsigned short 12\_data\_start\_offset;

LAYER: 3

{

}

```
STATE: message
CONDITIONS: KEYBOARD " "
ACTIONS: DL_DATA REQ "민독地《FOX》"
LAYER: 2
STATE: duplicate_list
CONDITIONS: DL_DATA REQ
ACTIONS:
{
```

\_dup\_il\_buff\_list(up\_dl\_il\_buff, up\_dl\_sdu, &l2\_data\_start\_offset);

/\* See \_append\_il\_buff\_list\_cnt routine on how information is appended to the buffer. \*/
}

## \_open\_space\_in\_il\_buff

### <u>Synopsis</u>

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 <u>open\_space\_in\_il\_buff</u> 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:

(void \*)(((long)il buffer\_number << 16) + available\_space\_offset);</pre>

### 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:

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(il\_buffer\_number, relay\_baton);
unsigned short il\_buffer\_number;
unsigned short relay baton;

#### Description

The <u>free\_il\_msg\_buff</u> 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(il\_buffer\_number, new\_bit\_ptr); unsigned short il\_buffer\_number; unsigned short \* new\_bit\_ptr;

### Description

The <u>set\_maint\_buff\_bit</u> 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 <u>get\_il\_msg\_buff</u> 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 1111111 1111110). The last maintain bit allocated is 7FFF (0111111 1111111). 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.

### <u>Example</u>

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 12\_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 12\_data\_start\_offset; unsigned short available\_space\_offset; static unsigned char 12\_data[2] = {0x01, 0x00}; int i; unsigned char \* ptr\_12;

```
#define make_ptr(number, offset) ((void *)(((long)number << 16) + offset))
LAYER: 3
     STATE: send fox message
         CONDITIONS: KEYBOARD " "
         ACTIONS: DL DATA REQ "ዒ ፟ ዄ ((FOX))"
LAYER: 2
     STATE: send_a_buffer
         CONDITIONS: DL DATA REQ
         ACTIONS:
/* See <u>insert_il_buff_list_cnt</u> routine for an explanation of how information is inserted in the buffer. */
          _dup_il_buff_list_start (up_dl_il_buff, up_dl_sdu, &l2_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 = data_l2[i];
             ptr_12++;
            }
          ptr 12 -=2;
          _insert_il_buff_list_cnt(up_dl_il_buff, l2_data_start_offset, ptr_l2, 2);
          _set_maint_buff_bit(up_dl_il_buff, &maintain_bit);
          _set_maint_buff_bit(up_dl_il_buff, &l2_relay_baton);
          send_ph_prmtv_below(up_dl_il_buff, 12_relay_baton, 12_data_start_offset, 0, 0x24, 0);
         3
LAYER: 1
      STATE: resend_buffer
         CONDITIONS: RECEIVE STRING "Eº₃ ((XXXX1001))"
         ACTIONS:
         {
           _set_maint_buff_bit(up_dl_il_buff, &resend_baton);
```

11 il transmit(up dl il\_buff, resend\_baton, 12\_data\_start\_offset, 1);

/\* See Section 59, Monitor/Transmit Line Data, for an explanation of the  $l1_il_transmit$  routine. \*/

CONDITIONS: RECEIVE STRING "匠匠((XXXX0001))" ACTIONS:

\_free\_il\_msg\_buff(up\_dl\_il\_buff, maintain\_bit);

/\* See \_free\_il\_msg\_buff for an explanation of this routine. \*/

# \_insert\_il\_buff\_list\_cnt

### <u>Synopsis</u>

3

}

```
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 <u>insert\_il\_buff\_list\_cnt</u> 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.

### <u>Returns</u>

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.

### <u>Example</u>

If text is to be copied into the buffer, a pointer to the text must be declared. If not, when calling the <u>insert\_il\_buff\_list\_cnt</u> 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.

#### {

unsigned short il\_buffer\_number; unsigned short relay\_baton; unsigned short 14\_relay\_baton unsigned short 13\_relay\_baton; unsigned short 12\_relay\_baton; unsigned short data\_start\_offset; unsigned short 12\_data\_start\_offset;

```
unsigned short available_space_offset;

static unsigned char data[] = "((FOX))";

static unsigned char 13_data[3] = {0x10, 0x04, 0x00};

static unsigned char 12_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_13, * ptr_12;
```

/\* Whenever make\_ptr is encountered, the first parameter will be shifted 16 bits to the left. . The second parameter will be added, and the result cast into a pointer. \*/

#define make\_ptr(number, offset) ((void \*)(((long)number << 16) + offset))

} LAYER: 5

```
STATE: begin_message
CONDITIONS: KEYBOARD " "
ACTIONS:
{
    _get_il_msg_buff(&il_buffer_number, &relay_baton);
    _start_il_buff_list(il_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\_t\_prmtv\_below for an explanation of this routine). \*/

\_insert\_il\_buff\_list\_cnt(il\_buffer\_number, data\_start\_offset, &data[0], length); send\_t\_prmtv\_below(il\_buffer\_number, relay\_baton, data\_start\_offset, 0, 0x84, 0); }

At Layer 4 a new maintain bit 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\_n\_prmtv\_below for an explanation of this routine).

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\_13 = make\_ptr(up\_n\_il\_buff, available\_space\_offset); for(i = 0; i < 3; i++) { \*ptr\_13 = l3\_data[i]; ptr\_13++; }

/\* 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, &l3\_relay\_baton); send\_dl\_prmtv\_below(up\_n\_il\_buff, l3\_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, &l2\_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++;
}</pre>

/\* 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\_12 -=2; if(\_insert\_il\_buff\_list\_cnt(up\_dl\_il\_buff, 12\_data\_start\_offset, ptr\_12, 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\_maint\_buff\_bit(up\_dl\_il\_buff, &l2\_relay\_baton); send\_ph\_prmtv\_below(up\_dl\_il\_buff, l2\_relay\_baton, l2\_data\_start\_offset, 0, 0x24, 0); }

The following text will be sent out onto the line and displayed as line data:

ችለሚ ፍለ THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG 0123456789NG

# \_append\_il\_buff\_list\_cnt

#### **Svnopsis**

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 <u>append\_il\_buff\_list\_cnt</u> 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 <u>insert\_il\_buff\_list\_cnt</u> 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 <u>append\_il\_buff\_list\_cnt</u> for every occurrence <u>insert\_il\_buff\_list\_cnt</u>. 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 <u>dup\_il\_buff\_list\_start</u>.

```
LAYER: 2
     STATE: insert_more
        CONDITIONS: DL DATA REQ
        ACTIONS:
         _dup_il_buff_list(up_dl_il_buff, up_dl_sdu, &l2_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_12 = 12_data[i];
             ptr_12++;
            }
         ptr 12 -=2;
         if(_append_il_buff_list_cnt(up_dl_il_buff, l2_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, &l2_relay_baton);
         send_ph_prmtv_below(up_dl_il_buff, l2_relay_baton, l2_data_start_offset, 0, 0x24, 0);
        3
```

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 ቢ ፍ ከ ት ነበረ መሆኑ እንዲሆኑ እንዲሆኑ

## (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\_ph\_to\_above

#### **Synopsis**
# 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 <u>\_get\_il\_msg\_buff</u> routine.

The second parameter is the returned maintain bit from the <u>\_get\_il\_msg\_buff</u> 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 <u>start\_il\_buff\_list</u>) 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.

```
{
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);
}</pre>
```

# (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

# <u>Synopsis</u>

```
extern void send_dl_prmtv_above(il_buffer_number, 12_relay_baton, 12_data_start_offset, size,
12_code, path);
unsigned short il_buffer_number;
unsigned short 12_relay_baton;
unsigned short 12_data_start_offset;
```

unsigned short 12\_data\_start\_offset; unsigned short size; unsigned char 12\_code; unsigned char path;

#### Description

The *send\_dl\_prmtv\_above* emulate routine passes a specified interlayer message buffer from Layer 2 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 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 <u>\_set\_maint\_buff\_bit</u>. 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.

# struct pdu

ł

ł

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\_seg; 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;

# INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

LAYER: 2 STATE: send\_buffer\_up CONDITIONS: PH\_DATA IND ACTIONS: { pdu\_ptr = (void \*)((long)lo\_ph\_pdu\_seg << 16); \_set\_maint\_buff\_bit(lo\_ph\_il\_buff, &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(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\_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:

# send\_ph\_prmtv\_below

# **Synopsis**

# 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 <u>\_get\_il\_msg\_buff</u> routine. (See <u>\_insert\_il\_buff\_list\_cnt</u> routine example at Layer 5.)

The second parameter is the returned maintain bit from a call to <u>\_set\_maint\_buff\_bit</u>. 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 <u>\_get\_il\_msg\_buff</u> routine. (See <u>\_insert\_il\_buff\_list\_cnt</u> 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 <u>insert\_il\_buff\_list\_cnt</u> 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, 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\_n\_prmtv\_above emulate routine passes a specified interlayer message buffer from Layer 3 to Layer 4 in an OSI primitive.

#### <u>Inputs</u>

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 <u>\_set\_maint\_buff\_bit</u>. 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;
};

```
struct pdu * pdu_ptr;
extern volatile unsigned short lo_dl_pdu_seg;
extern volatile const unsigned char lo_dl_prmtv_path;
extern volatile unsigned short lo_dl_il_buff;
extern volatile unsigned short lo_dl_sdu;
unsigned short l3_relay_baton;
```

LAYER: 3

# send\_m\_n\_prmtv\_above

# **Synopsis**

extern void send\_m\_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\_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  $m_lo_dl_il_buff$ ,  $m_lo_dl_sdu_offset$ , and  $m_lo_dl_sdu_size$  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:

# send\_dl\_prmtv\_below

# **Synopsis**

extern void send\_dl\_prmtv\_below(il\_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\_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 <u>\_get\\_il\\_msg\\_buff</u> routine. (See <u>\_insert\\_il\\_buff\\_list\\_cnt</u> routine example at Layer 5.)

The second parameter is the returned maintain bit from a call to <u>\_set\_maint\_buff\_bit</u>. 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 <u>\_get\_il\_msg\_buff</u> routine. (See <u>\_insert\_il\_buff\_list\_cnt</u> 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 <u>insert\_il\_buff\_list\_cnt</u> 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.

```
{
  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_maint_buff_bit(up_n_il_buff, &l3_relay_baton);
        send_dl_prmtv_below(up_n_il_buff, l3_relay_baton, up_n_sdu, 0, 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**

```
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 <u>\_set\_maint\_buff\_bit</u>. 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.

# {

# 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 l4\_relay\_baton;

LAYER: 4 STATE: send\_buffer\_up CONDITIONS: N\_DATA IND ACTIONS: {

# send\_m\_t\_prmtv\_above

# **Synopsis**

1

```
extern void send_m_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\_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:

# send\_n\_prmtv\_below

#### **Synopsis**

# 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 <u>\_get\\_il\\_msg\\_buff</u> routine. (See <u>\_insert\_il\\_buff\_list\_cnt</u> routine example at Layer 5.)

The second parameter is the returned maintain bit from a call to <u>\_set\_maint\_buff\_bit</u>. 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 <u>\_get\_il\_msg\_buff</u> routine. (See <u>\_insert\_il\_buff\_list\_cnt</u> 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 <u>insert\_il\_buff\_list\_cnt</u> 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.

```
{
  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
```

```
{
    _set_maint_buff_bit(up_t_il_buff, &14_relay_baton);
    send_n_prmtv_below(up_t_il_buff, 14_relay_baton, up_t_sdu, 0, 0x64, 0);
}
```

# (F) Layer 5 OSI Routines

ACTIONS:

The following routines pass OSI primitives from Layer 5 to either Layer 6 or Layer 4.

# send s prmtv above

#### **Synopsis**

```
extern void send_s_prmtv_above(il_buffer_number, 15_relay_baton, 15_data_start_offset, size,

15_code, path);

unsigned short il_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\_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  $lo_t_{il}_{buff}$  may be used to identify the buffer number.

The second parameter is the returned maintain bit from a call to <u>\_set\_maint\_buff\_bit</u>. 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.

# { 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 15\_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\_maint\_buff\_bit(lo\_t\_il\_buff, &15\_relay\_baton); send\_s\_prmtv\_above(lo\_t\_il\_buff, 15\_relay\_baton, lo\_t\_sdu, pdu\_ptr->data\_length, 0xa5, lo\_t\_prmtv\_path); }

# send\_m\_s\_prmtv\_above

# **Synopsis**

extern void send\_m\_s\_prmtv\_above(il\_buffer\_number, l5\_relay\_baton, l5\_data\_start\_offset, size, l5\_code, path); unsigned short il\_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:

# send\_t\_prmtv\_below

# **Synopsis**

extern void send\_t\_prmtv\_below(il\_buffer\_number, 15\_relay\_baton, 15\_data\_start\_offset, size, 15\_code, path); unsigned short il\_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\_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 <u>\_get\\_il\\_msg\\_buff</u> routine. (See <u>\_insert\_il\\_buff\_list\_cnt</u> routine example at Layer 5.)

The second parameter is the returned maintain bit from a call to <u>\_set\_maint\_buff\_bit</u>. 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 <u>\_get\_il\_msg\_buff</u> routine. (See <u>\_insert\_il\_buff\_list\_cnt</u> 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 <u>insert\_il\_buff\_list\_cnt</u> 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.

```
{
  extern volatile unsigned short up_s_il_buff;
  extern volatile unsigned short up_s_sdu;
  unsigned short 15_relay_baton;
```

LAYER: 5

3

STATE: pass\_buffer\_down CONDITIONS: S\_DATA REQ ACTIONS:

> { \_\_set\_maint\_buff\_bit(up\_s\_il\_buff, &15\_relay\_baton); \_\_send\_t\_prmtv\_below(up\_s\_il\_buff, 15\_relay\_baton, up\_s\_sdu, 0, 0x84, 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);

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\_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 <u>\_set\_maint\_buff\_bit</u>. 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 l6\_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, &l6\_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:

# send\_s\_prmtv\_below

# **Synopsis**

extern void send\_s\_prmtv\_below(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\_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 <u>\_get\_il\_msg\_buff</u> routine. (See <u>\_insert\_il\_buff\_list\_cnt</u> routine example at Layer 5.)

The second parameter is the returned maintain bit from a call to <u>\_set\_maint\_buff\_bit</u>. 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 <u>\_get\_il\_msg\_buff</u> routine. (See <u>\_insert\_il\_buff\_list\_cnt</u> 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 <u>insert\_il\_buff\_list\_cnt</u> 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 16\_relay\_baton;

LAYER: 6

{

}

STATE: pass\_buffer\_down CONDITIONS: P\_DATA REQ ACTIONS:

> '\_set\_maint\_buff\_bit(up\_p\_il\_buff, &l6\_relay\_baton); send\_s\_prmtv\_below(up\_p\_il\_buff, l6\_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 <u>\_get\_il\_msg\_buff</u> routine. (See <u>\_insert\_il\_buff\_list\_cnt</u> routine example at Layer 5.)

The second parameter is the returned maintain bit from the call to <u>\_get\_il\_msg\_buff</u>.

The third parameter is the returned offset (from a call to <u>start il\_buff\_list</u>) 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 <u>insert\_il\_buff\_list\_cnt</u> routine.) If values are entered for the code and path, variables for code and path need not be declared.

{

1

unsigned short il\_buffer\_number; unsigned short data\_start\_offset; unsigned short relay\_baton;

LAYER: 7

STATE: pass\_buffer\_down CONDITIONS: KEYBOARD " " ACTIONS: {

\_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); }

# INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

63-68

64 Print

# 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—printc, 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

Туре	Variable	Value (hex/decimal)	Meaning
Structure N	ame: print_buffe	r	Structure of the print buffer. Declared as type struct.
unsigned shor	t in	a-207/10-8199	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 <i>in</i> equals <i>out</i> , the print buffer is empty.
unsigned shor	t out	a-207/10-8199	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.
unsigned shor	t buffer_end	209/8201	offset to the physical end of the print buffer—i.e., to the end of the array named buffer (see below)
unsigned shor	t lock		when process is printing, locks out other processes from accessing the print buffer
char	polling	0 non-zero	print server is not polling print server is polling print buffer for text to print
char	overrun	0 non-zero	print buffer is not in overrun state print buffer is in overrun state—i.e., a process attempting to add text to the print buffer can't because unprinted text in the buffer would be overwritten. Following message will appear on printout: "print buffer overrun has occurred."
char	buffer [8192]		array of text transactions
<u>Structure N</u>	<u>lame:</u> _print_buff	tor of participan of gran of data jer tota antigation of the data ware and the second second second	An instance of the <i>print_buffer</i> structure, declared as type <i>extern struct print_buffer</i> . 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 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.)

```
#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 ck_buffer
             ACTIONS:
             {
              crnt_buffer_sz = ((_print_buffer.in + PRINT_BUFFER_SZ) - _print_buffer.out) %
                   PRINT BUFFER SZ;
              if(crnt_buffer_sz > STOP_POINT)
                suspend_rcrd_play();
              else if (crnt_buffer_sz < START_POINT)
                start_rcrd_play();
             1
```

TIMEOUT RESTART ck\_buffer 0.01

# 64.2 Variables

{

}

There are no variables associated exclusively with print functions.

# 64.3 Routines

# printc

#### **Synopsis**

extern void printc(character); const char character;

# Description

The *printc* routine outputs a single ASCII character to the print buffer for printing, converting the value provided as the argument into its ASCII equivalent. Decimal and octal values are converted to hexadecimal format before the ASCII equivalent is sought.

# **Inputs**

The only parameter is a numerical value. The 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 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 printer.

# <u>Example</u>

The printc entries on the left output the printed character given on the right:

printc('a');	a
printc(65);	Α
printc(0x65);	e
printc(065);	5

# 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.

# <u>Inputs</u>

#

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 0x (or 0X) 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 1 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.

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

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.

р

с

S

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 0D 0A, 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

ł

}

```
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 sprint routine is similar to the print routine, except that sprint 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 display f 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

#### <u>Synopsis</u>

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 nur	nber
##	#	

# 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

Page : 1 ##

2 ##

Page :

#p ####\n");

## 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  $\xi$ ) to the output string. Octal or hexadecimal values also may be included in the string, with octal preceded by  $\lambda$  and hex by  $\lambda$ . Pad each value to three integers with leading zeroes.

# INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

# <u>Example</u>

The following entry

prints("End of test.");

produces the following output to a printer:

End of test.

(1) State the second state of a state of a state of the state of th

1. And a substrate state of strain and dependent of several strate state state and an even of the constraints and a constraint for a strategic strategic for a straint of the constraints of the strategic dependent of the constraints and the terms of the substrategic of the first first first substrategic strategic strategic and the strategic strategic.

65 Disk I/O

# 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

#### <u>Synopsis</u>

#include <stdio.h>
extern FILE \* fopen(filename\_ptr, mode\_ptr);
const char \* filename\_ptr;
const char \* mode\_ptr;

#### 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.

#### <u>Inputs</u>

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:

Open an existing file for reading only. The file-position indicator is located at the start (character zero) of the file.

w

а

r

Create a file, or open an existing file, for writing only. For an existing file, truncate its length to zero and discard the contents.

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*.)

- Currently implemented the same as "r." Use "rb" for the fseek rb 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 lusr 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 O 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.
        }
```

"); ");

### 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.

### 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.

المتعام المتعالية المعام المتعالية المعالية المتعالية المتع		
#include <stdio.h></stdio.h>		
FILE * stream_ptr;		
}		
LAYER: 1		
STATE: open and close	a file	
CONDITIONS: ENTE		
		,,
	"Press O to open file.	
CONDITIONS: KEYB	UARD FOU"	
ACTIONS:		
{		
if((stream_ptr = fope	en ("FD2/usr/buff01", "r")) == 0)	
display prompt("		");
else		
display_prompt("	File opened	");
l anspiraty_prompt(	The opened.	,,
CONDITIONS: KEYB		
ACTIONS:		
{		
if(fclose(stream_ptr)	!= 0)	
display prompt("	Either file is already closed, or close cannot be executed.	");
else		
display prompt("	File closed	");
anspray_prompt(		,,
J		

### fflush

### **Synopsis**

#include <stdio.h> extern int fflush(stream\_ptr) 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.

# <u>Returns</u>

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></stdio.h>	
FILE * stream_ptr;	
extern volatile const unsigned char rcvd_frame_cntrl_byte_1;	
LAYER: 2	
STATE: write_then_fflush CONDITIONS: ENTER_STATE ACTIONS:	
if((stream_ptr = fopen("FD2/usr/frame_unkwn", "a")) == 0)	
display prompt ("Cannot open file.	");
else	),
display prompt("File opened.	");
pos_cursor(1,0);	,,
pos_cursor(1;0);	
CONDITIONS: RCV UNKNOWN	
ACTIONS:	
if (fprintf (stream ptr, "%02x\n", rcvd frame cntrl byte 1) < 0)	
displayf("Error in printing to stream.	\n");
else	<i>vii ),</i>
displayf("Print to stream completed.	\n");
if (fflush (stream ptr) != 0)	,,
display prompt ("Write error.	");
else	,,
display prompt("Write to file completed. Press C to close file.	");
<pre>windows</pre>	,,
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.

### <u>Returns</u>

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."

{	
#include <stdio.h></stdio.h>	
FILE * stream_ptr;	
int character;	
}	
LAYER: 1	
STATE: test_for_eof	
CONDITIONS: ENTER_STATE	
ACTIONS: PROMPT "Press O 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:	
A second seco	
if(fclose(stream_ptr) != 0)	
display_prompt("Either file is already closed, or close cannot be executed.	");
else	
display prompt("File closed.	");
}	

# ferror

### <u>Synopsis</u>

#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.

### <u>Returns</u>

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.

Experimental and a state of the second seco	
#include <stdio.h></stdio.h>	
FILE * stream_ptr;	
char data [6091];	
size_t n;	
LAYER: 1	
STATE: read_a_file	
CONDITIONS: ENTER_STATE	-
ACTIONS: PROMPT "Press O 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 "rR"	
ACTIONS:	
<pre>4</pre>	
$n = fread(data, 1, 6091, stream_ptr);$	
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 and a second s	
displayf("\n%.6091s", 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.
}
```

# 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.

🖌 en el consignador en la constante	
#include <stdio.h></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:	
an ang ang ang ang ang ang ang ang ang a	
if((stream_ptr = fopen("FD2/usr/buff01", "wb")) == 0)	
display_prompt("Cannot open file.	");
else	
display_prompt("File opened. Press P to write character.	");
}	

```
CONDITIONS: KEYBOARD "pP"
ACTIONS:
ł
character = fputc('h', stream_ptr);
if(character == EOF)
   ł
    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

#### <u>Synopsis</u>

#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 S 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.

{ #include <stdio.h></stdio.h>	
FILE * stream ptr;	
int character:	
LAYER: 1	
STATE: move indicator	
CONDITIONS: ENTER STATE	
ACTIONS: PROMPT "Press O 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. ");	
$pos\_cursor(0,14);$	
if (fseek (stream_ptr, 4, SEEK_SET) != 0)	
displays ("Improper fseek request.	");
else	
displays ("Fseek completed. Press S to seek new position.	");
}	
<pre>process } all states and states are states and states are states are states are states and states are states ar are states are states are</pre>	
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.	");
<pre>}</pre>	

```
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.

{			
	nclude <stdio.h></stdio.h>		
	LE * stream_ptr; t character;		
}			
LAY	YER: 1		
	STATE: move_indicator		
	CONDITIONS: ENTER_STATE		
	ACTIONS: PROMPT "Press O to op	pen file.	*
	CONDITIONS: KEYBOARD "00"		
	ACTIONS:		
	if((stream_ptr = fopen("FD2/usr/bu	(ff01", "rb")) == 0)	
	display_prompt("Cannot open fi	le.	");
	else		
	display_prompt("File opened. H	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 "" ACTIONS: {	
rewind(stream_ptr); display_prompt("Press G to get a character. }	");
CONDITIONS: KEYBOARD "gG" ACTIONS: {	
character =fgetc(stream_ptr); display_prompt("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	");
eise 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.

#include <stdio.h></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.	Managal Caracter 199	");
else	A Company A Weight A State Free Company A State S	
display_prompt("File opened.		");
n = fwrite(data, 1, sizeof(data)-1, stream_p	<i>&gt;tr</i> );	
<pre>pos_cursor(1,0);</pre>		
if(n != (sizeof(data)-1))		
displayf ("Write error.		n'');
else		
displayf("Write completed.		$n^{n};$
if(fclose(stream_ptr) != 0)	1.11年間1月4日時代各部人的新生产。 1.11日前日日	
displayf ("Either file is already closed, or	close cannot be executed.	");
else		
displayf ("File closed.		");
unlock(&key);		
}		
}		
LAYER: 1		
TEST: a		
STATE: write_and_signal	IS OLIVOK PROWN FOY"	
CONDITIONS: RECEIVE STRING "TH	IE QUICK BROWN FUX	
ACTIONS: SIGNAL xyz		
{		
write_fox();		
}		

```
TEST: b
STATE: write_only
CONDITIONS: ON_SIGNAL xyz
ACTIONS:
{
write_fox();
}
```

# unlock

### <u>Synopsis</u>

#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 the *feof* 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.

#include <trace\_buf.h>
#include <trace\_buf.h>
#include <stdio.h>
FILE \* stream\_ptr;
char data [6091];
size\_t n;
extern struct trace\_buf prog\_trbuf;
}

LAYER:	1	
ST	ATE: read_a_file	
	CONDITIONS: ENTER_STATE	
	ACTIONS: PROMPT "Press O 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.	");
	aispidy_prompt( The opened. Tress K to read the file.	/,
	; CONDITIONS: KEYBOARD "rR"	
	ACTIONS: REFBOARD TR	
	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.	");
	}	
	* } The Comparison of Comparison of Comparison (Comparison Comparison Comparison)	
	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.	");
	} · · · · · · · · · · · · · · · · · · ·	

### 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.

#include	<stdio.h></stdio.h>
{	

FILE \* stream\_ptr; char data [10];

#### LAYER: 1

}

STATE: read\_characters 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 G to get string.	");
}	
CONDITIONS: KEYBOARD "gG" ACTIONS:	
{ feets(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**

#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 G key is pressed, display the next character in the file.

#include <stdio.h></stdio.h>	
FILE * stream_ptr;	
int character, end;	
}	
LAYER: 1	
STATE: get_next_character	
CONDITIONS: ENTER_STATE	
ACTIONS: PROMPT "Press O 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 G to get a character.	");
displayf("\n");	
}	

```
CONDITIONS: KEYBOARD "gG"
ACTIONS:
character = fgetc(stream ptr);
if(character == EOF)
   {
    end = feof(stream_ptr);
    if(end != 0)
     display_prompt("EOF encountered.
                                                                                  ");
    else
     display_prompt("Read error.
                                                                                  ");
   }
else
    displayf ("%c", 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.
                                                                                  "):
}
```

### ungetc

#### <u>Synopsis</u>

#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.

#### <u>Inputs</u>

The first parameter is the character to be put into the input stream.

The second parameter is the stream pointer.

#### <u>Returns</u>

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.

,

{	
#include <stdio.h></stdio.h>	
FILE * stream_ptr;	
int character;	
}	
LAYER: 1	
STATE: get_next_character	
CONDITIONS: ENTER_STATE	_
ACTIONS: PROMPT "Press O 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 G to get a character.	");
}	
CONDITIONS: KEYBOARD "gG"	
ACTIONS:	
1	
character = fgetc(stream_ptr);	
if(character = EOF)	
display_prompt("End of file or read error.	");
else	,,
{	
pos_cursor(0,0);	
displayf("character = %c Press U to return character to stream.", character	cler);
}	
CONDITIONS: KEYBOARD "uU"	
ACTIONS:	
if((ungetc(character, stream_ptr)) == EOF)	
display_prompt("Character not returned.	");
else	
display_prompt("Character returned.	");
<pre>&gt;</pre>	
CONDITIONS: KEYBOARD "CC"	
ACTIONS:	
A second seco	
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. *fputc* 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 *displayf* writes output to the Display Window.

### fwrite

#### <u>Synopsis</u>

#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.

```
ł
#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 O 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)
            displays ("Cannot open new file.
                                                                                        ");
         else
                                                                                        ");
            displays ("New_file opened. 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:
         ł
         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)
    ł
    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";
}
```

LAYER: 1	
STATE: write_a_string	
CONDITIONS: ENTER_STATE	
ACTIONS: PROMPT "Press O 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(fputs(data, stream_ptr) != 0)	
display_prompt("Write error. Press C to close file.	");
else	
display_prompt("Write completed. Press C to close file.	");
ACTIONS:	
{	
if(fclose(stream ptr) != 0)	
display_prompt("Either file is already closed, or close cannot be executed.	");
else	7,
display prompt("File closed.	");
	<i>);</i>

# 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.

### 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 P key is pressed, only one character is copied, rather than the entire file.

and a start of the start of th	
\ #include <stdio.h></stdio.h>	
FILE * read_stream;	
FILE * write_stream;	
int character;	
LAYER: 1	
STATE: copy_a_character	
CONDITIONS: ENTER_STATE	
ACTIONS: PROMPT "Press O to open files.	
CONDITIONS: KEYBOARD "00"	
ACTIONS:	
if((read_stream = fopen("FD2/usr/buff01", "r")) == 0) {	
display_prompt("Cannot open buff01. ");	
<pre>pos_cursor(0,21); }</pre>	
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: KETBOARD PF	
1 footofood storem ) -	
character = fgetc(read_stream); if(character == EOF)	
is a strategy of the second second states and the second second second second second second second second second	
if(feof(read_stream) != 0)	
display_prompt("EOF encountered. Press C to close files. else	");
display_prompt("Read error. Press C to close files.	");

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) displayf ("Either buff01\_copy is already closed, or close cannot be executed. "); else display ("Buff01\_copy closed. "); }

# fprintf

#### <u>Synopsis</u>

#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 0x (or 0X) 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 1 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.

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.

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 Writes hexadecimal 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 %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

S

р

This routine returns the number of characters written, or a negative value if an output error occurred.

#### <u>Example</u>

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.

65 Disk I/O

#include <stdio.h></stdio.h>	
FILE * stream_ptr;	
extern volatile const unsigned char rcvd_frame_cntrl_byte_1;	
LAYER: 2	
STATE: save_unknowns	
CONDITIONS: ENTER STATE	
ACTIONS: PROMPT "Press O to open file.	
CONDITIONS: ENTER_STATE	
ACTIONS:	
if (stream at = fones ("ED2 (un) from a un from " " " "))	
if((stream_ptr = fopen("FD2/usr/frame_unkwn", "w")) == 0) display_prompt("Cannot open file.	
else	");
display_prompt("File opened.	");
CONDITIONS: BCV UNKNOWN	
ACTIONS: REV UNKNOWN	
if(fprintf(stream_ptr, "%02x\n", rcvd_frame_cntrl_byte_1) < 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.	");
A second s	

# (D) File Maintenance Routines

#### rename

#### **Synopsis**

#include <stdio.h>
extern int rename(oldfile\_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.

#### <u>Returns</u>

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.

#### <u>Example</u>

{

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.
```

remove

}

#### Synopsis

#include <stdio.h>
extern int remove(file\_ptr);
const char \* file\_ptr;

#### Description

This routine removes the named file from the disk. The file must be closed in order for the remove operation to succeed. Subsequent attempts to open the file will fail. Empty directories may also be removed with this routine.

");

·");

#### Inputs

The only input is a pointer to a string, i.e., the filename. It must be the absolute pathname, prefixed by the device name (HRD, FD1, or FD2).

#### **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

ł

}

Remove file oldfile from the /usr directory on the disk in floppy drive 1. Prompt whether or not the remove operation was successful.

```
#include <stdio.h>
LAYER: 1
     STATE: delete a file
        CONDITIONS: ENTER STATE
        ACTIONS: PROMPT "Press D to delete file.
        CONDITIONS: KEYBOARD "dD"
        ACTIONS:
        ł
         if(remove("FD1/usr/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></stdio.h>	
}	
LAYER: 1	
STATE: make directory	
CONDITIONS: ENTER STATE	
ACTIONS: PROMPT "Press M to make a directory.	•
CONDITIONS: KEYBOARD "mM"	
ACTIONS:	
■ 1.1 (1) (1) (1) (1) (1) (1) (1) (1) (1) (1	
if(mkdir("FD2/usr/disk_i_o") != 0)	
display_prompt("Directory not created.	");
else	
display_prompt("Directory created.	");
}	

# \_set\_file\_type

### <u>Synopsis</u>

```
#include <stdio.h>
extern int _set_file_type(pathname_ptr, type_buff_ptr);
char * pathanme_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 <u>set\_file\_type</u> 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

### 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.

# #include <stdio.h>

ł

}

```
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 O to open files.
        CONDITIONS: KEYBOARD "00"
        ACTIONS:
        ł
         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)
                                                                                        ");
            displays ("Cannot open new_file.
         else
            displays ("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:
ł
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:
ł
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.
}
```

# \_get\_file\_type

### <u>Synopsis</u>

```
#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

•
) :
,,
e);
.,,
; ; e)

INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108
# 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.

# INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

# Table 66-1 Status Structures

Туре	Variable	Value	(hex/decimal)	Meaning
Structure Name	e: unit_setup			Structure containing Line Setup and FEB tick-rate selections. Declared as type <i>extern struct</i> . Reference member variables of the structure as
				follows: unit_setup.speed_dce.
unsigned long	speed_dce			If Clock Source selection is Internal, this variable has Speed value entered on Line Setup. If Clock Source is External, this variable has DCE speed indicated under Clock Source: Internal Split.
unsigned long	speed_dte			If Clock Source selection is Internal, this variable has Speed value entered on Line Setup. If Clock Source is External, this variable has DTE speed Indicated under Clock Source: Internal Split.
unsigned long	usec_per_tick		a/10 54/100 3e8/1000 2710/10000 186a0/100000 f4240/100000	tick rate selected on FEB Setup 10 usec 100 usec 1 msec 10 msec 1000 msec 1 sec
unsigned char	bit_order_polarity	-	D 1 2 3	normal normal-inverse reverse-normal reverse-inverse
unsigned char	bits_per_byte	,	5–8	
unsigned char	clocking_type		0 1 2	internal external internal-split
unsigned char	data_source		0 1	disk line
unsigned char	format		0 1 2 3	sync bop async isoc
unsigned char	mode		0 1 2 3	automonitor monitor emulate dce emulate dte
unsigned char	parity		0 1 2 3 4	none even odd mark space
unsigned char	code_name [13]			ASCII, EBCDIC, etc.

66 Status

Туре	Variable		Value (hex/decimal	Meaning					
Structure Name	<u>e:</u> unit_confi	g		Structure containing unit configuration. Declare as type extern struct. Reference member variables of the structure as follows:					
e National de la companya de la company				unit_config.floppy_exists_mask.					
insigned char	floppy_exists_i	mask	1 2	floppy1 floppy2†					
insigned char	hard_disk		0 1	not present present					
insigned char	test_board		0 800 900 1 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9	not present present					
insigned char	mux			not present present					
insigned char	modem		0 1	not present present					
insigned char	num_mpms		0-4	number of MPM boards present					
truct mpm_info	mpm [4]			array of structures. Each element in the array an instance of the structure <i>mpm_info</i> and corresponds to one of four MPM boards which may be present. Reference member variables of the structure elements in the array as follows: <i>unit_config.mpm[0].present.</i>					
insigned char	cpm_rev			reserved					
insigned char	gbm_rev			reserved					
insigned char	pcm_rev			reserved					
insigned char	modem_rev			reserved					
unsigned char	mux_rev			reserved					
unsigned char	tim_type		f0/240	RS-232					
e in selen en en e			f1/241 f2/242 f3/243 f4/244	X.21 V.35 RS-449 expansion adaptor					
			f5/245 f6-fb/246-251 fc/252 fd/253	RC-8245 reserved ISDN G.703					
			fe/255 ff/255	T1 none					
unsigned long	last_ram_cpm	ר		the value of this variable plus one yields the CP memory size (in bytes)					

Table 66-1 (continued)

the drive is present.
For example, 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.

.

	Variable		e (hex/decimal)	Meaning
unsigned long	self_test_errors	15 (1) - 5 <sup>4</sup> 1 - 7 1 - 7	(mask)	self-test errors encountered during power-up††
			1	CPM DRAM error
			2	CPM 32-bit counter
			4	CPM System Timing Controller (9513a)
			8	
			10/16	MPM0 DRAM (tested from CPM-global bus
			20/32	MPM0 DRAM (tested from MPM0)
			40/64	MPM0 interrupt latch
			80/128	unused
			100/256	MPM1 DRAM (tested from CPM-global bus
			200/512	MPM1 DRAM (tested from MPM1)
			400/1024	MPM1 interrupt latch
			800/2048	unused
			1000/4096	MPM3 DRAM (tested from CPM-global bus
			2000/8192	MPM3 DRAM (tested from MPM3)
			4000/16384	MPM3 interrupt latch
			8000/32768	unused
			10000/65536	unused
			20000/131072	unused
			40000/262144	unused
			80000/524288	unused
			100000/1048576	unused
			200000/2097152	unused
			400000/4194304	unused
			800000/8388608	unused
			1000000/16777216	unused
			200000/33554432	unused
			4000000/67108864	unused
			8000000/134217728	unused
			1000000/268435456	unused
			2000000/536870912	unused -
			4000000/1073741824	unused
			8000000/2147483648	unused
unsigned long	version		6	current value for this version of unit_confi structure
unsigned long	model_number		19c8/6600	INTERVIEW 6600
	_		1a90/6800	INTERVIEW 6800 TURBO
			1b58/7000	INTERVIEW 7000
			1c20/7200	INTERVIEW 7200 TURBO
			1d4c/7500	INTERVIEW 7500
			1e14/7700	INTERVIEW 7700 TURBO

# Table 66-1 (continued)

(unit\_config continued on next page)

tt If (unit\_config.self\_test\_errors & mask) == mask, the error is present.
If (unit\_config.self\_test\_errors & 0xffffffff) == 0, no errors encountered during power-up.

Туре	Variable	Value (hex/decimal)	Meaning
unsigned char	feb_type	0	original version
		1	version with increased speed of software and faster access to ticks from FEB
		2	version which supports high-speed RAM recording, specifically aggregate T1 or G.703 data capture
	•	3	version which also supports INTERVIEW 7200 and 7700 TURBOs
unsigned char	is_turbo	0	unit is not <i>TURBO</i> unit it <i>TURBO</i>
unsigned char	xdram_rev_num		XDRAM revision number
unsigned char	xdram_present	0 1	XDRAM board is not present XDRAM board is present
unsigned long	xdram_lo_addr		low end of memory range
unsigned long	xdram_hi_addr		high end of memory range
unsigned long	spare1		reserved/undefined
unsigned long	spare2		reserved/undefined
unsigned long	spare3		reserved/undefined
unsigned long	spare4		reserved/undefined
unsigned long	spare5		reserved/undefined
unsigned long	spare6		reserved/undefined
unsigned long	spare7		reserved/undefined
unsigned long	spare8		reserved/undefined
unsigned long	spare9		reserved/undefined
unsigned long	sw version	1997 - 1997 -	software version **
unsigned long	_ fw_version		firmware version†††
<u>Structure Nan</u>	<u>ne:</u> mpm_info		Structure containing information on specific MPI board. Instance of this structure for each MPN board is contained in array named unit_config.mpm. Declared as type extern struct.
unsigned char	rev_num		MPM revision number
unsigned char	present	0 1	specific MPM board (of four) not present specific MPM board (of four) present
unsigned long	lo_addr		low end of memory range
unsigned long	hi_addr		high end of memory range

Table 66-1 (continued)

†††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 displayf (or tracef):

displayf(" %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.

# INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

		• • • • • • • • • • • •
a pressent i som han han han er pressentation standarder ander som kannander at en som her som som og atternet		
, na an a garage out d'an de <b>rma sins o</b> ut de conserva sourt		

# 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 *rmt\_getc*, *rmt\_getl*, and *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 *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 *rmt\_getc*, *rmt\_getl*, or *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) *rmt\_flushi* is executed. "Unlock" the input queue with *rmt\_unlock*. *rmt\_unlock*, *rmt\_flushi*, and *rmt\_flusho* are automatically executed whenever the INTERVIEW returns to Program mode.

**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.

Similarly, when the programmer wants to send data out the remote port, he calls *rmt\_putc*, *rmt\_puts*, or *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 that the input queue is ready to accept more data.

67-2

Туре	Variable	Value (hex/decimal) Meaning	
extern event	rmt_break	True when a break (NU framing error) is receiv through the remote po Setup configured for e monitor mode.	ved rt. Line
extern event	rmt_input_not_empty	True when remote inputransitions from empty empty. Beginning to r characters. Line Setu configured for emulate monitor mode.	to not eceive p
extern event	rmt_input_almost_full	True when the remote input-queue transitions less than 3/4 full to 3/4 data is being put into t Line Setup configured emulate or monitor mo	s from 4 full as the queue for
extern event	rmt_input_overflow	True when remote inputransitions from not fur At this point, the older data in the queue is di to make room for new coming in the remote Setup configured for e monitor mode.	ll to full. st existin scarded data port. Lir
extern event	rmt_input_almost_empt	y True when the remote input-queue transitions more than 1/4 full to 1 data is being taken ou queue. Line Setup co for emulate or monitor	s from 1/4 full as t of the nfigured
extern event	rmt_input_empty	True when remote inp transitions from not en empty. All characters been read or discarde Setup configured for e monitor mode.	mpty to s have ed. Line
	rmt_output_empty	transmitted. Line Set configured for emulate monitor mode.	mpty to ut to the tup

Table 67-1 Remote Port I/O Variables

# 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**

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  $\bigcirc$  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 L 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_getl(data, 20);

displayf("\n%u characters read:\n%.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  $rmt\_getl$  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.

**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\_gets* (or *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 [S] key is pressed, 4000 characters will be read from the remote port, put into an array called *data*, displayed on the screen (until a NULL is encountered—see %s in *tracef* routine, Section 61), and written to a file named *echo\_time*. This is the program that might be run to receive the file transmitted in the *rmt\_putb* example.

#define FILE\_LENGTH 4000
#define FILENAME "FD1/usr/echo\_time"
#include <stdio.h>
#include <trace\_buf.h>
extern struct trace\_buf 11\_trbuf;
FILE \* stream\_ptr;
size\_t n;
unsigned char data [FILE\_LENGTH];
int 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(&l1_trbuf, "%d characters read: \n%s\n\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:
   Ł
   display_prompt ("Press C to get next character.
   rmt_lock();
   rmt_flushi();
   }
   CONDITIONS: KEYBOARD "cC"
   ACTIONS:
   int character:
   character = rmt_getc(0);
   if(character == -1)
       display prompt("No character available.
   else
       displayf("%c", character);
   }
```

");

");

## rmt lock

# <u>Synopsis</u>

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  $rmt_getl$  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  $\Box$ . The next line of data in the input queue will be removed and put in the array named *data*.

```
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_getl(data, 20);

displayf("\n%u characters read:\n%.20s\n", number, data);

}
```

# 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  $\square$  (or **reserve**).

```
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_getl(data, 20);

displayf("\n%u characters read:\n%.20s\n", number, data);

}

CONDITIONS: KEYBOARD "uU"

ACTIONS:

{

rmt_unlock();

}
```

# (B) Output Routines

Use the following routines to transmit data through the remote port.

#### rmt putc

#### **Synopsis**

extern int rmt\_putc(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\_putc* 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\_putc* (*rmt\_puts* or *rmt\_putb*) in one test, do the same in all tests.

## <u>Returns</u>

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  $\Box$ . 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.

```
{
unsigned char data [] = "((FOX)) Se";
unsigned char character;
int i, length, error;
}
LAYER: 1
     STATE: transmit characters
        CONDITIONS: ENTER STATE
        ACTIONS:
         ł
         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_putc(character, 0);
            if(error == -1)
                 display_prompt("No space available in output queue.
                                                                                          ");
            else
                 displayf ("%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.

#### <u>Returns</u>

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 S 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.

```
unsigned char data [] = "((FOX)) S_{R}";
int count, length;
LAYER: 1
     STATE: transmit_string
        CONDITIONS: ENTER_STATE
        ACTIONS:
         display_prompt("Press S to transmit string.
         length = sizeof(data) - 1;
        }
        CONDITIONS: KEYBOARD "sS"
        ACTIONS:
         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**

{

}

```
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\_putb* (*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.

#### <u>Example</u>

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 T key, the contents of the array will be transmitted and displayed.

#define FILE\_LENGTH 4000
#define FILENAME "FD1/usr/echo\_time"
#include <stdio.h>
extern struct trace\_buf.h>
extern struct trace\_buf l1\_trbuf;
FILE \* stream\_ptr;
size\_t n;
unsigned char data [FILE\_LENGTH];
unsigned char size [FILE\_LENGTH+100];
int count;

LAYER: 1

{

STATE: transmit\_string CONDITIONS: ENTER\_STATE ACTIONS:

> if((stream\_ptr = fopen(FILENAME, "r")) == 0) display\_prompt("Cannot open file.");

```
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_putb(data, FILE_LENGTH, 1);
 if(count != FILE_LENGTH)
    display f("Could not output entire string. n");
 sprintf(size, "%d characters transmitted: %%.%dH", count, count);
 tracef(&11_trbuf, size, data);
 tracef(\&l1\_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.

#### <u>Returns</u>

*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 *rmt\_putc*. Notice that as the program enters the first state, the output queue is flushed.

```
{
unsigned char data [] = "((FOX))";
unsigned char character'
int i, length, error;
LAYER: 1
     STATE: transmit_a_character
        CONDITIONS: ENTER STATE
        ACTIONS:
        ł
         rmt_flusho();
         display_prompt ("Press C to transmit character.
         length = sizeof(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);
            }
        }
```

#### rmt suspendo

#### **Synopsis**

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

rmt\_suspendo returns a zero when transmitting is successfully suspended. Otherwise, it returns a non-zero value.

Example

{

3

When the INTERVIEW receives an X-OFF as a signal to stop sending data, it will suspend transmissions from the remote port.

```
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:
        ł
         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

ł

}

When the INTERVIEW receives an X-ON as a signal to send data, it will resume transmissions from the remote port.

```
int character;
LAYER: 1
     STATE: resume output
        CONDITIONS: ENTER STATE
        ACTIONS:
        ł
        rmt_lock();
       TIMEOUT RESTART ck_input 0.001
       CONDITIONS: TIMEOUT check input
       ACTIONS:
        ł
        character = rmt_getc(1);
        if(character == 0x11)
           rmt_resumeo();
```

TIMEOUT ck\_input RESTART 0.001

# rmt send break

#### **Synopsis**

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\_putc, rmt\_puts* or *rmt\_putb*) 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.

```
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

displayf("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.

#### <u>Returns</u>

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:
{
displayf("Bits = %d", rmt_set_bits(7));
}
```

# rmt\_set\_parity

#### <u>Synopsis</u>

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

#### <u>Synopsis</u>

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.

# <u>Inputs</u>

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.



Figure 67-1 Null-modem cable connections.

#### <u>Returns</u>

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:
{
    displayf("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:

{

displayf("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.

```
LAYER: 1
```

```
STATE: current_parameters
CONDITIONS: ENTER_STATE
ACTIONS:
{
    displayf("Baud = %d Bits = %d ", rmt_get_baud_rate(), rmt_get_bits());
}
```

# rmt\_get\_parity

**Synopsis** 

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.

<u>Example</u>

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.

```
LAYER: 1

STATE: current_parameters

CONDITIONS: ENTER_STATE

ACTIONS:

{

displayf("Baud = %d Bits = %d Parity = %d", rmt_get_baud_rate(),

rmt_get_bits(), rmt_get_parity());

}
```

# rmt\_get\_mode

**Synopsis** 

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:

> displayf("Baud = %d Bits = %d Parity = %d Mode = %d", rmt\_get\_baud\_rate(), rmt\_get\_bits(), rmt\_get\_parity(), rmt\_get\_mode());

.

# 68 AUX Port I/O

Bit Number	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Transmitter's AUX Port Lead Configuration	0	I	I	1	I	I	I	I/C	0	0	0	0	0	0	0	0
Pin Number	16	14	12	10	8	6	4	2	15	13	11	9	7	5	3	1
Bit will be used for	С	U	U	U	U	U	U	С	D	D	D	D	D	D	D	D
Bit Number	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Receiver's AUX Port Lead Configuration	I/C	1	I	1	I	I	I	0	I	I	I	1	Ι	Ι	Ι	I
Pin Number	16	14	12	10	8	6	4	2	15	13	11	9	7	5	3	1
					0 - 10 U 10 D D	Inj Inj Co Da										

# INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

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.

Туре	Variable	Meaning
extern fast_event	aux_change	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.
extern volatile const unsigned short	curr_aux_value	Each bit designates a different lead. A bit-value of one
	no ha serie de la constante de La constante de la constante de La constante de la constante de	indicates a given lead is on. When value of <i>curr_aux_value</i> is exclusive ored (^) with <i>prev_aux_value</i> , result indicates those leads whose status has
		changed. Updated when
		aux_change comes true. Line
		Setup configured for emulate or monitor mode.
extern volatile const unsigned short	prev_aux_value	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.

## Table 68-1 AUX Port I/O Variables

# 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

#### <u>Synopsis</u>

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(0xfeff); }

## set\_aux\_ctl\_leads

## <u>Synopsis</u>

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).

## <u>Example</u>

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).

```
LAYER: 1
STATE: set_control_leads
CONDITIONS: ENTER_STATE
ACTIONS:
{
set_aux_ctl_leads(0x0100);
}
```

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:

LAYER: 1

STATE: set\_control\_leads CONDITIONS: ENTER\_STATE ACTIONS: { set\_aux\_ctl\_leads(0x8000); }

## write aux

## <u>Synopsis</u>

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:

```
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 \leq 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^{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(0xfeff); 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: { displayf("Transmission %d received: %04x Press spacebar to send ACK. \n", count, curr); } NEXT\_STATE: send\_ack CONDITIONS: {count > 10} ACTIONS: { "); display\_prompt("End of test. } STATE: send\_ack CONDITIONS: KEYBOARD " " ACTIONS: {  $if(count \le 10)$ { 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 *prev\_aux\_value* with *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 *aux\_change* is sufficient to signal and to acknowledge transmission. The value of *prev\_aux\_value* does not have to be checked.

## set\_aux\_reg

#### <u>Synopsis</u>

extern void set\_aux\_reg(reg\_value\_word);
unsigned short reg\_value\_word;

#### Description

The CIO chip may be reconfigured by the user via the set\_aux\_reg routine.

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).

## 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 Z8036 Z-CIO/Z8536 CIO Counter/Timer and Parallel I/O Unit Technical Manual, March 1982.

## <u>Example</u>

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 *reg\_value\_word*. Port B's input/output is reflected in the most-significant byte of *reg\_value\_word*.

**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):

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); } INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

# **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.

Туре	Variable	Value (hex/decimal)	Meaning
Structure Nar	<u>ne:</u> keyboard		Declared as type <i>extern struct</i> . Declared automatically if program KEYBOARD condition is used. Updated by <i>keyboard_new_key</i> event variable. Reference the structure variable as follows: <i>keyboard.value</i> .
char	value		ASCII value of key just executed.
Structure Nar	<u>me:</u> tm		Structure of time of day. Declared as type <i>extern struct</i> . Reference a structure variable as follows: <i>tm.tm_sec</i> .
int	tm_sec	0-36/0-59	Seconds after the minute. Not currently updated; always set to -1.
int	tm_min	0-36/0-59	Minutes after the hour.
int	tm_hour	0-17/0-23	Hours since midnight.
int	tm_mday	1-1/1-31	Day of month.
nt	tm_mon	0-b/0-11	Months since January.
nt	tm_year		Years since 1900.
nt	tm_wday	0-6	Days since Sunday. Not currently updated; always set to -1.
int	tm_yday	0-16d/0-365	Days since January 1. Not currently updated; always set to -1.
int	tm_isdst	n da sina Tata si	Daylight Savings Time flag. Not currently updated; always set to -1.
Structure Na	<u>me:</u> crnt_tm		Structure of current time of day. Updated every minute. Declared as type extern struct tm.
Structure Na	<u>me:</u> prev_tm		Structure of previous time of day, one minute ago. Declared as type extern struct tm.
<u>Structure Na</u>	<u>me:</u> flag_struct		Structure of a flag. Declared as type <i>struct</i> . Declared automatically if a program flag is used. Program flags assigned to structure as follows: <i>struct flag_struct flag_</i> name. Reference a structure variable as follows: <i>flag_</i> name. <i>curren</i>
unsigned short	prev		When converting a flag action to C, the translat compares <i>prev</i> with <i>current</i> to determine whether flag has changed. Then <i>prev</i> is update to <i>current</i> and <i>flag_name_change</i> is signaled.
unsigned short	current		This value of flag is acted on directly by program actions.
unsigned short	old		When converting a flag condition to C, the translator compares <i>old</i> with <i>current</i> to determine whether true condition is new (transitional). After program logic has examine flag, <i>old</i> is updated to <i>prev</i> .

# Table 69-1 Structure Fields—Other Library Tools

Туре	Variable	Value (hex/decimal)	Meaning
<u>Structure Nam</u>	<u>ne:</u> timeout		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.
unsigned long	event_id		Four bytes of a 6-byte timeout, containing the segment number and offset. <i>Timeout_name_stop</i> routines set this event id to zero.
unsigned short	event_id_uid	<ul> <li>A state of the sta</li></ul>	Two bytes of a 6-byte timeout which uniquely identify (uld) the timeout. Do not try to assign a value to this variable.

## Table 69-1 (continued)

# 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 even, [22], [F1-F8 softkeys, and [2008].

## (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.

	Variable	Value (hex/dec	imal) Meaning
extern fast_event	fevar_time_of_day		True once per minute. Line Setup configured for emulate or monitor mode.
extern event	flag_name_change		This event must be signaled by the program itself; it is not "external" to the program. The translator signals this event as part of the FLAG increment, decrement, or set action. Line Setup configured for emulate of monitor mode.
extern event	timeout_ <i>name</i> _expired		This event must be signaled by the program itself. It is not "external" to the C program. The translator signals this event as part of the <i>timeout_restart_action</i> routine. Line Setup configured for emulate or monitor mode.
extern event	signal_name		True when the named signal is the argument in a <i>signal</i> routine Spreadsheet-token equivalent is ON_SIGNAL name. Line Setup configured for emulate or monitor mode.
extern volatile unsigned short	crnt_time_of_day	0-937/0-2359	Current time is stored in this variable. Updated as soon as time changes. Line Setup configured for emulate or monitor mode.
extern volatile unsigned short	prev_time_of_day	0-937/0-2359	Current time is stored in this variable. Updated when time changes, but only after iogic has had a chance to compare current and previous time. Line Setup configured for emulate o monitor mode.
extern volatile const unsigned char	crnt_date_of_day	1-1f/1-31	Current date is stored in this variable. Updated as soon as date changes. Line Setup configured for emulate or monitor mode.
extern volatile const unsigned char	prev_date_of_day		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 o monitor mode.
extern fast_event	keyboard_new_key		True when any ASCII key is pressed. Line Setup configure for emulate or monitor mode.

# Table 69-2 Other Library Variables

69 Other Library Tools

extern fast_event	keyboard_new_any_key		True when any key is pressed The only exceptions are Em, [@@], [F1-F8 softkeys, and M Line Setup configured for
			emulate or monitor mode.
extern volatile unsigned short	keyboard_any_key		Identifies last key or key-combination executed. I Setup configured for emulate
			monitor mode.
		0-7f/0-127 80-17f/	ASCII keys
		128-383	not used
		100/004	Field entry keys:
		180/384	HEX
		181/385	NOT EQUAL BIT MASK
		182/386 183/387	FLAG
		184/388	DONT
		185/389	CLEAR
		186/390	SHIFT - CLEAR
		187/391	
		18a/394	TAB
		18b/395	SHIFT - TAB
		18c/396	SHIFT - FLAG
		18d/397	CTRL-FLAG
		18e/398	SHIFT BIT MASK
		18f/399	CTRL - BIT MASK
		190/400	SHIFT - CONT
		191/401	CTRL - (BONT)
		192/402	
		193/403	
		194/404	
		195/405	
		196/406 10d/269	CTRL - HEX
		100/209 1a0/416	CTRL - RETURN
		1a1/417	SHIFT - RETURN
		101/71/	
		100/410	Editing Keypad Keys:
		1a2/418 1a3/419	INE RISERTI CHAR
		1a4/420	DELETE
		1a5/421	DELETE
		1a6/422	
		1a7/423	
		1a8/424	

# Table 69-2 (continued)

69-5

# INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

Туре	Variable	Value (hex/c	decimal) Meaning
ante programme de la com	(keyboard_any_ke	y continued)	Editing Keypad Keys (cont):
		1a9/425	
		1aa/426	
		1ab/427	
		1ac/428	
		1ad/429	
		140,420	
			Utility Keys:
		160/432	PROGRAM
		1b1/433	RUN
		1b2/434	LOAD
		1b3/435	SAVE
		164/436	FREEZE
		1b5/437	RECORD
		167/439	PRINT
		168/440	XEQ
		1ba/442	
		1bb/443	
		1bc/444	
		1bd/445	SHIFT - PROBAM
		1be/446	CTRL - PROGRAM
		1bf/447	SHIFT - RUN
		1c0/448	CTRL - RUN
		1c1/449	SHIFT LOAD
		1c2/450	CTRL - LOAD
		1c3/451	SHIFT - SAVE
		1c4/452	CTRL - SAVE
		1c5/453	SHIFT - FREEZE
		1c6/454	
		1c7/455	
		1c8/456	CTRL - RECORD
		1c9/457	
• • • • • • • • • •			
		1ca/458	
		1cb/459	
		1cc/460	
		1cd/461	
		1ce/462	CTRL - XEQ
			Pure Cursor Keys:
		1d0/464	①
		1d1/465	
		1d2/466	
		1d3/467	
		1d4/468	
		1d5/469	
		1d6/470	HOME

## Table 69-2 (continued)

(keyboard\_any\_key variable continued on next page)

Туре	Variable	Value (hex/de	ecimal) Meaning
	(keyboard_any_key o	continued)	Pure Cursor Keys (cont)
		1d7/471	SHIFT - HONE
		1d8/472	CTRL - HOME
		1d9/473	 
		1da/474	
		1db/475	
		1dc/476	J
		1dd/477	SHIFT-+
		1de/478	
			Cursor Keypad Keys:
		1e0/480	
		1e1/481	MARK
		1e2/482	SHIFT MARK
		1e3/483	
		1e4/484	SHIFT-BRK
		1e5/485	BACK
		1e6/486	FREY
		1e7/487	ROLL
		1e8/488	PAGE
		1e9/489	CTRL -MARK
		1ea/490	
		1eb/491	
		1ec/492	GTRL - BACK
		1ed/493	SHIFT - (ROLL FWO
		1ee/494	CTRL - ROLL FWD
		1ef/495	SHIFT - PAGE
		1f0/496	CTRL - PAGE
		1f1/497	SHIFT - NEXT PAGE
		1f2/498	CTRL - MEXT
		1f3/499	
		1f4/500	CTRL - RUB OVT
			Other Keys:
		1f5/501	
		1f6/502	
		440 450 4	
		1f8/504	
		1f9/505	<b>CTRL</b> -5
		1fa/506	
		1fb/507	
		1fc/508	
		188/392	<b>CTRL</b> -9
			<b>CTRL</b> -Ø
		1fd/509	<b>CTRL</b>

Table 69-2 (continued)

× 1.42

# 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.

{ 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: ł timeout\_stop\_action(&timeout\_example); timeout\_restart\_action(&timeout\_example, 2000, timeout\_example\_isp); } CONDITIONS: { timeout\_example\_expired } ACTIONS: ALARM

Here is a version of the program that accomplishes the same result without an action to signal the timeout event:

{ struct timeout { unsigned long event\_id; unsigned short event\_id\_uid; }; struct timeout timeout\_example; extern void sound\_alarm(); } LAYER: 2 STATE: example\_of\_timeout CONDITIONS: FRAME\_SENT ACTIONS: { timeout\_stop\_action(&timeout\_example); timeout\_restart\_action(&timeout\_example, 2000, sound\_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.

#### <u>Example</u>

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

#### <u>Synopsis</u>

extern 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.

<u>Inputs</u>

See index.

**Returns** 

See index.

Example

See index.

## load\_program

## **Synopsis**

extern 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.

## <u>Inputs</u>

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.

LAYER: 3

STATE: test\_26 CONDITIONS: ENTER\_STATE ACTIONS: SEND DIAG CONDITIONS: RCV CLEAR\_CONF ACTIONS: TRACE "Test\_26 passed" { load\_program("FD1/usr/module\_19"); }

## 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 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

,

{	
, #include <stdio.h></stdio.h>	
const char data [] = "((FOX)) \n";	
int key;	
void write fox()	
<pre>{</pre>	
FILE * stream_ptr;	
size_t n;	
lock(&key);	
<pre>if((stream_ptr = fopen("FD2/usr/buff01", "a")) == 0)     display_prompt("Cannot open file.</pre>	");
else	
display_prompt("File opened.	");
n = fwrite(data, 1, sizeof(data)-1, stream_ptr);	
pos_cursor(1,0);	
if(n != (sizeof(data)-1))	
displayf("Write error.	\n");
else	
displayf("Write completed.	\ <i>n</i> ");
if (fclose (stream_ptr) != 0)	the encourted ").
displayf("Either file is already closed, or close canno else	t be executed. ");
displayf("File closed.	");
unlock(&key);	
1	
a the second	
LAYER: 1	
TEST: a	
STATE: write and signal	
CONDITIONS: RECEIVE STRING "THE QUICK E	BROWN FOX"
ACTIONS: SIGNAL xyz	
{	
write_fox();	
}	-
TEST: b	
STATE: write_only	
CONDITIONS: ON_SIGNAL xyz	
ACTIONS:	
1 (mailed for ())	
write_fox();	
<pre></pre>	

## 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

<u>Synopsis</u>

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.

<u>Example</u>

## 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

<u>Synopsis</u>

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.

#### <u>Example</u>

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 **mean** 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

#### <u>Synopsis</u>

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.

## <u>Example</u>

## LAYER: 2

STATE: example
 CONDITIONS: KEYBOARD " "
 ACTIONS:
 {
 suspend\_rcrd\_play();
 }
}

## send key

<u>Synopsis</u>

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.

#### <u>Inputs</u>

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 I. As soon as a bad BCC is detected on the DTE side, the data display will change to the Layer 2 Protocol Trace screen.

```
{
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
        Ł
        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: we 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.

Туре	Variable	Value (hex/decimal)	Meaning
Structure Nan	<u>ne:</u> xmit_list	ro au abbo C m	tructure of a transmit list for x21_transmit_call butine. Declared as type struct. Declared utomatically if a softkey-entered ALL_SETUP_SEND action is taken. Reference nember variables of the structure as follows: mit_list.string_length
unsigned char *	string	- F	ointer to the location of the transmit string—the ransmit string is declared separately
unsigned short	string_length	0- <i>ffff</i> /0-65535 le	ength of the transmit string

Table 70-1 X.21 Structures

# 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*.

Туре	Variable	Value (hex/decima	l) Meaning
extern fast_event	fevar_eia_changed	for	e when the status change an EIA lead. Line Setup
			nfigured for emulate or nitor mode.
extern const volatile unsigned	short current_eia_leads	2 B ( 4 I-v 8 R-1	valid RS-232 equivalent is SQ) alid (RI) valid (DSR) valid (DSR)
		40/64 I (( 80/128 C 100/256 R ( 200/512 UA	CTS) (RTS) RD)
			TD) value in this list indicates
	na se du se se se l'utilité de la sea l'anne de la secondaria	wh Wh cui eq the val	ich lead(s) you care about een anded (&) with rrent_eia_leads, the result uals zero if the lead is on ( mask if the lead is off). idity checks, the result of ding with current_eia_lead.
		eq	uals the mask for valid (or o for invalid).
	·	<u>Ex</u>	amples:
		{ /	ATE: c_on_and_valid f ((current_eia_leads & 0xt 1) sound_alarm();    }
	t en legene en tradiciparita <del>da</del>	{ h	ATE: c_off_and_valid f ((current_eia_leads & 0x0 0x81) sound_alarm(); }
		sta Cic	te: This variable will store atus if (1) internal or exter ock is supplied, (2) EIA lea e enabled on FEB Setup, a
		(3 up co	) fevar_eia_changed has dated the leads. Line Set nfigured for emulate or onitor mode.
extern const volatile unsigned	short previous_eia_leads	cu wh afi co lea	me values as <u>rrent_eia_leads</u> . Updated ien leads change, but only ier logic has had a chance mpare current and previou ads. Line Setup configured r emulate or monitor mode

Table 70-2 X.21 Variables

## (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 in 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:

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

## (C) Detect Change and Current Status

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

CONDITIONS:

```
(fevar_eia_changed && (((current_eia_leads ^ previous_eia_leads) & 0x40) == 0x40) &&
((current_eia_leads & 0x44) == 4))
```

This example approximates the translator's version of the spreadsheet-token condition LEADS | 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."

## ctl\_eia

#### <u>Synopsis</u>

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 controlleads. Which lead you control depends on your emulation mode. When the Line Setup menu shows **Mode:** EMULATE DOE, 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 **EXELATE DOE**. 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

## <u>Synopsis</u>

extern void x21\_idle\_action(character);
unsigned char character;

## Description

Only for format SYNC, the  $x21\_idle\_action$  routine allows you to change the idle-line condition applied by the INTERVIEW. A LEADS R BELLS action, for example, requires the  $x21\_transmit\_call$  routine in addition to  $x21\_idle\_action$ .

## Inputs

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

## Example

To signal an incoming call, you would use the  $x21\_transmit\_call$  routine to send the sync pattern. Then you would use the  $x21\_idle\_action$  routine to send an idle string of *bells*:

## LAYER: 1

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

}

## x21\_transmit\_call

## **Synopsis**

extern void x21\_transmit\_call(count, struct\_send\_string\_ptr, xmit\_tag); unsigned short count; struct xmit\_list
{
 char \* string\_ptr;
 unsigned short string\_length;
 };
struct xmit\_list \* struct\_send\_string\_ptr;
unsigned short xmit\_tag;

## Description

The x21\_transmit\_call routine sends a specified data string in call-setup mode. The softkey equivalent of this routine is the CALL\_SETUP\_SEND action.

## **Inputs**

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

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

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

## Example

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

## LAYER: 1

}

```
{
unsigned char syncs [] = {0x16,0x16};
unsigned char number [] = "1234567";
unsigned char end [] = "+";
struct xmit_list
{
    unsigned char * string;
    unsigned short string_length;
};
```

struct xmit\_list send\_string [] = {&syncs[0], 2, &number[0], sizeof(number) - 1, &end[0], 1};

STATE: send CONDITIONS: RECEIVE STRING "[5]++" 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 " $\S$  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*.

#### <u>Inputs</u>

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

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

## Example

Assume you are emulating a DTE. To indicate a clear confirmation, enter the following spreadsheet program:
```
LAYER: 1

STATE:

CONDITIONS: KEYBOARD * "

ACTIONS:

{

set_tcr_b (0xf3, 0x08);

ctl_eia(0xff, 0x04);

}
```

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

### (B) Phase

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

### enter\_call\_phase

#### **Synopsis**

extern void enter\_call\_phase();

#### Description

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

### Example

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

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

### enter\_data\_phase

### **Synopsis**

extern void enter\_data\_phase();

#### Description

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

### Example

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

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

70-10

# 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

 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

Туре	Variable	Value	(hex/decimal)	Meaning
Structure Name	: send_frame_s	tructure		Structure of a frame in X.25. Declared as type struct. Declared automatically if a softkey-entered SEND action is taken. Program
				frames assigned to structure as follows: struct
				send_frame_structure name. Reference a structure variable as follows: name.bcc_type. If values in the frame structure are not initialized by the user, they default to 0. You may initialize the values when the structure is declared:
				struct send frame_structure name = $\{1, 1, 1, 0, 1, 1, 3, 0x71, 3, 0\};$
unsigned char	addr_type		)   2	command response other
unsigned char	frame_type		(The codes for fr cvd_frame_type.	ame_type are the same as for the X.25-variable )
unsigned char	nr_type	(	)	auto
			2	value received ns plus 1
		-	3	last nr sent
unsigned char	ns type		0	auto
			l se	skip
			2 3	last nr received value
unsigned char	p f type	(	D	0
-			1	1 loopback
unsigned sher	has turns	-	- D	
unsigned char	bcc_type		1	default (bad bcc) good bcc
			2	bad bcc
			3	abort
unsigned char	addr_value		1 3	to DCE to DTE
unsigned char	cntrl_byte		(actual value of t	the control byte)
unsigned char	nr_value	an an the	0-7 (MOD 8)	if nr_type = 1
unsigned char	ns_value		0–7 (MOD 8)	if ns_type = 3
- The management of th				

Table 71-1X.25 Layer 2 Structures

Туре	Variable Value (hex	(decimal) Meaning
extern event	dte_frame	True when a DTE frame is detected. Line Setup configured for emulate or monitor mode.
extern event	dce_frame	True when a DCE frame is detected. Line Setup configured for emulate or monitor mode.
extern event	dte_good_bcc	True when a good BCC is calculated for a DTE frame. Line Setup configured for emulate or monitor mode.
extern event	dce_good_bcc	True when a good BCC is calculated for a DCE frame. Line Setup configured for emulate or monitor mode.
extern event	dte_bad_bcc	True when a bad BCC is calculated for a DTE frame. Line Setup configured for emulate or monitor mode.
extern event	dce_bad_bcc	True when a bad BCC is calculated for a DCE frame. Line Setup configured for emulate or monitor mode.
extern event	dte_abort	True when an abort is detected for a DTE frame. Line Setup configured for emulate or monitor mode.
extern event	dce_abort	True when an abort is detected for a DCE frame. Line Setup configured for emulate or monitor mode.
extern event	rcvd_frame	True when a frame is received Line Setup configured for emulate mode only.
extern event	invalid_frame	True when an invalid frame is detected. Line Setup configured for emulate mode only.
extern event	I2_T1	True when the T1 timeout-time has expired. Line Setup configured for emulate mode only.
extern event	bcc_error	True when a BCC error is detected. Line Setup configured for emulate mode only.
extern event	nr_error	True when an N(R) error is detected in a received INFO o supervisory frame. Line Setur configured for emulate mode only.
extern event	ns_error	True when an N(S) error is detected in a received INFO

Table 71-2X.25 Layer 2 Variables

Туре	an a baile a finish a sa an	Variable	Value (hex/dec	cimal) Meaning
xtern event	an a	frame_sent	n na an	True when frame is passed down to Layer 1. Line Setup configured for emulate mode only.
xtern volatile con	-	m_frame_addr	1 3	to DCE to DTE
				Line Setup configured for emulate or monitor mode.
xtern volatile con	st unsigned char	m_frame_type	(same as rcvd_ configured for	frame_type—Line Setup emulate or monitor mode)
xtern volatile con	st unsigned char	m_frame_cntrl_byte_1	(actual value o configured for	f control byte—Line Setup emulate or monitor mode)
xtern volatile con	st unsigned char	m_frame_pf	0 10/16	pf=0 pf=1
				Line Setup configured for emulate or monitor mode.
xtern volatile con	st unsigned char	m_frame_bcc_type	1 2 3	good bad abort
				Line Setup configured for emulate or monitor mode.
xtern volatile con		rcvd_frame_addr	1 3	to DCE to DTE
				Line Setup configured for emulate mode only.
extern volatile con	st unsigned char	rcvd_frame_type	0 1	info rr
			5 9 d/13 2f/47 6f/111	rnr rej srej sabm sabme
			43/67 f/15 f/15	disc dm sarm
			63/99 87/135 ff/255 ff/255	ua frmr other unknown
			11/200	Line Setup configured for emulate mode only.
extern volatile con	st unsigned char	rcvd_frame_cntrl_byte_1	(actual value of configured for	of control byte—Line Setup remulate mode only)
extern volatile con	st <b>unsigne</b> d char	rcvd_frame_pf	0 10/16	pf=0 pf=1
n ann an 1990. Seachar Seachair				Line Setup configured for emulate mode only.
extern volatile con	st unsigned char	rcvd_frame_bcc_type	1 2 3	good bad abort
				Line Setup configured for emulate mode only.
extern volatile con		rcvd_frame_nr	0-7 (MOD 8)	Line Setup configured for emulate mode only.

# Table 71-2 (continued)

71-4

Туре	Variable	Value (hex/dec	imal) Meaning
extern volatile const unsigned char	rcvd_frame_ns	0-7 (MOD 8)	Line Setup configured for emulate mode only.
extern volatile unsigned short	rcvd_frame_buff_seg		Inter-layer message buffer number (actually, an IAPX-286 segment number) in a received frame. This segment number can be converted to a pointer by shifting it left 16 bits. Line Setup configured for emulate mode only.
extern volatile unsigned short	rcvd_frame_sdu_offset		Offset to where the service da unit begins in an inter-layer message buffer in a received
			frame. Add to buffer segment number (converted to pointer) to point to first byte in frame. Line Setup configured for emulate mode only.
extern volatile unsigned short	rcvd_frame_sdu_size		Size of service data unit in a received frame. Line Setup configured for emulate mode only.
extern volatile unsigned short	l2_current_window_edge		When equal to upper edge, window is full; when equal to lower edge, window is empty; when not equal to upper edge, window is not full; and when no equal to lower edge, window is not empty. Line Setup configured for emulate mode only.
extern volatile unsigned short	12_lower_window_edge		see I2_current_window_edge
extern volatile unsigned short	l2_upper_window_edge		see I2_current_window_edge
extern volatile unsigned short	l2_resend_edge		When resend edge is not equa to lower window edge, there is more to resend; when resend edge is equal to lower window edge, there is no more to resend. Line Setup configured for emulate mode only.
extern unsigned char	l2_enhance	0 1 4 5 8 9 12/18	normal reverse low reverse low blink reverse blink blink low Line Setup configured for
			emulate or monitor mode.
extern unsigned char	l2_suppress	0 1	off on Line Setup configured for emulate or monitor mode.

Table 71-2 (continued)

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

CONDITIONS: DTE BDBCC ACTIONS: COUNTER bad\_bcc INC CONDITIONS: DCE BDBCC ACTIONS: COUNTER bad\_bcc INC

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

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

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

CONDITIONS:

{ dte\_frame }

2. Emulate mode only. Some events may be detected in emulate mode only. The event variables are rcvd\_frame, invalid\_frame, l2\_T1, bcc\_error, nr\_error, ns\_error, and frame\_sent.

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

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

rcvd\_frame && (rcvd\_frame\_type == 0)
}

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

#### CONDITIONS:

{ rcvd\_frame }

Error detecting may be accomplished via *bcc\_error*, *nr\_error*, *ns\_error*, and *invalid\_frame*. These variables equate to the softkey tokens bearing similar names.

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

### (B) Status Variables

Status variables are those in Table 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:

}

rcvd\_frame && (rcvd\_frame\_type == 0)

 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\_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 71-2 containing a *rcvd* prefix are valid in emulate mode only: *rcvd\_frame\_addr*, *rcvd\_frame\_type*, *rcvd\_frame\_cntrl\_byte\_l*, *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;
}</pre>

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	12_current_window_edge == 12_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	12_resend_edge == 12_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:
{
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(l2\_enhance == 1) l2\_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

<u>Synopsis</u>

extern void 12\_give\_data();

#### Description

The l2\_give\_data routine takes an interlayer message buffer associated with a received INFO frame, changes the SDU offset to point to higher-level data, and sends a DL\_DATA IND primitive up to Layer 3 along with a reference to this buffer. The softkey equivalent of this routine is the GV\_DATA action on the Protocol Spreadsheet.

### Example

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

LAYER: 2 STATE: datalink CONDITIONS: DL\_CONNECT REQ ACTIONS: DL\_CONNECT CONF CONDITIONS: DL\_DATA REQ ACTIONS: SEND INFO "((DL\_DATA))" CONDITIONS: RCV INFO ACTIONS: { l2\_give\_data(); }

### (B) Transmit

### resend\_frame

### **Synopsis**

extern void resend\_frame(pf, first\_or\_next); unsigned char pf; unsigned char first\_or\_next;

### Description

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

### **Inputs**

The first parameter is the value of the P/F bit in the frame. It may be set to either 0, 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 (PROTOCL) RSET\_NS action on the Protocol Spreadsheet.

<u>Example</u>

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

#### <u>Synopsis</u>

extern void send\_frame(il\_buffer\_number, relay\_baton, data\_start\_offset, transmit\_frame\_ptr); unsigned short il\_buffer\_number; unsigned short relay\_baton; unsigned short data\_start\_offset; struct send\_frame\_structure {

unsigned char addr\_type; unsigned char frame\_type; unsigned char nr\_type; unsigned char ns\_type; unsigned char p\_f\_type; unsigned char bcc\_type; unsigned char addr\_value; unsigned char cntrl\_byte; unsigned char nr\_value; unsigned char ns\_value; };

struct send\_frame\_structure \* transmit\_frame\_ptr;

#### Description

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

#### <u>Inputs</u>

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.

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

INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

71-14

# 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:

dte\_packet && (m\_packet\_type == 0)

# INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

Туре	Variable Value	e (hex/decimal)	Meaning
Structure Name	<u>e:</u> send_packet_structure		Structure of a packet in X.25. Declared as type struct. Declared automatically if a softkey-entered SEND action is taken. Program packets assigned to structure as follows: struct send_packet_structure name. Reference a structure variable as follows: name.q_bit. If values in the frame structure are not initialized by the user, they default to 0. You may initialize the values when the structure is declared: struct send_packet_structure name = $\{2, 0x13, 0x13, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 2, &afacilities_string[0], 0, 0\};$
unsigned char	path_num	0–8 fe/254	path number use path number of last received packet
unsigned char	packet_type	(The codes for p m_packet_type.)	acket_type are the same as for the X.25-variable
unsigned char	packet_type_byte	(actual value of t	he packet type byte)
unsigned char	m_bit	0 1	m = 0 m = 1
unsigned char	d_bit	0 40/64	d = 0 d = 1
unsigned char	q_blt	0 80/128	q = 0 q = 1
unsigned char	pr_type	0 1 2 3	auto value received ps plus 1 last pr sent
unsigned char	ps_type	0 1 2 3	auto skip received pr value
unsigned char	pr value	0-7 (MOD 8)	if pr_type = 1
unsigned char	ps_value	0-7 (MOD 8)	if ps_type = 3
unsigned char	cause		byte—see Figure 36–15)
unsigned char	diag_flag	0	diagnostic field not present diagnostic field is present
unsigned char	diag	(value of diagno: pp. 237-8)	stic byte—consult CCITT Recommendation X.25,
unsigned char	spare	0 - 2 an di	reserved space
unsigned char	facilities_len	0-ff/0-255	length of the facilities field
char *	facilities		pointer to the location of the facilities field—the facilities field is declared separately
unsigned short	data_len		reserved for future use
char *	data		reserved for future use

# Table 72-1 X.25 Layer 3 Structures

Туре	Variable	Value (hex/dec	imal) Meaning
extern event	dte_packet		True when a DTE packet is detected. Line Setup configured for emulate or monitor mode.
extern event	dce_packet		True when a DCE packet is detected. Line Setup configured for emulate or monitor mode.
extern event	rcvd_packet		True when a packet is received from Layer 2. Line Setup configured for emulate mode only.
extern event	invalid_packet		True when an invalid packet is detected. Line Setup configured for emulate mode only.
extern event	pr_error	•	True when an P(R) error is detected in a data or supervisory packet. Line Setup configured for emulate mode only.
extern event	ps_error		True when an P(S) error is detected in a data packet. Line Setup configured for emulate mode only.
extern event	packet_sent		True when a packet has been passed down to Layer 2. Line Setup configured for emulate mode only.
extern volatile unsigned short	m_packet_lcn	0-fff/0-4095	Logical channel number. Line Setup configured for emulate o monitor mode.
extern volatile unsigned char	m_packet_lcn_grp	0-f/0-15	Logical channel group number. Line Setup configured for emulate or monitor mode.
extern volatile const unsigned char	m_packet_q	0 80/128	q=0 q=1 Line Setup configured for emulate or monitor mode.
extern volatile const unsigned char	m_packet_d	0 40/64	d=0 d=1 Line Setup configured for emulate or monitor mode.
extern volatile const unsigned char	m_packet_m	0 10/16	m=0 m=1 Line Setup configured for emulate or monitor mode.
extern volatile const unsigned char	m_packet_pr	0-7 (MOD 8)	Line Setup configured for emulate or monitor mode.
extern volatile const unsigned char	m_packet_ps	0-7 (MOD 8)	Line Setup configured for emulate or monitor mode.

Table 72-2X.25 Layer 3 Variables

Туре	Variable	Value (hex/dec	imal) Meaning
extern volatile const unsigned char	m_packet_cause	(same as rovd_pkt_cause—Line Setup configured for emulate or monitor mod	
extern volatile const unsigned char	m_packet_diag_code	(same as rcvd_pkt_diagn—Line Setup configured for emulate or monitor mode)	
extern volatile const unsigned char	m_packet_type_byte	(actual value of packet type byte—Line Set configured for emulate or monitor mode)	
extern volatile const unsigned char	m_packet_type	b/11 f/15 13/19 17/23 0 23/35 27/39 1 5 9 1b/27 1f/31 fb/251 ff/255 f1/241 f3/243 f7/247 11/17 11/17	call call acc clear clear conf data int int conf rr rnr rej reset reset conf restart restart conf diag reg reg conf other pkt unknown pkt Line Setup configured for emulate or monitor mode.
extern volatile unsigned short	rcvd_pkt_lcn	0-fff/0-4095	Logical channel number in a received packet. Line Setup configured for emulate mode only.
extern volatile const unsigned char	rcvd_pkt_q	0 80/128	q=0 q=1 Line Setup configured for emulate mode only.
extern volatile const unsigned char	rcvd_pkt_d	0 40/64	d=0 d=1 Line Setup configured for emulate mode only.
extern volatile const unsigned char	rcvd_pkt_m	0 10/16	m=0 m=1 Line Setup configured for emulate mode only.
extern volatile const unsigned char	rcvd_pkt_pr	0-7 (MOD 8)	Line Setup configured for emulate mode only.
extern volatile const unsigned char	rcvd_pkt_ps	0-7 (MOD 8)	Line Setup configured for emulate mode only.
extern volatile const unsigned char	rcvd_pkt_cause	(see Figure 36 emulate mode	6–15—Line Setup configured for only)
extern volatile const unsigned char	rcvd_pkt_diagn	(consult CCIT pp.237–8—Lin mode only)	T Recommendation X.25, e Setup configured for emulate

# Table 72-2 (continued)

.

Туре	Variable	Value (he>	(decimal)	Meaning
extern volatile const unsigned char	rcvd_pkt_type_byte	(actual va configure	alue of packet ty d for emulate n	vpe byte—Line Setup node only)
extern volatile const unsigned char	rcvd_packet_type	b/11	call	
		f/15	call acc	
		13/19	clear	
		17/23	clear cor	nf
		0 23/35	data	
		23/35	int conf	
		1	rr	
		5	rnr	
		9	rej	
		1b/27	reset	
		1f/31	reset co	nr
		fb/251 ff/255	restart restart c	onf
		f1/241	diag	
		f3/243	reg	
		f7/247	reg conf	
		11/17	other pk	ALL ALL ALL ALL ALL
		11/17	unknown	•
				up configured for mode only.
extern volatile unsigned short	m_packet_buff_seg	]	number	er message buffer (actually, an IAPX-28 : number). This
			segment	number can be
				ed to a pointer by
				t left 16 bits. Line onfigured for emulate mode.
extern volatile unsigned short	m_packet_info_seg	le per en angradate. Nga	Same as	s m_packet_buff_seg.
extern volatile unsigned short	m_packet_sdu_offs	set	unit beg	o where the service d ins in an inter-layer
				e buffer. Add to
				ouff_seg (converted to to point to first
				header byte. Line Set
			configur	ed for emulate or
			monitor	mode.
extern volatile unsigned short	m_packet_info_off	set		o where the packet tion begins, excluding
				der. Add to
				buff_seg (converted to
				to point to packet da tup configured for
				or monitor mode.
extern volatile unsigned short	m_packet_length		header.	of the packet, includir Line Setup configure late or monitor mode
extern volatile unsigned short	m_packet_info_len	gth	informa	of the packet tion, excluding the Line Setup configure
				late or monitor mode
		-5		

Table 72-2 (continued)

# INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

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

# Table 72-2 (continued)

Туре	Variable	Value (hex/d	lecimal)	Meaning
	nar rcvd_device_path		receive LCN ar parame Level S	umber connecting ad packet to particular nd particular set of call eters on the X.25 Packet setup screen. Line Setup ired for emulate mode
extern unsigned char	13_enhance	0 1 4 5 8 9	normal reverse low reverse blink reverse	e low
		12/18		e or monitor mode.
extern unsigned char	l3_suppress	0 1		etup configured for e or monitor mode.

Table 72-2 (continued)

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:

ł

}

{

ł

rcvd\_packet

Error detecting may be accomplished via *pr\_error*, *ps\_error*, and *invalid\_packet*. These variables equate to the softkey tokens bearing similar names.

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

### (B) Status Variables

Status variables are those in Table 72-2 that do not include *event* in the Type column. Their associated event variables guarantee that they are updated and tested.

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

CONDITIONS:

{

}

rcvd\_packet && (rcvd\_packet\_type == 0)

 Packet characteristics. All status variables in Table 72-2 containing an m\_ prefix are valid in either emulate or monitor mode: m\_packet\_lcn, m\_packet\_lcn\_grp, m\_packet\_q, m\_packet\_d, m\_packet\_m, m\_packet\_pr, m\_packet\_ps, m\_packet\_cause, m\_packet\_diag\_code, m\_packet\_type, and m\_packet\_type\_byte.

All status variables in Table 72-2 containing a *rcvd*\_prefix are valid in emulate mode only: *rcvd\_pkt\_lcn*, *rcvd\_pkt\_q*, *rcvd\_pkt\_d*, *rcvd\_pkt\_m*, *rcvd\_pkt\_pr*, *rcvd\_pkt\_ps*, *rcvd\_pkt\_cause*, *rcvd\_pkt\_diagn*, *rcvd\_pkt\_type*, and *rcvd\_pkt\_type\_byte*.

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

- 2. Packet buffers. Packets are passed up to Layer 3 from Layer 2 in IL message buffers. Several variables provide the user with access to the information in the packet that is located beyond the packet-type byte. These variables are rcvd\_pkt\_buff\_seg, m\_packet\_buff\_seg, rcvd\_pkt\_sdu\_offset, m\_packet\_sdu\_offset, rcvd\_pkt\_length, and m\_packet\_length. See Section 63.1 for a more detailed discussion of the buffer components to which these variables refer.
- 3. Pointers. Two variables, rcvd\_pkt\_info\_ptr and m\_packet\_info\_ptr, point to the first byte beyond the packet header. You may move these pointers to

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

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

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

### (C) Controlling Protocol Trace Display

To enhance or suppress particular packets on the Layer 3 Protocol Trace screen in emulate or monitor mode, assign a coded value to *l3\_enhance* or *l3\_suppress*. The values are listed in Table 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

<u>Synopsis</u>

extern void l3\_give\_data();

### Description

The *l3\_give\_data* routine takes an interlayer message buffer associated with a received data packet, changes the SDU offset to point to higher-level data, and sends an N\_DATA IND primitive up to Layer 4 along with a reference to this buffer. The softkey equivalent of this routine is the GV\_DATA action on the Protocol Spreadsheet.

### Example

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

```
LAYER: 2
```

```
STATE: datalink
CONDITIONS: DL_CONNECT REQ
ACTIONS: DL_CONNECT CONF
CONDITIONS: DL_DATA REQ
ACTIONS: SEND INFO "((DL_DATA))"
CONDITIONS: RCV INFO
ACTIONS: GIVE_DATA
```

LAYER: 3

STATE: pass\_data\_up CONDITIONS: N\_CONNECT REQ ACTIONS: SEND CALL CONDITIONS: RCV CALL\_CONF ACTIONS: N\_CONNECT IND CONDITIONS: N\_DATA REQ ACTIONS: SEND DATA "((N\_DATA))" CONDITIONS: RCV DATA ACTIONS:

13 give\_data();

LAYER: 4

}

STATE: establish\_link CONDITIONS: ENTER\_STATE ACTIONS: N\_CONNECT REQ

### (B) Transmit

### 13\_clear\_path

### **Synopsis**

extern void 13\_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
```

### I3\_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.

### <u>Returns</u>

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 &&(13_more_to_resend(0xfe) != 0)
}
ACTIONS: RESEND_NEXT
CONDITIONS:
{
packet_sent &&(13_more_to_resend(0xfe) == 0)
}
NEXT_STATE: xfer
```

### I3\_window\_full

### <u>Synopsis</u>

extern unsigned char l3\_window\_full(path\_number);
unsigned char path\_number;

#### Description

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

### Inputs

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

### <u>Returns</u>

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 && (13\_window\_full(0xfe) != 0)

} ACTIONS: SEND DATA "(FOX)"

### I3\_window\_empty

### **Synopsis**

extern unsigned char l3\_window\_empty(path\_number); unsigned char path\_number;

#### Description

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

#### Inputs

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

#### Returns

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

### Example

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

{	
extern event timeout_ack_expired; extern event rcvd packet;	
}	
LAYER: 3	
STATE: check window	
CONDITIONS: PACKET_SENT	
ACTIONS: TIMEOUT ack RESTART CONDITIONS:	
{ rcvd_packet	
}	
ACTIONS: TIMEOUT ack STOP	
CONDITIONS:	
{ timeout_ack_expired && (13_window_emp	oty(Oxfe) = 0
}	·)(-·)(-)
ACTIONS: RESEND FIRST	

### resend\_packet

### <u>Synopsis</u>

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

/\* 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 (PROTOCL) RSTPRPS action on the Protocol Spreadsheet.

#### Inputs

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

#### Example

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

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

### send\_packet

#### **Synopsis**

# 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.

### <u>Inputs</u>

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.

{ static unsigned short il\_buffer\_number; static unsigned short relay\_baton; static unsigned short data\_start\_offset;

struct send\_packet\_structure

{ unsigned char path num; unsigned char packet\_type; unsigned char packet\_type\_byte; unsigned char m\_bit; unsigned char d\_bit; unsigned char q\_bit; unsigned char pr type; unsigned char ps type; unsigned char pr\_value; unsigned char ps\_value; unsigned char cause; unsigned char diag\_flag unsigned char diag; unsigned char cntrl byte; unsigned char facilities len; char \* facilities; unsigned short data len; char \* data;

};

static char transmit\_string [] = "((FOX))";

static char facilities\_string [] = "0,0,0,0,4,4,0,3,4,0,0,7";

struct send\_packet\_structure transmit\_packet =  $\{0, 0x13, 0x13, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, (sizeof(facilities_string)-1), & facilities_string[0], 0, 0\};$ 

#### LAYER: 2

STATE: datalink CONDITIONS: DL\_CONNECT REQ ACTIONS: DL\_CONNECT CONF CONDITIONS: DL\_DATA REQ ACTIONS: SEND INFO "((DL\_DATA))" CONDITIONS: RCV INFO ACTIONS: GIVE\_DATA

LAYER: 3

STATE: send\_a\_packet CONDITIONS: KEYBOARD " "

ACTIONS:

{

\_get\_il\_msg\_buff(&il\_buffer\_number, &relay\_baton);

\_start\_il\_buff\_list(il\_buffer\_number,&data\_start\_offset);

\_insert\_il\_buff\_list\_cnt(il\_buffer\_number, data\_start\_offset, &transmit\_string[0], (sizeof(transmit\_string) - 1));

send\_packet(il\_buffer\_number, relay\_baton, data\_start\_offset, &transmit\_packet);
}

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

static char facilities\_string [] = {1, 1, 4, 1, 0x41, 0x45, 0x03, 0x43, 7, 7}; --then transmit\_packet would look like this--

# 73 SDLC Library

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

The variables and routines approximate SDLC Layer 2 spreadsheet-generated conditions and actions. Refer to Section 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

 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

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

### Table 73-1 SDLC Structures

- 1908 bes fill serve of smart server sergisk – spold servel) is Herein en i transverment er en server de server al en er en er
## Table 73-2 SDLC Variables

true 2

Туре	Variable	Value (hex/decimal)	Meaning
extern event	dte_frame	detecto configu	hen a DTE frame is ed. Line Setup red for emulate or r mode.
extern event	dce_frame	detecto configu	hen a DCE frame is ed. Line Setup red for emulate or r mode.
extern event	dte_good_bcc	calcula Line Se	hen a good BCC is ted for a DTE frame. etup configured for e or monitor mode.
extern event	dce_good_bcc	calcula Line Se	hen a good BCC is ted for a DCE frame. atup configured for e or monitor mode.
extern event	dte_bad_bcc	calcula Line Se	hen a bad BCC is ted for a DTE frame. etup configured for e or monitor mode.
extern event	dce_bad_bcc	calcula Line Se	hen a bad BCC is ted for a DCE frame. stup configured for e or monitor mode.
extern event	dte_abort	for a C configu	then an abort is detected TE frame. Line Setup ured for emulate or r mode.
extern event	dce_abort	for a D configu	when an abort is detected DCE frame. Line Setup ared for emulate or r mode.
extern event	rcvd_frame	Line Se	when a frame is received etup configured for e mode only.
extern event	invalid_frame	detect	when an invalid frame is ed. Line Setup ured for emulate mode
extern event	I2_T1	has ex	when the T1 timeout-time pired. Line Setup ured for emulate mode
extern event	bcc_error	detect	vhen a BCC error is ed. Line Setup ured for emulate mode
extern event		True v detect superv	when an N(R) error is ed in a received INFO o visory frame. Line Setuj ured for emulate mode
extern event		detect frame	when an N(S) error is ted in a received INFO . Line Setup configured nulate mode only.

	Variable	Value (hex/de	cimal) Mea	ning
	frame_sent	na na serie de la companya de la com La companya de la comp La companya de la comp	True when fram down to Layer configured for only.	1. Line Setup
tern volatile const unsigned char	m_frame_addr	00-ff/0-255	Line Setup cor emulate or mo	
ttern volatile const unsigned char	m_frame_type		frame_type—Lir r_emulate_or_mor	
ttern volatile const unsigned char	m_frame_cntrl_byte_1		of control byte—I r emulate or mor	
ktern volatile const unsigned char	m_frame_pf	0 10/16	pf=0 pf=1	
			Line Setup cor emulate or mo	
ktern volatile const unsigned char	m_frame_bcc_type	1 2 3	good bad abort	
		And the state	Line Setup cor emulate or mo	
tern volatile const unsigned char	rcvd_frame_addr	00-ff/0-255	Line Setup cor emulate mode	
ktern volatile const unsigned char	rcvd_frame_type	0 1	info rr	
		5 9	rnr rej	
		d/13	srej	
an a Quest terms for a constant on attactor for example do		3 7	ui rim	
		7 f/15	sim dm	
		23/35 43/67	up disc	
		43/67 63/99	rd ua	
		83/131 87/135	snrm frmr	
		af/175 c7/199 cf/207	xid cfgr snrme	
		e3/227	test	
		ef/239 b/11	bcn Ipda	
		ff/240 ff/240	other unknown	
and and a second se The second se			Line Setup co emulate mode	
xtern volatile const unsigned char	rcvd_frame_cntrl_byte_1	(actual value configured fo	of control byte— or emulate mode	Line Setup only)
xtern volatile const unsigned char	rcvd_frame_pf	0 10/16	pf=0 pf=1	
			Line Setup co	nfigured for

# Table 73-2 (continued)

	Variable	Value (hex/dec	cimal) Meaning
extern volatile const unsigned char	rcvd_frame_bcc_type	1 2 3	good bad abort
			Line Setup configured for emulate mode only.
extern volatile const unsigned char	rcvd_frame_nr	0-7 (MOD 8)	Line Setup configured for emulate mode only.
extern volatile const unsigned char	rcvd_frame_ns	0-7 (MOD 8)	Line Setup configured for emulate mode only.
extern volatile unsigned short	rcvd_frame_buff_seg	na contrata Las Martas Martas sogares Martas sogares	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.
	rcvd_frame_sdu_offset		Offset to where the service data unit begins in an inter-layer message buffer in a received frame. Add to buffer segment number (converted to pointer) to point to first byte in frame. Line Setup configured for emulate mode only.
extern volatile unsigned short	rcvd_frame_sdu_size		Size of service data unit in a received frame. Line Setup configured for emulate mode only.
extern volatile unsigned short	l2_current_window_edge	fones of a con Nori CNS Nori CNS Nori CNS Nori CNS Nori CNS	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. see l2_current_window_edge
extern volatile unsigned short	12_lower_window_edge		
	I2_resend_edge	n shatisni cini setos sasti bash	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

Table 73-2 (continued)

73-5

Type (1.00	Variable	Value (he	x/decimal)	Meaning
extern unsigned char	l2_enhance	anne a star second at at second the second star in the second star second star second star second star second s	normal	na Magnagana (Desta menun) nagesatak gapatan anggapan (anggapan m
		рар в селеко <b>1</b> меря 4	reverse low	
		5	reverse blink	low
• Stational and stations Station and stations		9 12/18	reverse blink lov	
				tup configured for or monitor mode.
extern unsigned char	l2_suppress	0	off	
		en l'encendra de la composición Composición		tup configured for or monitor mode.
			के में के समय के लिए	

### Table 73-2 (continued)

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

CONDITIONS: DTE BDBCC ACTIONS: COUNTER bad\_bcc INC CONDITIONS: DCE BDBCC ACTIONS: COUNTER bad\_bcc INC

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

dte\_frame && (m\_frame\_type == 0)

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

### CONDITIONS:

### dte\_frame

3

}

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

no second visit e not gal ni Emar windlow visite, intera is ano ta recente, within version (to ta catrat to inversivation) and Tame is an enter to frame type. When the translator converts a RCV INFO condition into C, it will include two C variables, one event variable and one status variable:

rcvd\_frame && (rcvd\_frame\_type == 0)

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

CONDITIONS:

{ rcvd\_frame }

}

Error detecting may be accomplished via *bcc\_error*, *nr\_error*, *ns\_error*, and *invalid\_frame*. These variables equate to the softkey tokens bearing similar names.

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

### (B) Status Variables

Status variables are those in Table 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:

CONDITIONS:

ł

rcvd\_frame && (rcvd\_frame\_type == 0)

 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).

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

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	12_current_window_edge != 12_lower_window_edge
MORE_TO_RESEND	12_resend_edge != 12_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 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:
```

ł

{

{ l2\_enhance = 0; }

or an ACTIONS block:

CONDITIONS: RCV INFO ACTIONS:

```
if(l2_enhance == 1)
    l2_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 l2\_give\_data.*"

(A) Receive

12 give data

<u>Synopsis</u>

extern void 12\_give\_data();

### Description

The *l2\_give\_data* routine takes takes an interlayer message buffer associated with a received INFO frame, changes the SDU offset to point to higher-level data, and sends a DL\_DATA IND primitive up to Layer 3 along with a reference to this buffer. The softkey equivalent of this routine is the GIVE\_DATA action on the Protocol Spreadsheet.

### Example

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

### LAYER: 2

```
STATE: datalink

CONDITIONS: DL_CONNECT REQ

ACTIONS: DL_CONNECT CONF

CONDITIONS: DL_DATA REQ

ACTIONS: SEND INFO "((DL_DATA))"

CONDITIONS: RCV INFO

ACTIONS:

{

I2_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).

The Refrectation calles taxes an merioper message build associated with a received INFO frame, changes the SDH affast to print to higher-levil data, and rends a DL\_DATA IND primitive op to Layer 3 along with a reference to the buffer. The softkey equivalent of this routing is the GWE\_DATA action to the Protocol Spreadshare.

### 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
           HE WE HOTING PARTICIPATION
ACTIONS:
  {
   resend_frame(1, 0);
  }
  NEXT_STATE: recover
STATE: recover
  CONDITIONS: FRAME SENT
      MORE TO RESEND
  ACTIONS:
  {
   resend_frame(1,1);
  3
  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 (PROTOCL) RSET NS action on the Protocol Spreadsheet.

### Example

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

```
LAYER: 2
```

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

### send frame

### **Svnopsis**

extern void send\_frame(il\_buffer\_number, relay\_baton, data\_start\_offset, transmit\_frame\_ptr); unsigned short il\_buffer\_number; unsigned short relay\_baton; unsigned short data\_start\_offset; struct send\_frame\_structure {

unsigned char addr\_type; unsigned char frame\_type; unsigned char nr\_type; unsigned char ns\_type; unsigned char p\_f\_type; unsigned char bcc\_type; unsigned char addr\_value; unsigned char cntrl\_byte; unsigned char nr\_value; unsigned char ns\_value; };

struct send\_frame\_structure \* transmit\_frame\_ptr;

### Description

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

### Inputs

The first parameter is the interlayer message buffer number. See Section 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.

```
ł
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:
         ł
         _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);
        }
```

nan Arek burkobistar 5 Skali Kalikura mekada busha busha bushan. Sea Seaten 19 A(A), 199 ar-buloporalaci (193) mutema

ens soccas parametri A sur su britan (di mater su heist due buller while dhe secil specation is build, paracerci - Secilie de constanti (di 1)

a and promptor with a from the analysis beginning of the onlysis to the cost of the second data and the factors of the second size in the cost of the second size in the second size in

a a contra contractor e a percirar la carte fracto directaria to be sont - Con a destatabilità el sont<u>a p</u>arte per conte per Calte Conte Contes.

#### a a suceda

.

hend am Imra Frenke europaints; i hoannac fins idensolet chill b pares 2000 (mins (Flag. Beal

> ા ગામ મથ્યાપ્રકાર આવ્યા છે. કેમ્પ્રેસ્ટ પ્રચાલ છે. ૧૯૨૬ પ્રકારણાં આ પ્રેસ્ટ પ્રાયં કરે છે. ૧૯૨૬ સાર પ્રકારણાં કેમ્પ્રેસ્ટ કેસ્ટ કરે હતા કે સ્ટેટ છે. ૧૯૧૯ સ્ટેસ્ટ ટ્રીસ્ટ્ર જેટલ સ્ટ્રોસ્ટ ટ્રીસ્ટ

.

Califyriae Control (2007) [http:// califyriae cheffformal. califyriae E Control (2007) califyriae E Control (2007) califyriae E Control (2007) califyriae Cheff (2007)

atenasis ann'n<u>a</u>tearata, constituted industria (f<sub>a</sub>ina elle) Netifica estanata en elemente antenes a la miglica (fai

.

nst ja brog GUA 43. Na Andrahaman

ALC: CARL

Alexand a strate was not as the fact that the state of th

الا الذي 20 من 2000 من 2011 المالين 100 من المركز العام المحاج المحاج المالي 2014 من التي المركز المركز المحاج المحاجبة المحاج المح

(High Grightensk, safte order atta state atta (High College) (High College)

terreterik istika war orden anden bere ver triter isterreterik

### 74 SNA Library

# 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.

t sideltal/

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)

INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

Туре	Variable	Value	(hex/decir	mal)	Meaning
extern volatile unsigned short	m_packet_length			the tra reques Setup	of the packet, including nsmission and t/response headers. Lin configured for emulate of r mode.
nn as an inn francol Spraalshee			ol eldellev	s ¢nin	o er saldshuv
extern volatile const unsigned char	<u>Transmission Header:</u> m_packet_fid_type	nogol n o			<u>t Identification Type:</u> TH 10 bytes
		2010		FID1;	TH 10 bytes
		2 3	a nana ara ang	FID3;	TH 6 bytes TH 2 bytes
		4 f/15			TH 26 bytes TH 26 bytes
			at noissian	Line S	etup configured for e or monitor mode.
extern volatile const unsigned short	m_packet_daf	0-fff	f/0-65535		ation address field-2
			091	bytes in FID0 and FID1; in FID2. Line Setup con for emulate or monitor	2. Line Setup configured
extern volatile const unsigned short Str	m_packet_def	0-fff	f10-65535	bytes; configu	ation element field—2 FID4 only. Line Setup ured for emulate or r mode.
extern volatile const unsigned long	m_packet_dsaf		fffff/		ation subarea address
		sisci per	94967295	Line S	4 bytes; FID4 only. etup configured for te or monitor mode.
yine smluma yë bistoqqat :					
extern volatile const unsigned char	ି m_packet_Isid େରିବର୍ତ୍ତର ର	(act	ual value vte)	FID3 o SSCP-	<u>Session Identification:</u> nly -PU session -LU session
			• • · · · ·	Reserv	/ed
					etup configured for te or monitor mode.
	m nackat oaf	00.4	f/0-255	1345365	address field—2 bytes ir
extern volatile const unsigned short	m_packet_oaf		170-200 A A A	FID0 a	and FID1; 1 byte in FID2
			i sakisinev	Line S emula	etup configured for te or monitor mode.
extern volatile const unsigned short	m_packet_oef	00-1	110-255		element field—2 bytes; only. Line Setup
life transs with a FID2				config	ured for emulate or
stell The condition should look	alacy O exercised radius	101 .56L			or mode.
extern volatile const unsigned long	m_packet_osaf		ffffff) 294967295	bytes: config	subarea address field— ; FID4 only. Line Setup ured for emulate or or mode.

Table 74-1 SNA Variables†

† Refer to Table 73-2 for SDLC variables.

74 SNA Library

Туре	Variable	Value (hex/	decimal)	Meaning
en internationale dis Autority i	Transmission Header (co	ontiued) :		
extern volatile unsigned char *	th_ptr	ba ba Lood D	header contair configu	r for the transmission r; begins at the byte ning FID type. Line Setup ured for emulate or or mode.
none measure to solution	Request/Response Head	er:		
extern volatile const unsigned char	m_packet_ru_category	0		<u>st/Response Unit:</u> on Management Data
	usiael abarrio	20/32 40/64 60/96	Data F	rk Control (NC) Flow Control (DFC) n Control (SC)
en al agrif independent of the off Service (1196-105-2005) d 01.9%	na kisena harazi di Ginti 1 diguna set	angus ta 22 Album di 23 a		etup configured for e or monitor mode.
extern volatile const unsigned char	m_packet_fi			t Indicator:
una geleget algebrais and anna Ana Anna geleget anna anna airte	est Trans of Monor 1955 markenilist	cie 0(2, 6062) 2018 - 1111 - 2 2019 - 2019	In LU- header NC, or format	lata without header in RU LU frame, indicates r follows the RH. In SC, r DFC RU, indicates a tted RU beginning with a st code.
				etup configured for e or monitor mode.
extern volatile const unsigned char	m_packet_rri	0 80/128	Reque reques respor	
				etup configured for te or monitor mode.
extern volatile unsigned char	m_packet_rti	0 10/16	positiv	<u>nse Type Indicator:</u> e response ve response
in the second least in the second	anter an instant		Line S	etup configured for te or monitor mode.
extern volatile unsigned char	m_packet_sdi	0 4	sense	Data Indicator: data not included data included
			Line S	etup configured for te or monitor mode.
extern volatile unsigned char *	<b>rh_ptr</b> inno- ocigan		heade conta indica	r for the request/respons r; begins at the byte ining the request/respons tor. Line Setup configure nulate or monitor mode.
	Request/Response Unit			
extern volatile unsigned char *	ru_ptr		unit; l the u	er for the request/respons begins at the first byte in hit. Line Setup configured nulate or monitor mode.

C

- Info frame characteristics. Most status variables in Table 74-1 contain an m\_ prefix, indicating that they are valid in emulate or monitor mode. Some variables are associated with the transmission header: m\_packet\_fid\_type, m\_packet\_daf, m\_packet\_def, m\_packet\_dsaf, m\_packet\_lsid, m\_packet\_oaf, m\_packet\_oef, and m\_packet\_osaf. Other variables are associated with the request/response header: m\_packet\_ru\_category, m\_packet\_fi, m\_packet\_rri, m\_packet\_rti, and m\_packet\_sdi.
- 2. Pointers. There are three pointers to SNA fields. th\_ptr points to first byte of the transmission header, rh\_ptr points to the first byte of request/response header, and ru\_ptr points to the start of the request/response unit.

# (C) Controlling Protocol Trace Display

To enhance or suppress particular packets on the Layer 2 Protocol Trace screen in emulate or monitor mode, assign a coded value to *l2\_enhance* or *l2\_suppress*. The values are listed in Table 73-2. To assign a value to either of these variables, place the statement in an ACTIONS block. For example, display only Info frames with FID2 transmission headers. Of these, display frames with sense data in reverse-video.

```
CONDITIONS:

{

dte_frame && (m_frame_type == 0) && (m_packet_fid_type != 2)

}

ACTIONS:

{

12_suppress = 1;

}

CONDITIONS:

{

dte_frame && (m_frame_type == 0) && (m_packet_fid_type == 2) && (m_packet_sdi == 4)

}

ACTIONS:

{

12_enhance = 1;

}
```

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

CONDITIONS:

dte\_frame && (m\_frame\_type == 0) && (m\_packet\_fid\_type != 2) && (l2\_suppress == 0) } ACTIONS:

```
{
12_suppress = 1;
```

or an ACTIONS block:

```
CONDITIONS:

{

dte_frame && (m_frame_type == 0) && (m_packet_fid_type != 2)

}

ACTIONS:

{

if(l2_suppress == 0)

l2_suppress = 1;

}
```

# 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.

### INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

75 DDCMP Library

计分词 前上的神经方法

# 75 DDCMP Library

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

#### 75.1 Structures

There are no extern structures associated exclusively with DDCMP.

a name familier:



TO BE LEAR AND DEPARTURE.

# 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:

extern fast event fevar bd bcc2 td;
and dependences of the second
LAYER: 1
STATE: count all bad dte bccs
CONDITIONS: DTE BAD BCC
ACTIONS: COUNTER t bdbcc INC
CONDITIONS:
지수는 비행 방법을 가지 않는 것이 없는 것이 없다.
fevar_bd_bcc2_td
성장 이는 것은 것을 다 귀엽다. 영화 방법을 받았다.
A DEVENIE A DEVITED & Laters INO

ACTIONS: COUNTER t\_bdbcc INC

# 75.3 Routines

There are no routines associated exclusively with DDCMP.

### INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

Туре	Variable	Value (hex/deci	mal) Meaning
extern fast_event	fevar_gd_bcc_rd		True when a good header BCC is received on RD. Line Setup configured for emulate or monitor mode.
extern fast_event Externation (Sticksong, Cotternation) Chife are the Promotal Statendahas Mana at Caver E.		parkára is torése nihris dre us ty parketat is sat	True when a good <i>header</i> BCC is received on TD. Line Setup configured for emulate or monitor mode.
extern fast_event	fevar_bd_bcc_rd		True when a bad <i>header</i> BCC is received on RD. Line Setup configured for emulate or monitor mode.
extern fast_event	fevar_bd_bcc_td	encente arctico en	True when a bad <i>header</i> BCC is received on TD. Line Setup configured for emulate or monitor mode.
extern fast_event	fevar_gd_bcc2_td		True when a good <i>data</i> BCC is received on TD. Line Setup configured for emulate or monitor mode.
extern fast_event id and brid asha Bad we hore postel of man menus and million to the	fevar_gd_bcc2_rd	ak bedata ik kadada n n seki katan saven 1973 neki katan ji 19	True when a good <i>data</i> BCC is received on RD. Line Setup configured for emulate or monitor mode.
extern fast_event			True when a bad <i>data</i> BCC is received on TD. Line Setup configured for emulate or monitor mode.
extern fast_event	fevar_bd_bcc2_rd	green salt <b>mage</b> s	True when a bad <i>data</i> BCC is received on RD. Line Setup configured for emulate or monitor mode.

### Table 75-1 DDCMP Variables

75.3 Rodines

There are no nothing associated encluded, why DDCMP.

# 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: , 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.

Туре	Var	iable	Value (hex/decimal)	Meaning
Structure Name:	xmit_	list		Structure of a transmit list for send_d_frame routine. Declared as type struct. Reference member variables of the structure as follows: xmit_list.string_length.
unsigned char *	string	ndr av fre	ning in the second s	pointer to the location of the transmit string-the transmit string is declared separately
unsigned short	string_le	angth	0-ffff10-65535	length of the transmit string

# Table 76-1

# 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 76-2 ISDN Variables

Type mass 5 123 area constituted by		Value (hex/decimal)	Meaning
extern event angow des fab	sang2 Houston <mark>d_dte_frame</mark> Vice of _4CC0 = 0.7 The Jerman Country of the second	detect Line S	when a DTE frame is ted on the D channel. tetup configured for te or monitor mode.
extern event	d_dce_frame	True v detec Line S	when a DCE frame is ted on the D channel. tetup configured for te or monitor mode.
extern event ການປະຊຸດັກ ນາຍ 10 ປາມມາກ ເຮັດກຸ ໃຫ້ເວີ ໂອະ ລາ	d_rcv_frame STETEELEELEELEELEE BELEELEELEELEELEELEELEELEELEELEE STETEELEE	size with start on the	when a frame is received D channel. Line Setup jurged for emulate mode

# 76.3 Routines

There are two routines associated with the ISDN\_D package: *send\_d\_frame* and *send\_d\_frame\_il*. Another ISDN routine, *set\_isdn\_speaker\_chan*, controls the speaker for either of the B channels. This routine is supplied by the ISDN Test Interface Module.

### (A) Transmit

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

### send\_d\_frame

### **Synopsis**

extern void send\_d\_frame(count, struct\_send\_string\_ptr, xmit\_tag); unsigned short count; struct xmit\_list
{
 unsigned char \* string\_ptr; unsigned short string\_length;
 }; struct xmit\_list \* struct\_send\_string\_ptr; unsigned short xmit\_tag;

### Description

The *send\_d\_frame* routine sends a specified string on the D channel with a user-determined BCC.

### Inputs

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

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

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

### Example

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

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

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

## send\_d\_frame\_il

### <u>Synopsis</u>

extern void send\_d\_frame\_il(il\_buffer\_number, relay\_baton, data\_start\_offset, transmit\_tag); unsigned short il\_buffer\_number; unsigned short relay\_baton; unsigned short data\_start\_offset; unsigned short transmit\_tag;

### Description

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

<u>Inputs</u>

The first parameter is the interlayer message buffer number.

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

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

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

### Example

Send the same text as in the example for send\_d\_frame. Refer to Section 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 [] = "01x0001004x000((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)); send\_d\_frame\_il(il\_buffer\_number, relay\_baton, data\_start\_offset, 1); }

### (B) Speaker Control

### set isdn speaker chan

### <u>Synopsis</u>

extern void set\_isdn\_speaker\_chan(selection);
unsigned short selection;

### Description

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

### Inputs

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

### Example

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

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

# INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

,

# 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

 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:

CONDITIONS:

}

dte\_bad\_bcc || dce\_bad\_bcc

ACTIONS: COUNTER bad\_bcc INC

### INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

### Table 77-1 LAPD Structures

Туре	Variable Valu	e (hex/decimal)	Meaning
Structure Nam	<u>ie:</u> 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. I values in the frame structure are not initialized by the user, they default to 0. You may initialize the
			values when the structure is declared: struct send_frame_structure name = {1, 1, 2, 0,
			0, 0, 1, 1, 1, 0, 0, 0;
unsigned char	sapi_type	1	no other value valid—indicates a value is given
unsigned char	tei_type	1	no other value valid—indicates a value is given
unsigned char	cr_type	0	0
		1	1 Jaanhaak
		2	loopback
unsigned char	frame_type	(The codes for fr rcvd_frame_type.	rame_type are the same as for the LAPD-variable .)
unsigned char	nr_type	0	auto
		1 2	value received ns plus 1
		3	last nr sent
unsigned char	ns_type	0	auto
		1	skip last nr received
		2 3	value
unsigned char	p f type	0	0
anaigned end	P_1_()P0		a <b>1</b> and a state state of the
		2	loopback
unsigned char	bcc_type	0	default (bad bcc)
		1	good bcc bad bcc
		2 3	abort
unsigned char	sapi_value	00-3f/0-63	
unsigned char	tei_value	00-7f/0-127	
unsigned char	cntrl_byte	(actual value of	the control byte)
unsigned char	nr_value	0-7 (MOD 8)	if nr_type = 1
unsigned char	ns_value	0-7 (MOD 8)	If ns_type = 3

77 LAPD Library

Туре	Variable Value (	hex/decimal) Meaning
extern event	dte_frame	True when a DTE frame is detected. Line Setup configured for emulate or monitor mode.
extern event	dce_frame	True when a DCE frame is detected. Line Setup configured for emulate or monitor mode.
extern event	dte_good_bcc	True when a good BCC is calculated for a DTE frame. Line Setup configured for emulate or monitor mode.
extern event	dce_good_bcc	True when a good BCC is calculated for a DCE frame. Line Setup configured for emulate or monitor mode.
extern event	dte_bad_bcc	True when a bad BCC is calculated for a DTE frame. Line Setup configured for emulate or monitor mode.
extern event	dce_bad_bcc	True when a bad BCC is calculated for a DCE frame. Line Setup configured for emulate or monitor mode.
extern event	dte_abort	True when an abort is detected for a DTE frame. Line Setup configured for emulate or monitor mode.
extern event	dce_abort	True when an abort is detected for a DCE frame. Line Setup configured for emulate or monitor mode.
extern event	rcvd_frame	True when a frame is received. Line Setup configured for emulate mode only.
extern event	invalid_frame	True when an invalid frame is detected. Line Setup configured for emulate mode only.
extern event	12_T1	True when the T1 timeout-time has expired. Line Setup configured for emulate mode only.
extern event	bcc_error	True when a BCC error is detected. Line Setup configured for emulate mode only.
extern event	nr_error	True when an N(R) error is detected in a received INFO or supervisory frame. Line Setup configured for emulate mode only.
extern event	ns_error	True when an N(S) error is detected in a received INFO frame. Line Setup configured for emulate mode only.

Table 77-2 LAPD Variables

Туре	Variable	Value (hex/dec	imal) Meaning	
extern event	frame_sent	an an an Arran Maria. An Arranga	True when frame is passed down to Layer 1. Line Setup configured for emulate mode only.	
extern volatile const unsigned cha	ar m_frame_addr_sapi	00-3f/0-63	Line Setup configured for emulate or monitor mode.	
extern volatile const unsigned ch	ar m_frame_addr_tei	00-7f/0-127	Line Setup configured for emulate or monitor mode.	
xtern volatile const unsigned ch	ar m_frame_addr_cr	0	0 1	
			Line Setup configured for emulate or monitor mode.	
xtern volatile const unsigned ch	ar m_frame_type		frame_type—Line Setup emulate or monitor mode)	
xtern volatile const unsigned ch	ar m_frame_cntrl_byte_1		f control byte—Line Setup emulate or monitor mode)	
xtern volatile const unsigned ch	ar m_frame_pf	0 10/16	pf=0 pf=1 Line Setup configured for emulate or monitor mode.	
xtern volatile const unsigned ch	ar m_frame_bcc_type	1 2 3	good. bad abort Line Setup configured for	
n an	•		emulate or monitor mode.	
xtern volatile const unsigned ch	ar m_frame_nr	0-7 (MOD 8)	Line Setup configured for emulate or monitor mode.	
xtern volatile const unsigned ch	ar m_frame_ns	0-7 (MOD 8)	Line Setup configured for emulate or monitor mode.	
xtern volatile const unsigned ch	ar rcvd_frame_addr_sapi	00-3f/0-63	Line Setup configured for emulate mode only.	
xtern volatile const unsigned ch	ar rcvd_frame_addr_tei	00-7f/0-127	Line Setup configured for emulate mode only.	
xtern volatile const unsigned ch	ar rcvd_frame_addr_cr	0 1	0	
		2	loopback Line Setup configured for emulate mode only.	
xtern volatile const unsigned ch	ar rcvd_frame_type	0 1 3 5 9 2f/37 6f/111 43/67 f/15 f/15 f/15 63/99 67/103 87/135 e7/224 ff/255	info rr ul rnr rej sabm sabme disc dm sarm ua slo frmr si1 other	
		ff/255	unknown Line Setup configured for emulate mode only.	

# Table 77-2 (continued)

77 LAPD Library

Туре	Variable	Value (hex/dec	cimal) Meaning
extern volatile const unsigned char	rcvd_frame_cntrl_byte_1		of control byte—Line Setup emulate mode only)
extern volatile const unsigned char	r rcvd_frame_pf	0 10/16	pf=0 pf=1
			Line Setup configured for emulate mode only.
extern volatile const unsigned cha	r rcvd_frame_bcc_type	1 2	good bad
		3	abort
			Line Setup configured for emulate mode only.
extern volatile const unsigned cha	r rcvd_frame_nr	0-7 (MOD 8)	Line Setup configured for emulate mode only.
extern volatile const unsigned cha	r rcvd_frame_ns	0-7 (MOD 8)	Line Setup configured for emulate mode only.
extern volatile unsigned short	rcvd_frame_buff_seg		Inter-layer message buffer number (actually, an IAPX-280 segment number) in a receive frame. This segment number can be converted to a pointer by shifting it left 16 bits. Line Setup configured for emulate mode only.
extern volatile unsigned short	rcvd_frame_sdu_offset		Offset to where the service da unit begins in an inter-layer message buffer in a received frame. Add to buffer segment number (converted to pointer to point to first byte in frame. Line Setup configured for emulate mode only.
extern volatile unsigned short	rcvd_frame_sdu_size		Size of service data unit in a received frame. Line Setup configured for emulate mode
extern volatile unsigned short	l2_current_window_edge		only. 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 n equal to lower edge, window is not empty. Line Setup configured for emulate mode only.
extern volatile unsigned short	12_lower_window_edge		see I2_current_window_edge
extern volatile unsigned short	12_upper_window_edge		see I2_current_window_edge
extern volatile unsigned short	i2_resend_edge		When resend edge is not equato lower window edge, there more to resend; when resend edge is equal to lower window edge, there is no more to resend. Line Setup configure for emulate mode only.

Table 77-2 (continued)

77-5

Туре	1. 人名波德尔 网络	Variable	Value (hex/	decimal)	Meaning
extern unsigned char		I2_enhance	0 1 4 5 8 9 12/18		blink , tup configured for
extern unsigned char		l2_suppress	0 1	off on Line Set	or monitor mode. tup configured for or monitor mode.

### Table 77-2 (continued)

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

CONDITIONS: DTE BDBCC ACTIONS: COUNTER bad\_bcc INC CONDITIONS: DCE BDBCC ACTIONS: COUNTER bad\_bcc INC

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

dte\_frame && (m\_frame\_type == 0)
}

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

CONDITIONS: {
 dte\_frame
}

2. Emulate mode only. Some events may be detected in emulate mode only. The event variables are rcvd\_frame, invalid\_frame, l2\_T1, bcc\_error, nr error, ns\_error, and frame\_sent.

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

When the user selects RCV on the first rack of softkeys for Layer 2 conditions, a second rack appears from which he must select a particular

frame type. When the translator converts a RCV INFO condition into C, it will include two C variables, one event variable and one status variable:

### rcvd\_frame && (rcvd\_frame\_type == 0)

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

CONDITIONS: { rcvd\_frame

\_

3

}

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:

{

rcvd\_frame && (rcvd\_frame\_type == 0)

 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\_1, 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\_1*, *rcvd\_frame\_pf*, *rcvd\_frame\_bcc\_type*, *rcvd\_frame\_nr*, and *rcvd\_frame\_ns*.

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

2. Frame buffers. As BOP frames are received, they are automatically placed in IL message buffers to be passed up the layers. Three emulate-mode variables provide the user with access to the information in the frame that is located beyond the control byte. These variables are rcvd\_frame\_buff\_seg, rcvd\_frame\_sdu\_offset, and rcvd\_frame\_sdu\_size. See Section 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 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 WINDOW EMPTY WINDOW NOT\_FULL WINDOW NOT\_EMPTY MORE\_TO\_RESEND NO MORE TO\_RESEND 12\_current\_window\_edge == 12\_upper\_window\_edge
12\_current\_window\_edge == 12\_lower\_window\_edge
12\_current\_window\_edge != 12\_upper\_window\_edge
12\_current\_window\_edge != 12\_lower\_window\_edge
12\_resend\_edge != 12\_lower\_window\_edge
12\_resend\_edge == 12\_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(l2_enhance == 1)
    l2_enhance = 0;
}
```

# 77.3 Routines

ł

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 l2\_give\_data();

### Description

The l2\_give\_data routine takes an interlayer message buffer associated with a received INFO frame, changes the SDU offset to point to higher-level data, and sends a DL\_DATA IND primitive up to Layer 3 along with a reference to this buffer. The softkey equivalent of this routine is the GIVE\_DATA action on the Protocol Spreadsheet.

### Example

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

### LAYER: 2

### (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.

### <u>Inputs</u>

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

#### <u>Synopsis</u>

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

<u>Synopsis</u>

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 (PROTOCL) RSET\_NS action on the Protocol Spreadsheet.

#### Example

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

```
LAYER: 2
```

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

## send\_frame

## <u>Synopsis</u>

extern void send frame(il\_buffer\_number, relay\_baton, data\_start\_offset, transmit\_frame\_ptr); unsigned short il\_buffer\_number; unsigned short relay\_baton; unsigned short data\_start\_offset; struct send\_frame\_structure

unsigned char sapi\_type; unsigned char tei\_type; unsigned char cr\_type; unsigned char rr\_type; unsigned char nr\_type; unsigned char ns\_type; unsigned char p\_f\_type; unsigned char bcc\_type; unsigned char sapi\_value; unsigned char tei\_value; unsigned char cntrl\_byte; unsigned char nr\_value; unsigned char ns\_value; };

struct send\_frame\_structure \* transmit\_frame\_ptr;

#### Description

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

#### Inputs

The first parameter is the interlayer message buffer number. See Section 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.

#### {

static unsigned short il\_buffer\_number; static unsigned short relay\_baton; static unsigned short data\_start\_offset; struct send\_frame\_structure {

unsigned char sapi\_type; unsigned char tei\_type; unsigned char cr\_type; unsigned char frame\_type; unsigned char nr\_type; unsigned char ns\_type; unsigned char p\_f\_type; unsigned char bcc\_type; unsigned char sapi\_value; unsigned char tei\_value; unsigned char cntrl\_byte; unsigned char nr\_value; unsigned char ns\_value;

};

struct send\_frame\_structure transmit\_frame; static char transmit\_string [] = "((FOX))";

#### LAYER: 2

{

STATE: send\_a\_frame CONDITIONS: KEYBOARD " " ACTIONS:

\_get\_il\_msg\_buff(&il\_buffer\_number, &relay\_baton);

\_start\_il\_buff\_list(il\_buffer\_number,&data\_start\_offset);

transmit\_frame.bcc\_type = 1;

- \_insert\_il\_buff\_list\_cnt(il\_buffer\_number, data\_start\_offset, &transmit\_string[0], (sizeof(transmit\_string) - 1));
- send\_frame(il\_buffer\_number, relay\_baton, data\_start\_offset, &transmit\_frame);
  }

• •

# 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:

dte\_packet && (m\_message\_type == 0x7b)
}

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

CONDITIONS:

dte\_packet

ł

}

Туре	Variable	Value (hex/de	cimal)	Meaning
extern event	dte_packet		detected	en a DTE packet is I. Line Setup ed for emulate or mode.
extern event	dce_packet		detected	en a DCE packet is I. Line Setup ed for emulate or mode.
extern volatile const unsigned char	m_packet_bcc_type	1 2 3	good bad abort	
				up configured for or monitor mode.
extern volatile const unsigned char	m_prot_disc	00-ff/0-255	discrimin	alue of protocol ator—should be 8. Lin onfigured for emulate o mode.
extern volatile const unsigned char	m_call_ref_flag	0 1	originatio destinati	
				up configured for or monitor mode.
extern volatile const unsigned char	m_message_type_defin	ed O		alue received is not a value for a LAPD
		1 A	Actual v the follow	alue received is one of wing valid values for a essage type:
			1 2 5 7 d/13 f/15 20/32 21/33 22/34 25/37 26/38 2d/45 2e/46 40/64 45/69 48/72 4d/77 5a/90 60/96 62/98 64/100 68/104 6a/106	alerting call proceeding setup connect setup ack connect ack user info suspend rej resume rej suspend resume suspend ack resume ack detach disconnect detach ack release release complete cancel facility register cancel ack facillty ack

Table 78-1 Q.931 Variables

(m\_message\_type\_defined continued on next page)

Туре	Variable	Value (hex/de	cimal)	Meaning
	(m_message_type_defin	ed continued)	70/112 72/114 74/116 79/121 7b/123 7d/125	cancel rej facility rej register rej congestion control info status
				p configured for or monitor mode.
extern volatile const unsigned char	m_message_type	00-ff/0-255		-type byte. Line Setu d for emulate or
extern volatile const unsigned char	m_call_ref_len	0–15	value field	the call-reference I. Line Setup d for emulate or node.
extern volatile const unsigned char	m_info_element_ien		field. The informatic	information element e total includes all on elements. Line nfigured for emulate o node.
extern volatile const unsigned char *	m_ptr_to_call_ref		value field byte, con reference	the call-reference d. Begins at the first taining the call elength. Line Setup d for emulate or node.
extern volatile const unsigned char *	m_ptr_to_info_element		element f first byte message	o the information ield. Begins at the after the -type byte. Line Setu d for emulate or node.
extern unsigned char	l3_enhance	0 1 4 5 8 9 12/18	normal reverse low reverse lo blink reverse b blink low	
				p configured for or monitor mode.
extern unsigned char	l3_suppress	0 1	off on	
				ip configured for or monitor mode.

## (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:

## CONDITIONS:

}

dte\_packet && (m\_message\_type == 0x7b)

- 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.
- 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:

```
CONDITIONS: DTE SUSPEND
ACTIONS:
{
13_enhance = 1;
}
CONDITIONS: DTE STATUS
ACTIONS:
{
13_suppress = 1;
}
```

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

CONDITIONS: DTE INFO { *l3\_enhance* == 1 } ACTIONS: { *l3\_enhance* = 0; } or an ACTIONS block:

CONDITIONS: DTE INFO ACTIONS: {

if(13\_enhance == 1) 13\_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

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

# Table 79-1SS#7 Layer 2 Variables

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

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

{
 dte\_frame && (m\_unit\_type == 1)
}

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

CONDITIONS:

dte\_frame }

{

## (B) Status Variables

Status variables are those in Table 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  $m_{\rm prefix}$  are valid in either emulate or monitor mode:  $m_{\rm unit\_type}$ ,  $m_{\rm bib}$ ,  $m_{\rm fib}$ ,  $m_{\rm li}$ ,  $m_{\rm so0}$ , and  $m_{\rm frame\_bcc\_type}$ .

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:

{

12_suppress = 1;

}

CONDITIONS:

{

dte_frame && (m_unit_type == 2) && (m_so0 == 2)

}

ACTIONS:

{

12_enhance = 1;

}
```

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

Table 79-2 SS#7 Layer 1 Variables

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

```
CONDITIONS:
{
dte_frame && (m_unit_type == 2) && (m_so0 == 2) && (12_enhance == 0)
}
ACTIONS:
{
12\_enhance = 1;
}
or an ACTIONS block:
CONDITIONS:
{
dte_frame \&\& (m\_unit\_type == 2) \&\& (m\_so0 == 2)
}
ACTIONS:
{
 if(l2_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:

{ dte\_packet && (m\_sio\_si == 0) }

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

CONDITIONS: {
 dte\_packet
}

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

Table 80-1 SS#7 Layer 3 Variables

(m\_code\_type continued on next page)

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

header = m\_code\_type & 0x0f;

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

Туре	Variable	Value (hex/d	lecimal)	Meaning
(m_code_type continued)			SCCP	headers:
		1	cr	
		2	cc	
		3	cref	
		4	risd	
		5	ric	
		6	dt1	
		7	dt2	
		8	ak	•
		9	udt	
		a/10	udts	
		b/11	ed	
		c/12	ea	
		d/13	rsr	
		e/14	rsc	
		f/15	err	
		10/16	it	
			NETM	headers:
		11/17	COO	
		12/18	eco	
		13/19	rct	
		14/20	tfp	
		15/21	rsp (U	S format only)
		15/21		CITT format only)
		16/22	lin	
		18/24	dic	
		21/33	coa	
		22/34	eca	
		23/35	tfc	
		24/36	tcp (U	S format only)
		25/37	rsr (U	S format only)
		25/37	rst (C	CITT format only, nation
			option	)
		26/38	lun	
		28/40	CSS	
		34/52	tfr	
		35/53		IS format only)
		36/54	lia	
		38/56	cns	
		44/68		S format only)
		45/69		S format only)
		46/70	lua	
		48/72	cnp	
		51/81	cbd	
		54/84	tfa	
		56/86	lid	
		61/96	cba	
		64/100		JS format only)
		66/102	lfu	o format only
		76/118	i ili	
			lri	
		86/134	48.8	

n_code_type continued)	Туре	Variable	Value (hex/o	decimal) Meaning
6         anu           10         reserved           11/17         iam           12/18         gsm           13/19         grq           14/20         acm           15/21         sec           15/22         anc           17/23         rig           17/23         rig           18/24         mgb           19/25         cfm           21/33         iai           24/36         cgc           26/38         ann           27/39         blo           28/40         mba           28/41         cpm           31/49         sam           32/50         cot           36/53         nnc           38/56         mgu           38/57         cpa           41/05         sao           41/05 <th>a_code_type continued)</th> <th></th> <th></th> <th>TUP headers:</th>	a_code_type continued)			TUP headers:
10       reserved         11/17       lam         12/18       gsm         13/19       grq         13/10       acm         15/21       sec         16/22       anc         17/23       rig         18/24       mgb         19/25       cfm         21/33       lai         24/36       ohg         26/37       cgc         26/38       ann         27/99       blo         28/40       mba         29/41       cpm         31/49       sam         32/50       cot         35/55       bla         38/56       mgu         39/57       opa         31/49       sam         32/50       cot         31/55       bla         38/56       mgu         39/57       opa         41/65       sac         41/55       sac         42/66       cof         45/69       adi         45/70       ch         56/86       ran         56/86       foh         56/86 <td></td> <td></td> <td>6</td> <td>2211</td>			6	2211
11/17       iam         12/18       gsm         13/19       grq         14/20       acm         15/21       sec         16/22       anc         17/23       rig         18/24       mgb         19/25       cfm         21/33       iai         24/36       chg         25/37       cgc         26/38       ann         27/39       bio         28/40       mba         29/41       cpm         31/49       sam         32/50       cot         31/49       sam         32/50       cot         33/56       mgu         33/57       cpa         41/65       sao         38/56       mgu         39/57       cpa         41/65       sao         42/66       cof         42/66       cof         42/70       cif         42/71       ubi         48/72       csv         56/88       hgb         59/89       cvm         59/88       cif         48/710 <td></td> <td></td> <td>10</td> <td></td>			10	
12/18       grm         13/19       grm         13/19       grm         14/20       acm         15/21       sec         15/21       sec         17/23       rig         18/24       mgb         19/25       cfm         21/33       iai         24/36       chg         25/37       cgc         26/38       ann         27/39       bio         28/40       mba         28/41       cpm         28/43       sam         31/48       sam         32/53       nnc         36/54       cbk         37/55       bla         38/56       mgu         38/56       mgu         41/65       sao         41/65       sao         41/65       sao         41/66       cf         56/66       ran         56/66	-			
13/19       grq         14/20       acm         15/21       sec         16/22       arc         17/23       rig         18/24       mgb         19/25       cfm         21/33       ial         24/36       chg         25/37       cgc         26/38       ann         27/39       bio         27/39       bio         28/40       mba         29/41       cpm         31/49       sam         32/50       cot         36/54       cbk         37/55       bla         38/56       mgu         39/57       cpa         41/65       sao         42/66       cof         45/69       adi         46/70       cff         47/71       ubl         48/72       mua         49/73       csv         55/85       cfi         56/86       ran         57/87       uba         58/88       hgb         59/89       cvm         68/101       sb         68/102 <td></td> <td></td> <td></td> <td></td>				
14/20       acm         15/21       sec         16/22       anc         17/23       rig         18/24       mgb         19/25       cfm         21/33       iai         24/36       obg         26/37       ogc         26/38       ann         27/39       bio         28/40       mba         29/41       cpm         31/49       sam         32/50       cot         35/53       nnc         36/54       cbk         38/55       bia         38/56       mgu         41/65       sao         42/66       cof         45/69       adi         46/70       cff         47/71       ubi         48/72       mua         49/73       csv         55/85       cfi         58/88       hgb         59/89       cvm         66/101       sb         66/102       fot         67/113       ccr         75/117       un         75/117       reci         77/11				
15/21       sec         16/22       anc         17/23       rig         18/24       mgb         19/25       cfm         21/33       iai         24/36       chg         25/37       cgc         26/38       ann         27/39       bio         28/40       mba         29/41       cpm         31/49       sam         32/50       cot         36/54       cbk         37/55       bia         38/56       mgu         39/57       cpa         41/65       sao         42/66       cof         45/69       adi         46/70       cff         47/71       ubi         48/72       mua         49/73       csv         56/86       ran         57/87       uba         68/101       sbi         68/102       fot         68/103       cr         78/11       ubi         68/104       hba         68/105       cr         78/117       un         78/120				
16/22       anc         17/23       rig         18/24       mgb         19/25       cfm         21/33       iai         24/36       chg         25/37       cgc         26/38       ann         27/39       blo         28/30       mba         28/40       mba         29/41       Cpm         31/49       sam         32/50       cot         35/53       nnc         35/54       cbk         37/55       bla         38/56       mgu         38/56       mgu         38/56       mgu         38/56       mgu         38/56       mgu         38/56       mgu         41/65       sao         42/66       cof         45/59       adi         46/70       cff         47/71       ubl         48/72       mua         48/72       mua         56/65       cfi         56/66       ran         57/87       uba         58/88       hgb         68/101 <td></td> <td></td> <td>15/21</td> <td></td>			15/21	
17/23       rlg         18/24       mgb         19/25       cfm         21/33       iai         24/36       chg         25/37       cgc         26/38       ann         27/39       blo         28/40       mba         29/41       cpm         31/49       sam         32/50       cot         35/53       nnc         35/54       cbk         37/55       bla         38/56       mgu         39/57       cpa         41/65       sao         42/66       ccf         45/59       adi         45/59       adi         45/59       adi         45/59       adi         45/58       cfl         55/85       cfl         55/85       cfl         55/85       cfl         57/87       uba         58/101 <td></td> <td></td> <td></td> <td></td>				
18/24       mgb         19/25       cfm         21/33       ial         24/36       chg         25/37       cgc         26/38       ann         27/39       blo         28/40       mba         28/41       cpm         31/49       sam         32/50       cot         35/53       nnc         35/53       nc         35/54       cbk         37/55       bla         38/56       mgu         39/57       cpa         41/65       sao         42/66       ccf         45/59       adi         46/70       cff         47/71       ubl         48/72       mua         48/72       mua         48/72       mua         55/85       cfi         56/86       ran         57/87       uba         58/88       hgb         59/89       cvm         68/101       sb         68/102       fot         67/103       ccr         77/119       rsc         78/120<			17/23	
19/25       cfm         21/33       iai         21/33       iai         22/36       chg         26/37       cgc         28/40       mba         28/40       mba         28/41       cpm         31/49       sam         31/49       sam         32/50       cot         35/53       nnc         36/54       cbk         37/55       bia         38/56       mgu         38/56       mgu         38/57       cpa         41/65       sao         41/65       sao         41/65       sao         42/66       cof         45/69       adi         41/65       sao         41/65       sao         41/65       sao         41/65       sao         51/85       cfi         41/65       sao         51/87       ubi         45/68       adi         51/87       ubi         51/87       ubi         51/87       ubi         51/88       hgb         51/87 <td></td> <td></td> <td></td> <td></td>				
21/33       iai         24/36       ohg         26/37       cgc         26/38       ann         27/39       blo         28/40       mba         29/41       cpm         31/49       sam         32/50       cot         33/55       bla         32/50       cot         33/55       bla         33/56       mgu         33/57       cpa         41/65       sao         41/65       sao         41/65       sao         42/66       cof         41/65       sao         42/66       cof         45/70       cff         47/71       ubl         48/72       mua         49/73       csv         55/85       cfi         56/86       ran         57/87       uba         58/88       hgb         59/89       cvm         65/101       ssb         66/102       fot         67/103       cor         68/104       hba         68/105       crm         78/1			19/25	
26/37       cgc         26/38       ann         27/39       bio         28/40       mba         29/41       cpm         31/49       sam         32/50       cot         35/53       nnc         36/54       cbk         37/55       bla         38/56       mgu         39/57       cpa         41/65       sao         42/66       cof         45/69       adi         46/70       cif         47/71       ubl         48/72       mua         49/73       csv         55/85       cfl         56/86       ran         57/87       uba         58/88       hgb         59/89       cvm         65/101       ssb         66/102       fot         67/103       ccr         68/104       hba         78/120       hgu         78/120       hgu         78/120       hgu         78/120       hgu         78/133       los         88/136       hua <td></td> <td></td> <td>21/33</td> <td>lai</td>			21/33	lai
26/37       cgc         26/38       ann         27/39       blo         28/40       mba         29/41       cpm         31/49       sam         32/50       cot         35/53       nnc         36/54       cbk         37/55       bla         38/56       mgu         39/57       cpa         41/65       sao         42/66       cof         45/69       adi         46/70       cif         47/71       ubl         48/72       mua         48/72       mua         48/73       csv         55/85       cfl         56/86       ran         57/87       uba         59/89       cvm         65/101       ssb         66/102       fot         67/103       ccr         68/104       hba         78/120       hgu         78/120       hgu         78/120       hgu         78/120       hgu         78/120       hgu         78/120       hgu <td< td=""><td></td><td></td><td>24/36</td><td></td></td<>			24/36	
27/39       blo         28/40       mba         29/41       cpm         31/49       sam         32/50       cot         35/53       nnc         36/54       cbk         37/55       bla         38/56       mgu         39/57       cpa         41/65       sao         42/66       ccf         45/69       adi         48/72       mua         49/73       csv         55/85       cfl         56/86       ran         57/87       uba         58/88       hgb         59/99       cvm         65/101       ssb         66/102       fot         67/103       ccr         76/118       ccl         77/119       rsc         78/120       hgu         79/121       cli         85/133       los         88/136       hua			25/37	cgc
28/40       mba         29/41       cpm         31/49       sam         32/50       cot         32/50       cot         35/53       nnc         36/54       cbk         37/55       bla         38/56       mgu         39/57       cpa         41/65       sao         42/66       ccf         45/69       adi         46/70       clf         47/71       ubl         48/72       mua         49/73       csv         55/85       cfl         56/86       ran         57/87       uba         58/88       hgb         58/88       hgb         58/88       hgb         58/89       cvm         66/102       fot         67/103       scb         68/104       hba         69/105       crm         76/118       ccl         77/19       rsc         78/120       hgu         78/121       cli         88/136       hua				
29/41       cpm         31/49       sam         32/50       cot         35/53       nnc         36/54       cbk         37/55       bla         38/56       mgu         39/57       cpa         41/65       sao         42/66       ccf         45/69       adi         46/70       clf         47/71       ubl         48/72       mua         49/73       csv         55/85       cfi         56/86       ran         57/87       uba         58/88       hgb         59/89       cvm         66/102       fot         67/103       ccr         68/104       hba         69/105       crm         76/118       ccl         77/119       rsc         78/118       ccl         78/120       hgu         78/133       los         88/136       hua			27/39	
31/49       sam         32/50       cot         35/53       nnc         36/54       cbk         37/55       bla         38/56       mgu         39/57       cpa         41/65       sao         42/66       ccf         45/69       adi         45/69       adi         46/70       cff         47/71       ubl         48/72       mua         49/73       csv         56/86       ran         59/89       cvm         59/89       cvm         65/101       ssb         66/102       fot         67/103       ccr         68/104       hba         69/105       crm         76/118       ccl         79/121       cli         79/121       cli         8				
32/50       cot         33/53       nnc         36/54       cbk         37/55       bla         37/55       bla         39/57       cpa         41/65       sao         41/65       sao         41/65       sao         45/69       adi         46/70       clf         47/71       ubl         48/72       mua         49/73       csv         55/85       cfl         56/86       ran         57/87       uba         58/88       hgb         59/89       cvm         66/102       fot         67/103       ccr         68/104       hba         69/105       crm         76/118       ccl         77/119       rsc         78/120       hgu         79/121       cli         85/133       los         88/136       hua				•
35/53       nnc         36/54       cbk         37/55       bla         38/56       mgu         39/57       cpa         41/65       sao         42/66       ccf         45/69       adi         46/70       clf         47/71       ubl         48/72       mua         49/73       csv         55/85       cfl         56/86       ran         59/89       cvm         65/101       ssb         66/102       fot         67/103       ccr         68/104       hba         69/105       crm         75/117       unn         76/118       ccl         77/119       rsc         78/120       hgu         79/121       cli         85/133       los         88/136       hua				
36/54       cbk         37/55       bla         38/56       mgu         39/57       cpa         41/65       sao         42/66       ccf         45/69       adi         45/69       adi         46/70       clf         47/71       ubl         48/72       mua         49/73       csv         55/85       cfl         57/87       uba         57/87       uba         58/88       hgb         66/102       fot         67/103       ccr         68/104       hba         69/105       crm         76/118       ccl         78/120       hgu         78/121       cli         85/133       los         88/136       hua				
37/55       bla         38/56       mgu         39/57       cpa         41/65       sao         42/66       ccf         45/69       adi         46/70       clf         47/71       ubl         48/72       mua         49/73       csv         55/85       cfl         56/86       ran         57/87       uba         58/88       hgb         59/89       cvm         66/102       fot         67/103       ccr         68/104       hba         69/105       crm         76/118       ccl         77/119       rsc         78/120       hgu         78/120       hgu         79/121       cli         88/136       hua				
38/56       mgu         39/57       cpa         41/65       sao         42/66       ccf         45/69       adi         46/70       cif         47/71       ubl         48/72       mua         49/73       csv         55/85       cfl         56/86       ran         57/87       uba         58/88       hgb         59/89       cvm         65/101       ssb         66/102       fot         67/103       ccr         68/104       hba         69/105       crm         76/118       ccl         77/119       rsc         78/120       hgu         78/120       hgu         78/133       los         85/133       los         85/133       los				
39/57       cpa         41/65       sao         42/66       cof         45/69       adi         46/70       cif         47/71       ubl         48/72       mua         49/73       csv         55/85       cfl         56/86       ran         57/87       uba         58/88       hgb         59/89       cvm         66/102       fot         67/103       ccr         68/104       hba         69/105       crm         76/118       ccl         77/119       rsc         78/120       hgu         78/120       hgu         79/121       cli         85/133       los         88/136       hua				
41/65       sao         42/66       ccf         45/69       adi         46/70       clf         47/71       ubl         48/72       mua         49/73       csv         55/85       cfl         56/86       ran         57/87       uba         58/88       hgb         59/89       cvm         65/101       ssb         66/102       fot         67/103       ccr         68/104       hba         69/105       crm         76/118       ccl         77/119       rsc         78/120       hgu         79/121       cli         85/133       los         88/136       hua	<ul> <li>A state of the sta</li></ul>			
42/66       ccf         45/69       adi         46/70       clf         47/71       ubl         48/72       mua         48/72       mua         49/73       csv         55/85       cfl         56/86       ran         57/87       uba         58/88       hgb         59/89       cvm         66/102       fot         67/103       ccr         67/103       ccr         68/104       hba         69/105       crm         75/117       unn         76/118       ccl         77/119       rsc         78/120       hgu         79/121       cli         85/133       los         88/136       hua				
45/69       adi         46/70       clf         47/71       ubl         48/72       mua         49/73       csv         55/85       cfl         56/86       ran         57/87       uba         58/88       hgb         59/89       cvm         65/101       ssb         66/102       fot         67/103       ccr         68/104       hba         69/105       crm         75/117       unn         76/118       ccl         77/119       rsc         78/120       hgu         79/121       cli         85/133       los         88/136       hua				
46/70       clf         47/71       ubl         48/72       mua         49/73       csv         55/85       cfl         56/86       ran         57/87       uba         58/88       hgb         59/89       cvm         65/101       ssb         66/102       fot         67/103       ccr         68/104       hba         69/105       crm         76/117       unn         76/118       ccl         77/119       rsc         78/120       hgu         79/121       cli         85/133       los         88/136       hua				
47/71       ubl         48/72       mua         49/73       csv         55/85       cfl         56/86       ran         57/87       uba         58/88       hgb         59/89       cvm         65/101       ssb         66/102       fot         67/103       ccr         68/104       hba         69/105       crm         75/117       unn         76/118       ccl         77/119       rsc         78/120       hgu         79/121       cli         85/133       los         88/136       hua				
48/72       mua         49/73       csv         55/85       cfl         56/86       ran         57/87       uba         58/88       hgb         59/89       cvm         65/101       ssb         66/102       fot         67/103       ccr         68/104       hba         69/105       crm         75/117       unn         76/118       ccl         77/119       rsc         78/120       hgu         79/121       cli         85/133       los         88/136       hua				
49/73       csv         55/85       cfl         56/86       ran         57/87       uba         58/88       hgb         59/89       cvm         65/101       ssb         66/102       fot         67/103       ccr         68/104       hba         69/105       crm         75/117       unn         76/118       ccl         77/119       rsc         78/120       hgu         79/121       cli         85/133       los         85/136       hua			48/72	
55/85       cfl         56/86       ran         57/87       uba         58/88       hgb         58/89       cvm         65/101       ssb         65/102       fot         66/102       fot         67/103       ccr         68/104       hba         69/105       crm         75/117       unn         76/118       ccl         77/119       rsc         78/120       hgu         79/121       cli         85/133       los         88/136       hua			49/73	
56/86       ran         57/87       uba         58/88       hgb         59/89       cvm         65/101       ssb         66/102       fot         67/103       ccr         68/104       hba         69/105       crm         75/117       unn         76/118       ccl         77/19       rsc         78/120       hgu         79/121       cli         85/133       los         88/136       hua			55/85	
57/87       uba         58/88       hgb         59/89       cvm         65/101       ssb         66/102       fot         67/103       ccr         68/104       hba         69/105       crm         75/117       unn         76/118       ccl         77/119       rsc         78/120       hgu         79/121       cli         85/133       los         88/136       hua				
59/89       cvm         65/101       ssb         66/102       fot         67/103       ccr         68/104       hba         69/105       crm         75/117       unn         76/118       ccl         77/119       rsc         78/120       hgu         79/121       cli         85/133       los         88/136       hua			57/87	uba
65/101       ssb         66/102       fot         67/103       ccr         68/104       hba         69/105       crm         75/117       unn         76/118       ccl         77/119       rsc         78/120       hgu         79/121       cli         85/133       los         88/136       hua			58/88	hgb
66/102       fot         67/103       ccr         68/104       hba         69/105       crm         75/117       unn         76/118       ccl         77/119       rsc         78/120       hgu         79/121       cli         85/133       los         88/136       hua			59/89	
67/103 ccr 68/104 hba 69/105 crm 75/117 unn 76/118 ccl 77/119 rsc 78/120 hgu 79/121 cli 85/133 los 88/136 hua			65/101	ssb
68/104 hba 69/105 crm 75/117 unn 76/118 ccl 77/119 rsc 78/120 hgu 79/121 cli 85/133 los 88/136 hua			66/102	
69/105       crm         75/117       unn         76/118       ccl         77/119       rsc         78/120       hgu         79/121       cli         85/133       los         88/136       hua				
75/117       unn         76/118       ccl         77/119       rsc         78/120       hgu         79/121       cli         85/133       los         88/136       hua				
76/118       ccl         77/119       rsc         78/120       hgu         79/121       cli         85/133       los         88/136       hua			69/105	
77/119 rsc 78/120 hgu 79/121 cli 85/133 los 88/136 hua			75/117	
78/120 hgu 79/121 cli 85/133 los 88/136 hua			76/118	
79/121 cli 85/133 los 88/136 hua				
85/133 los 88/136 hua				
88/136 hua				
			00/130	nua

# INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

(m\_code\_type continued on next page)

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

Line Setup configured for emulate or monitor mode.

Туре		Variable	Value (hex/dec	imal) Meaning
extern volatile unsigned ic	ong	m_label_dpc	0-3fff/ 0-16383 0-ffffff/	CCITT format (2 bytes)
			0-16777215	ANSI format (3 bytes) Line Setup configured for emulate or monitor mode.
extern volatile unsigned lo	ong	m_label_opc	0-3fff/ 0-16383 0-ffffff/	CCITT format (2 bytes)
			0-16777215	ANSI format (3 bytes) Line Setup configured for emulate or monitor mode.
extern volatile const unsi	gned char	m_label_sis	0-f/0-15 0-1f/0-31	CCITT format ANSI format Line Setup configured for emulate or monitor mode.
extern volatile unsigned s	short	m_cic	0-fff/0-4095 0-ffff/0-65535	TUP MSUs ISDN MSUs Line Setup configured for emulate or monitor mode.
extern unsigned char		I3_enhance	0 1 4 5 8 9 12/18	normal reverse low reverse low blink reverse blink blink low
				Line Setup configured for emulate or monitor mode.
extern unsigned char		I3_suppress	0 1	off on Line Setup configured for emulate or monitor mode.

# (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:

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\_label\_opc, m\_label\_sls, and m\_cic.

# (C) Controlling Protocol Trace Display

To enhance or suppress particular packets on the Layer 3 Protocol Trace screen in emulate or monitor mode, assign a coded value to *l3\_enhance* or *l3\_suppress*. The values are listed in Table 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:
{
    I3_suppress = 1;
}
CONDITIONS:
{
    dte_packet && (m_sio_si == 0) && (m_code_type == 0x34)
}
ACTIONS:
{
    I3_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(12_suppress == 0)
    12_suppress = 1;
}
```

80-7

# 80.3 Routines

There are no routines associated exclusively with SS#7.

Appendix A Operator Messages

# **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.

#### MESSAGE

Attempt to transfer source to itself

Attempted to mount uninitialized disk

Attempting to initialize link

BNDX message request failed

Backup complete, no errors

Bad object file format

Can't load object file – Incompatible FEB installed

# MEANING

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.

Check disk. It may require formatting.

Physical link being established prior to transfer.

Message should not normally appear. If it recurs, contact Customer Service.

Duplication process is successful.

Data is not recognized in format of object file. Try again to save the source file as an object file.

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.

Can't load object file – Incompatible MPM addressing

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.

Can't load object file – Incompatible mux installed

Can't load object file – Insufficient MPMs

Can't load object file - No mux installed

Can't read disk

Cannot append to a wrapped DAT

Cannot copy a file to itself

Cannot copy directory tree into itself

Cannot delete a non-empty directory

Cannot open file

Cannot open redirect file

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.

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.

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.

Make sure disk is correctly inserted. If message recurs, disk may be bad.

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.

Be certain that name of destination file on File Maintenance screen differs from name of source file.

Attempt to copy a directory into one of its subdirectories. For example, a command to copy */usr* into */usr/programs* will fail.

File named for deletion is directory containing files. Before deleting directory, delete or move files.

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.

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.

Cannot move file across disk boundaries Attempt to rename a file from one disk to another. Make sure only one drive is specified on File Maintenance screen. Cannot remove an open file Attempt was made to copy a directory into itself. Files being copied also need to be deleted, but cannot be since they are open. Copy the directory to another source. In general, close files before attempting to delete them. Cannot unmount disk, files open Attempt was made to remove disk before operation was completed. Cannot write to redirect file Printer Setup menu shows that output will be redirected to a file. Error in trying to write to the file. Check disk. Change floppy disk 1 During multi-disk recording operation, disk in drive 1 has been filled. Remove old disk and insert new one. During multi-disk recording operation, disk in drive 2 Change floppy disk 2 has been filled. Remove old disk and insert new one. Character buffer not yet allocated From field on the Data Transfer screen shows CHARBUFFER, but unless Run has been executed, there is no character buffer. Press w, and then try the transfer operation again. User has pressed ABORT softkey or mount to arrest the Compilation aborted Compile operation (from the File Maintenance menu). Destination file may have been partially overwritten if compile was to an existing file. The Compile command (from the File Maintenance Compilation completed menu) has been executed. The Compile command (from the File Maintenance Compilation failed - Errors detected menu) has been aborted because of errors. Go to the Protocol Spreadsheet and press EDT, F8 to display the first error message. The Compile command (from the File Maintenance Compilation is in progress menu) is being executed to compile and save a file of standard C code as object code.

Copy completed

Copy is in progress

Could not load a layer personality package

Current test invalidated

Data transfer source and destination are the same

Destination disk does not contain user file system

Destination file is a directory

Destination file is write protected

Directory file expected

Directory is empty

Directory is not empty

Disk corrupted

Disk duplication aborted

Selected file(s) has (have) been copied successfully.

Selected file(s) is (are) being copied.

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.

Changes have been made to the menu screens or Protocol Spreadsheet which invalidate a loaded object file.

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**.

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.

Copy or save operation not complete because file named to receive copy is a directory. Change destination filename and re-execute.

Change destination filename or write-enable file. Then repeat save or copy operation.

Check directory named in Change Directory command. Use only names labeled DIR in file listings.

Attempted to copy the contents of an empty directory.

Directory cannot be deleted until all of the files it contains have been deleted.

Disk is worn out or damaged and should not be used for future operations.

Operator has aborted disk duplication. Data on destination disk may have been partially overwritten.

Disk duplication in progress

Disk formatted

Disk full

Disk not mounted

Disk record error (controller error) -- Aborted

Disk record error (timeout) -- Aborted

Display screen command failed

Entering testprep

Error during load of code file

Error trying to print file

Error trying to print screen

Errors occurred during load

Errors occurred during save

Errors occurred during testprep

Disk is being duplicated. Do not remove disks from active drives.

Formatting operation is complete.

No space left on disk to perform operation. Use a new disk, or remove unneeded data from disk.

Re-insert disk and attempt operation again. Also try to power-up again. If message recurs, the disk may be bad.

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.

May be an internal error or bad disk. Try again with a new disk. Disk may be write protected or recording too fast.

Message should not normally appear. Contact Customer Service if it recurs.

First status message entering Run mode. Test preparation mode precedes compilation of program.

Check code files. Message indicates code file is not found, or code file has been modified.

Try printing again. If attempt fails, disk file is probably corrupted.

Try printing again. If problem recurs, contact Customer Service.

Try loading again. If attempt fails, disk file is probably corrupted.

Try saving again. If attempt fails, disk is probably corrupted.

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.

Fatal Hardware Error

Fatal Software Error

FE buffer overflowed – Incoming data halted

File access error

File copy aborted

File is a directory

File is write-protected

File loaded

File name not found

File name already exists

File saved

File size can't be increased, index block full

Formatting disk – max floppy disk DAT allowed = 1422K bytes

Formatting disk – max hard disk DAT allowed = 20774K bytes

Invalid hardware setup. Contact Customer Service.

Message should not normally appear. If it recurs, contact Customer Service.

Data is coming in faster than it can be received.

File named cannot be accessed. Check disk.

User has pressed ABORT softkey to arrest copy operation. Destination file may have been partially overwritten if copy was to an existing file.

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.

File named cannot be deleted or saved. Check name. To perform operation on named file, write-enable it from the File Maintenance screen.

File has been loaded successfully as a Program, Setup, or Object file.

Filename (or directory) as entered does not appear in listings. Check spelling of entry. Make sure you are operating in correct directory.

Attempt to use the Make Directory command, naming a directory that already exists.

File has been saved successfully as a Program, Setup, or Object file.

File is larger than the file system can handle.

Too much space specified for data acquisition tracks. Maximum space has been allocated.

Too much space specified for data acquisition tracks. Maximum space has been allocated.

Formatting disk

Formatting will destroy data - Depress F1 key to continue.

Function failed -- Check media

Function(s) not yet implemented

Il buffer services error

Illegal device name

Illegal expansion unit, not 1-255

Illegal file number passed to open

Illegal major device number given

Illegal parameter to volume init. function

Illegal pathname

Illegal position parameter, not 0-2

Illegal synchronization mode, not 0-2

Indirect stat update msg received

Formatting in progress. Do not remove disk from active drive.

Message appears when a formatted disk has been inserted for reformatting. Press ABORT to avoid overwriting data. Press F1 to format the disk.

Attempt operation again. If operation still fails, disk may be bad. Try new disk.

Operation attempted is not available with the software version installed.

Error in using OSI variables or routines. May occur, for example, if operation is attempted on buffer which no longer exists. Set maintain bits at each layer that needs to reserve the buffer for subsequent operations.

Message should not normally appear. If it recurs, contact Customer Service.

Message should not normally appear. If it recurs, contact Customer Service.

Message should not normally appear. If it recurs, contact Customer Service.

Message should not normally appear. If it recurs, contact Customer Service.

Message should not normally appear. If it recurs, contact Customer Service.

Pathname provided is incomplete or invalid. Check entry.

Message should not normally appear. If it recurs, contact Customer Service.

Message should not normally appear. If it recurs, contact Customer Service.

Message should not normally appear. If it recurs, contact Customer Service.

Insert next disk -- Depress F1 key to continue

Insert destination disk, depress F1 key to continue

Insert source disk, depress F1 key to continue

Internal disk sub-system error

Inter-processor communication overrun

Invalid contents in field

Invalid DAT block version number

Invalid DAT version number

Invalid data type

Invalid destination

Invalid disk sub-system function number

Invalid file identifier, no such open file

Invalid file identifier, out of range

Invalid filetype

More than one disk required to perform duplication.

Operation involving more than one disk being performed using one drive.

Operation involving more than one disk being performed using one drive.

Message should not normally appear. If it recurs, contact Customer Service.

Communication from MPM to CPM occurring too fast for CPM. Available buffer space exceeded.

Entry made in menu field is illegal.

Each block in DAT has a version number. If it is wrong, the disk may be corrupted.

Header in DAT has a version number. If it is wrong, the disk may be corrupted.

During attempted playback, INTERVIEW did not recognize type of data. Be certain recorded data rather than program data is being accessed.

Destination file in a Copy command is a relative pathname on a drive which is not the current drive.

Message should not normally appear. If it recurs, contact Customer Service.

Message should not normally appear. If it recurs, contact Customer Service.

Message should not normally appear. If it recurs, contact Customer Service.

Command cannot be used on file type indicated. A Load command, for example, is not valid for SYS files.

#### Appendix A1 Interactive Messages

Invalid filetype for viewing

Invalid layer number

Invalid object code version

Invalid section name

Invalid stat update msg received

Load aborted

Load is in progress

Loaded package and configuration screen don't match

Marked entry not copied

#### Marked entry not deleted

#### Marked entry not printed

View command cannot be used for data in file indicated. Files with type SYS, for example, cannot be viewed.

User has entered layer number out of valid range.

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.

Message should not normally appear. If it recurs, contact Customer Service.

Message should not normally appear. If it recurs, contact Customer Service.

Operator has aborted Load operation. Program already residing in INTERVIEW may have been altered.

Selected Program, Setup, or Object file is being loaded.

Message should not normally appear. If it recurs, contact Customer Service.

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.

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.

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.

Maximum number of entries exceeded Error on Tabular Statistics screen. Maximum number of entries is 100. Maximum disks already mounted Message should not normally appear. If it recurs, contact Customer Service. Memory has not been unlocked yet Message should not normally appear. If it recurs, contact Customer Service. Message exchange full Message should not normally appear. If it recurs, contact Customer Service. Message ID too big Message should not normally appear. If it recurs, contact Customer Service. Attempt to rename a directory the same as one of its Move would destroy directory tree subdirectories. structure MPM -- Bus error 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. MPM -- Divide fault Program may include an attempt to divide by zero. Check for other logic errors in program. May indicate a hardware problem, but check program MPM -- Processor fault for logic errors relating to storage allocation. May have attempted to access something that doesn't exist. MPM -- Memory fault 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.

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.

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 FDI or FDE.

MPM -- Stack fault

NEWDISK illegal with source of hard disk
NEWDISK illegal with source of RAM or hard disk

No DAT RAM currently allocated

No default directory set

No file name specified

No package loaded for this layer

No packages loaded

No RAM recording memory available

No start of section indicator

No message entered in message buffer

Obsolete object program – Source must be recompiled

Out of memory

Operation not allowed on specified file

Parent directory of file is write-protected

Parent directory of target file does not exist

When To field for a Data Transfer command is **NEW**, it means that the same drive will be used to perform a transfer involving two disks. Change From field to **FD** or **FD2**.

Attempt to transfer data from RAM without it having been recorded previously to RAM.

Message should not normally appear. If it recurs, contact Customer Service.

Enter or indicate file on which operation attempted is to be performed.

Selection has been made on the Layer Setup screen, but no protocol package has been loaded. Return to Layer Setup, check selection, and press 100.

Selections have been made on the Layer Setup screen, but no protocol packages have been loaded. Return to Layer Setup, check selections, and press **me**.

Program is too large to be recorded into available RAM.

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.

Check BERT screen. Configured menu indicated a message would be sent, but none was entered.

Object file is incompatible with current software.

Insufficient memory to perform operation. (Program is too large to run.)

Selected command cannot be used on file indicated.

Parent directory must be write-enabled before you can modify or delete this file.

Check spelling of directory.

Play underrun

Premature end of section

Previous lock user has died

Print queue is full

Printing is done

Record overrun

Remove screen command failed

Replace screen command failed

Resetting compiled test

Routine calling save\_prog\_setup is unknown

Save aborted

Save is in progress

Seek attempted before beginning of DAT

Data could not be output at speed requested.

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.

Message should not normally appear. If it recurs, contact Customer Service.

Maximum number of print jobs has been requested. Wait for some requests to be completed. Then repeat print operation.

Print jobs requested are completed.

Data being received too rapidly for capture to RAM.

Message should not normally appear. If it recurs, contact Customer Service.

Message should not normally appear. If it recurs, contact Customer Service.

Program being run again without recompiling. Menus can be viewed and selected changes can be made to menus without forcing a recompile.

Message should not normally appear. If it recurs, contact Customer Service.

Operator has aborted save operation. If save was to an existing file, the file may have been partially overwritten.

Selected Program, Setup, or Object file is being saved.

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. Seek attempted past end of DAT

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.

Message should not normally appear. If it recurs, contact Customer Service.

Error in entries made for disk duplication. Check disks selected.

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.

In Interview 10/15/20 file transfer, the source file which was specified does not exist.

During playback, stopped at end of recorded data. During recording, stopped at end of data acquisition tracks.

Program is being compiled.

Message should not normally appear. If it recurs, contact Customer Service.

Seek attempted past end of file

Source & destination can't be the same disk

Source disk does not contain user file system

Source file not found

Stopped at end of DAT

TEST PREPARATION Phase 1 TEST PREPARATION Phase 2 TEST PREPARATION Phase 3 TEST PREPARATION Phase 4 TEST PREPARATION Phase 5 TEST PREPARATION Phase 6 TEST PREPARATION Phase 7 There are no free locks left

Token in load file is invalid

Token is incomplete

Too many files in directory

Too many files on disk, FLIST full

Too many open files for process

Too many open files for system

Too many processes using disk sub-system

Too many source files selected

Transfer aborted

Transfer complete

Transfer in progress

Transmit overrun

Unable to access disk

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.

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.

Maximum number of files that can be displayed is 200. If the current directory contains more than 200 files, this message is displayed.

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.

Each process is limited to a maximum of ten files open at one time.

There is a system-wide limit of 20 files open at one time.

There can be no more than twelve processes using file I/O simultaneously.

Operator has used were 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.

Data transfer operation has been aborted. Partial transfer of data may have occurred, overwriting storage medium at destination.

Data transfer has been completed successfully.

Data transfer being executed.

Attempt to transmit data faster than unit can transmit.

No file named in a Data Transfer to a file, disk not in drive, disk is write-protected, or disk is unformatted.

A1-14

Unable to access disk in selected drive.

Unable to access m list

Unable to execute XEQ key

Unable to open DAT

Unable to open file

Unable to open next disk

Unable to read DAT info block

Unable to read file

Unable to read from DAT

Unable to write to DAT

Unhandled CPM interrupt

Unhandled MPM interrupt

Unknown DAT type

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.

Disk error. Check disk and try operation again. If message recurs, disk may need reformatting.

Attempt to execute a File Maintenance command before the current directory is displayed.

There are no data acquisition tracks on the disk being accessed.

Disk error. Check disk and try operation again. If message recurs, disk may need reformatting.

Recording using more than one disk. Next disk may not be installed.

DAT block not where indicated. Disk may be corrupted.

Check disk.

Check disk.

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.

Message should not normally appear. If it recurs, contact Customer Service.

Press recover from this error. Message should not normally appear, however. If it recurs, contact Customer Service.

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.

Unknown filetype

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.

Screen requested cannot be printed. Refer to section on printing for printable screens.

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.

Unprintable screen

Unrecoverable error during format

## 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 F8.

#### MESSAGE

AR "C" conditions text too long

Bad format in object file

BIB value out of range

Bit mask exceeds maximum length

Cannot find object file

Duplicate state name

Constant reference stack overflow Constant value too long A C region in a CONDITIONS block is more than 300 characters.

MEANING

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.

An SS#7 condition at Layer 2 specifies a BIB= value that is not zero or one.

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.

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.

Attempt to nest constants more than eight deep, or
 constants are defined circularly.

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.

Attempt to use state name twice in the same test.

A2-1

Duplicate test name

Edit buffer full

Empty conditions section

FIB value out of range

Identifier exceeds maximum length

Idle string must contain exactly one character

Illegal bit value

Illegal cause value

Illegal CIC type for ISDN

Illegal CIC type for TUP

Illegal control byte

Illegal diag value

Attempt to use test name twice in the same task (layer).

Spreadsheet program is too large. Use include files.

There is no entry for a CONDITIONS block.

An SS#7 condition at Layer 2 specifies an FIB= value that is not zero or one.

Message should not normally appear. It means, however, that an identifier is too long for the context in which it is being used.

Layer 1 IDLE action includes a string with more than one character.

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.

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.

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.

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.

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.

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.

#### Appendix A2 Error Messages Issued by C Translator

Illegal DPC type

Illegal frame address

Illegal LCN value

Illegal OPC type

Illegal path number

Illegal P/F bit

Illegal PR value

Illegal PS value

Illegal receive count

Illegal SAPI value

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 FFFFFF.

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.

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.

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 FFFFFF.

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.

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.

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.

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.

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.

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.

A2-3

Illegal send count

Illegal SI type

Illegal SLS type

Illegal TEI value

Incomplete EIA action

Invalid constant reference

Invalid counter arguments

Invalid counter value

Invalid day of month

Invalid flag arguments

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.

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.

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.

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.

Required number of softkey selections have not been made for a Layer 1 EIA action.

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.

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.

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.

TIME condition specifies a day of the month which is outside the valid range. Select a value between one and thirty-one.

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.

A2-4

#### Appendix A2 Error Messages Issued by C Translator

Invalid time value

Invalid time of day

Invalid timeout value

Invalid trigger (lacks transitional)

Invalid trigger (multiple transitional-only)

Invalid character in constant name

No closing »

No closing ]

No more errors

Not an object file

Obsolete object file version

TIME condition specifies a time which is outside the valid range for the 24-hour format.

TIME condition specifies a time which is outside the valid range for the 24-hour format.

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.

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.

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.

Valid characters include 0-9, upper- and lower-case letters, and underscores. Name cannot begin with a number.

Double parentheses delimit constants.

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.

There are no more errors to be displayed via GO\_ERR. The next time you press FB, the last error message will be displayed.

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.

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. Obsolete package loaded

Out of buffer space

Out of memory

Premature end of file

Receive string cannot be longer than 32 characters

Reference to undefined constant

Reference to undefined state

Syntax error

There is no next state

Unclosed AR"C" region

Unclosed quoted string Undefined name

Unexpected character in constant name

Unknown object file version

#### WAIT EOF with ENTER\_STATE

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.

The translator has run out of memory.

The translator has run out of memory.

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.

RECEIVE STRING condition at Layer 1 contains more than 32 characters. Note that constants are expanded before they are counted.

Attempt to use a constant that has not been defined.

State name referenced in NEXT\_STATE does not exist.

Program may contain punctuation errors or incomplete softkey selections. This message often accompanies other errors messages.

Included NEXT\_STATE: NEXT, but no state follows.

1) Unequal number of opening and closing curly braces, 2) unclosed quotation marks, or 3) unclosed parentheses.

Insert closing quotation marks at end of string.

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.

Valid characters include 0-9, upper- and lower-case letters, and underscores. Name cannot begin with a number.

Should not normally appear. May be attempting to use version of a linkable-object file which is incompatible with older versions of software.

A trigger condition has both ENTER\_STATE and a line condition including the WAIT\_EOF option. This error is a special case of the "Invalid trigger (multiple transition-only)" error. Appendix A3 Error Messages Issued by C Compiler

# 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

001	Only integral values may be added to pointers.
002	Within constant expressions, the operand of the unary '&' operator must be an object of static storage class.
003	Only integers and pointers may be converted to pointers.
004	Attempt to create more than one instance of a task which waits for fast_event variable—task instance '(identifier)'.
005	Only numeric values may be converted to float.
006	Illegal operation on relocatable value in constant expression.
007	Illegal conversion from a structure or union type.
008	Operands of binary operator have incompatible types.
009	Illegal indirection through a non-pointer value.
010	An integral constant expression is required.
011	A scalar expression is required.
012	Bitfield values are not allowed in constant expressions.
013	Operands of '*', '/', and '%' must be numeric.

## Table A3-1 Numbered Error Messages Returned for C Coding†

† Errors 001 - 699 are returned by the compiler. Errors 700 and higher are returned by the pre-processor.

014	Operands of logical operators must be integers.
015	Pointer values being compared or subtracted in constant expressions must point to the same aggregate.
016	Assignment operators are invalid in constant expressions.
017	Operands of % operator may not be floating point.
018	The ++ and operators are invalid in constant expressions.
019	A non-relocatable constant expression is expected.
020	Relocatable quantities cannot be converted to float in constant expressions.
021	Void expressions are not permitted in constant expressions.
022	A structure or union is required for membership operators.
023	Attempt to apply a subscript to something other than an array or pointer.
024	Only integral values and pointers may be subtracted from pointers.
025	Pointers may only be subtracted from pointers of the same type.
026	Undeclared variable '(identifier)'.
027	Constant expressions may not have type 'void'.
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.

035	Parameter declarations are invalid with function prototypes.
036	Functions may not be initialized.
037	Task instances may not be initialized.
038	Typedefs may not be initialized.
039	Invalid initializer on function or task.
040	Array or structure initializers must be a list of constant expressions.
041	Attempt to initialize a bitfield with a relocatable value.
042	String is too long to fit into array.
043	Too many levels of braces in initializer.
044	Too many initializers.
045	Union (identifier) undefined.
046	Struct (identifier) undefined.
047	Task has more than one entrypoint.
048	File has more than one entrypoint.
049	A function exceeds 64K bytes in size.
050	Integral type expected.
051	Incompatible types.
052	Pointers must be of the same type.
053	Integral expression expected.
054	Illegal operands of minus.
055	Arithmetic types required.

056	Division by zero.
057	Division by zero prohibited.
058	Illegal types.
059	Arithmetic types expected.
060	Integral types expected.
061	The operands of the (symbol) operator have incompatible types.
062	Operands of incompatible type to '(symbol)' operator.
063	Branch condition must have scalar type.
064	Value of void function used.
065	Value of task invocation used.
066	Attempt to invoke an object which is not a function or task.
067	Argument must not be void.
068	Not enough arguments supplied in function call.
069	Too many arguments supplied in function call.
070	Unknown size.
071	Attempt to call bad function.
072	Extensive use of fast_event variables has caused a code segment to overflow it's 64K byte limit.
073	(Identifier) undeclared.
074	The left operand of the DOT operator must be of structure or union type.

075 The left operand of the -> operator must be either a pointer to a structure, or a pointer to a union.

076	(Identifier) is an unknown member.	
077	Illegal indirection or illegal subscript.	
078	Illegal L-value.	
079	Operand of prefix (symbol ++ or) operator must be scalar.	
080	Operand of postfix (symbol ++ or) operator must be scalar.	
081	Unary PLUS operator requires scalar operand.	
082	Unary minus operator requires arithmetic operand.	
083	Bitwise NOT operator requires integral operand.	
084	DEFAULT not inside SWITCH.	
085	Multiple DEFAULT's in switch.	
086	Label '(identifier)' multiply defined.	
087	BREAK outside of loop or switch.	
088	CONTINUE outside of loop.	
089	RETURN is invalid inside WAITFOR.	
090	Void functions must not return a value.	
095	Conflicting tag: (struct, union, enum, or task) (identifier) and (struct, enum, or task) (identifier).	union,
091	Expression must have type 'label'.	
092	Controlling expression must be integral.	
093	CASE expression must be integral.	
094	Duplicate CASE, value = (number).	

096	WAITFOR is invalid within another WAITFOR.
097	Attempt to wait for non-event variable.
098	Invalid L-value.
099	Attempt to modify CONST L-value.
100	Attempt to use an event value in an expression.
101	Attempt to use a label value in an expression.
102	Attempt to use a void value.
103	Arrays of functions or tasks are invalid.
104	Illegal storage class for function.
105	Illegal storage class for task instance.
106	Invalid storage class.
107	Extern variables may not be initialized within a function.
108	Function (identifier) redeclared.
109	Function (identifier) redefined.
110	(Identifier) redeclared.
111	(Identifier) redefined.
112	Typedef redefined.
113	Label '(identifier)' is undefined.
114	(1) Unknown size for (identifier).
115	(2) Unknown size for (identifier).
116	Enum (tag identifier) redeclared.

Appendix A3 Error Messages Issued by C Compiler

- 117 Newline in character constant.
- 118 Newline in string constant.
- 119 Unknown character.
- 120 Unexpected character.
- 121 Token type missing.
- 122 Wrong type of declarator for function definition.
- 123 Parameter # (number) has no identifier.
- 124 (Named item) is declared in the parameter declarations, but is not listed in the parameter list of the function.
- 125 Extra ; in function definition.
- 126 Syntax error in attribute.
- 127 Syntax error in declarator or initializer.
- 128 Syntax error in initializer.
- 129 Syntax error in parameter definition.
- 130 Attempt to initialize a formal parameter.
- 131 Syntax error in parameter.
- 132 Syntax error in struct/union declaration.
- 133 Syntax error in type specifier.
- 134 Syntax error in structure member.
- 135 Syntax error in enumerator list.
- 136 Syntax error in enumerator.

137	Syntax error in task specifier.
138	Syntax error in array subscript.
139	Syntax error in parameter list.
140	Unresolved reference (identifier).
141	Syntax error in statement.
142	Syntax error in conditional expression.
143	Syntax error in DO statement.
144	Syntax error in condition.
145	Syntax error in BREAK statement.
146	Syntax error in CONTINUE statement.
147	Syntax error in RETURN statement.
148	Syntax error in GOTO expression.
149	Syntax error in TASK list.
150	Syntax error in FOR statement.
151	Syntax error in FOR initialization.
152	Syntax error in FOR condition.
153	Syntax error in FOR increment.
154	Syntax error in SWITCH condition.
155	Syntax error in CASE expression.
156	Syntax error in compound statement.

## Table A3-1 (Continued)

A3-8

Appendix A3 Error Messages Issued by C Compiler

Table A3-1 (Continued)

and a second	· 读 / 上传法: di di di di angle re	
157	Syntax error in sizeof type name.	
158	Syntax error in subscript.	
159	Syntax error in function call.	
160	Syntax error in expression.	
161	Type clash.	
162	More than one storage class.	
163	Array of unknown size.	
164	Cannot take size of function.	
165	Structure or union of unknown size.	
166	Circular definition of enumerated type.	
167	Cannot take size of task.	
168	Struct (identifier) redeclared.	•
169	Functions and tasks may not be structure members.	
170	Zero-width bitfields may not be named.	
171	Structure member (identifier) multiply defined.	
172	Invalid negative bit-field width.	
173	(Struct, union, enum, or task ) (identifier) multiply defined.	
174	Task (identifier) redeclared.	
175	Union (identifier) redeclared.	
176	Functions and tasks may not be union members.	

177	Invalid zero-bit member.
178	Union member (identifier) multiply defined.
179	No main routine supplied.
180	Arrow operator given structure, not a pointer.
181	DOT operator given a pointer to a structure, not a structure.
182	Address of array.
183	Address of function.
184	Address of register variable.
185	Address of bit-field.
186	Address of non-L-value.
187	Attempt to use a LABEL value in an expression.
188	Attempt to use an EVENT value in an expression.
189	Invalid zero or negative array dimension.
190	Only const or volatile allowed.
191	Maximum bit-field width is 16.
192	Illegal storage class for formal parameter.
193	Function parameters may not be functions.
194	Function parameters may not be tasks.
195	Bad parameter storage class.
196	Event expression required in waitfor clause.

197	Scalars must be initialized with a single expression, optionally in braces.
198	Label undeclared.
199	Syntax error in declarator or initializer.
200	Pointers to different objects shouldn't be subtracted.
201	Duplicate formal parameters of a function.
202	External variables may not be initialized inside of a function.
203	Formal parameters of functions may not be initialized.
204	Attempt to use labels outside a function.
205	Variable (identifier) undeclared.
206	Attempt to take the value of a typedef.
207	Function's stack frame is too large.
208	Floating point has not yet been implemented.
209	Invalid conversion of relocatable quantity in constant expression.
210	Attempt to redefine the reserved name '(identifier)'.
211	CASE outside of switch statement.
212	Returned values of this size are not implemented.
213	Unrecoverable syntax error.
214	Parsing stack overflow.
215	Too many errors have been encountered during compilation.
216	Compiler aborted.

217	Register variable '(identifier)' declared with non-scalar type.
218	Implicit declaration of function '(identifier)'.
219	Out of memory during compilation-program too big.
220	Internal software error in compiler (error number). Compilation aborted.
221	No T1 Mux Installed.
222	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.
223	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.
226	Invalid or Incompatible Data Acquisition Tracks on selected playback device.
227	No ISDN Mux Installed.
230	Object file (name) is in obsolete format. Fix by recompiling it.
231	Symbol (identifier) multiply defined by object file.
232	The symbol (identifier) has been used as an event variable in one module and as a function or variable in another.
233	The symbol (identifier) has type event in one module, but has type fast_event in another.
234	Different modules have used the symbol (identifier) inconsistently as code, data, or read-only data.
235	Bad format in object file (name).
236	Cannot find object file (name).

Appendix A3 Error Messages Issued by C Compiler

## Table A3-1 (Continued)

.

700	#else inside of #else clause.
701	#elif inside of #else clause.
702	Too many nested #if's.
703	Extra tokens at end of line.
704	Unexpected end of file.
705	Identifier missing from #ifdef directive.
706	Identifier missing from #ifndef directive.
707	#elif without matching #if.
708	#else without matching #if.
709	#endif without matching #if.
710	Syntax error.
711	Syntax error in constant expression.
712	String constant in constant expression.
713	Invalid character in constant expression.
714	Error in hex number.
715	End of file in char constant.
716	Newline in char constant.
717	End of file in string.
718	Newline inside string.
719	Attempt to divide by zero.
720	Unknown preprocessor command.

•

721	Syntax error in formal parameters of macro.
722	Duplication of formal parameter (identifier) in macro definition.
723	No macro name given.
724	Macro redefined.
725	Syntax error in #line directive.
726	Unterminated string literal.
727	Cannot open include file (identifier).
728	Cannot find include file (identifier).
729	Identifier does not exist.
730	Syntax error in #include directive.
731	Include identifier is not defined.
732	Unterminated character constant.
733	End of file inside char constant.
734	End of file inside string.
735	End of file inside comment.
736	Argument list required.
737	Attempt to close bracket [ or { with ).
738	Attempt to close arg list with }.
739	Attempt to close bracket [ or ( with }.
740	Attempt to close arg list with ].

Appendix A3 Error Messages Issued by C Compiler

#### Table A3-1 (Continued)

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**

. .

ACK	Acknowledgment	
ACTLU	Activate Logical Unit (SNA)	
ADR	Address	
AK	ACK: Acknowledgment	
ANSI	American National Standards Institute	
ASCII	American Standard Code for Information Interchange, standard digital communications	code for
ASYNC	Asynchronous format (indicates START and STOP bits)	
AUX	Auxiliary	
BAUDOT	Five bit code for data transmission using one start and one stop	element; used
	in some teletypewriter machines	
BBI	Begin Bracket Indicator (SNA)	
BCC	Block Check Calculation	
BCN	Beacon (SDLC)	
BDLC	Burroughs Data Link Control	
BEL	Bell	
BERT	Bit Error Rate Testing	
BIB	Backward Indicator Bit (SS#7 Layer 2)	
BISYNC	Binary Synchronous Communications Protocol (IBM); also BSC	
BITIM	Bit image	
BL	BEL: Bell	
BLI	Blink (CRT enhancement)	
BM	Bit Mask	
BNC	A highly reliable twist-lock connector used to carry wide-band	video/digital
	signals: used with coaxial cable (G.703)	
BOP	Bit-Oriented Protocol, e.g., SDLC	
bps	Bits per second	
BS	Backspace	
BSC	BISYNC	

С	Control (X.21 signal)
CAN	Cancel
CAS	Channel Associated Signaling (G.703)
CCITT	Consultative Committee, International Telephone and Telegraph
CCS	Clear Channel Signaling (G.703)
CCSS#7	Common Channel Signaling System #7
CD	Carrier Detect (RS-232/V.24 and V.35 signal); same as RLSD
CDI	Change Direction Indicator (SNA)
CEDI	Conditional End Bracket Indicator (SNA)
CF	Command Format (SNA)
CHAR	Character
CHDAT	Character data
CIC	Circuit Identifier Code (SS#7 Layer 3)
CLR	Clear
CN	CAN: Cancel
CONF	Confirm
const	Constant, modifier to a declaration in C language
СРМ	Central Processing Module
CR	Carriage Return
CRC	Cyclic Redundancy Check
CSI	Code Selection Indicator (SNA)
CSN	Command Sequence Number (SNA)
CTS	Clear To Send (RS-232/V.24 and V.35 signal)
D	D bit (Bit 7 in first octet of packet-level X.25)
DAF	Destination Address Field (SNA)
DAT	Data Acquisition Tracks
DB-25	25-Pin D connector (standard for RS-232/V.24)
DC1	Device Control 1
DC3	Device Control 3
DCE	Data Circuit-terminating Equipment (or Data Communications
	Equipment), typically a modem
DCF	Data Count Field (SNA)
DDCMP	Digital Data Communications Message Protocol
DEC	Decrement
DEF	Destination Element Field (SNA)
#define	Preprocessor directive, C language
DEL	Delete
DFC	Data Flow Control (SNA)
DIAG	Diagnostic (X.25 Layer 3)
DIR	Directory
DISC	Disconnect (SDLC, LAPD, X.25 Layer 2)
DL	DLE: Data Link Escape; also Data Link layer (OSI primitive)
DLC	Data Link Control
DLE	Data Link Escape (used principally in transparent BISYNC)

DM	Disconnected Mode (SDLC, LAPD, X.25 Layer 2)
DMA	Direct Memory Access
DPC	Destination Point Code (SS#7 Layer 2)
DRAM	Dynamic Random Access Memory; one Mbyte of memory space of each
	MPM, dedicated to storage or receive data
DRI	Direct Response Indicator (SNA)
DSAF	Destination Subarea Address Field (SNA)
DSK	Disk
DSR	Data Set Ready (RS-232/V.24 and V.35 control lead)
DTE	Data Terminal Equipment
DTR	Data Terminal Ready (RS-232/V.24 and V.35 control lead)
DUP	Duplicate
EB	ETB, EOB: End of Transmission Block
EBCD	Extended Binary Coded Decimal
EBCDIC	Extended Binary Coded Decimal Interchange Code
EBI	End Bracket Indicator (SNA)
EC	ESC: Escape
EIA	Electronic Industries Association
#elif	Else if, preprocessor directive, C language
EM	EOM: End of Message
#endif	Preprocessor directive, C language
ENQ	Enquiry
enum	Enumeration, set of integer constants, C language
EOB	End of Transmission Block
EOF	End of Frame
EOM	End of Message
EOT	End of Transmission
EPROM	PROM containing power-up software and initialization routines
EQ	Equal
EQ	ENQ: Enquiry
ERI	Exception Response Indicator
ERN	Explicit Route Number
ESC	Escape
ET	EOT: End of Transmission
ETB	End of Transmission Block
ETX	End of Text
evar	Event variable, pre-declared identifier, ARD extension to C language
EX	ETX: End of Text
extern	External, storage class specifier, C language
FAC	Facilities (X.25 Layer 3)
FAS	Frame Alignment Signal (G.703)
FCS	Frame Check Sequence (used in BOP)
FD1	Floppy-disk Drive 1

B-3

FD2	Floppy-disk Drive 2	
FDX	Full duplex (permits simultaneous data in both directions)	
FEB	Front End Buffer	
fevar	Fast event variable, pre-declared identifier, ARD extension	to C language
FF	Form Feed	
FI	Format Indicator (SNA)	
FIB	Forward Indicator Bit (SS#7 Layer 2)	
FID	Format Identifier (SNA)	
fifo	First in, first out; memory queue on boards	
FMD	Function Management Data (SNA)	
FRMR	Frame Reject (SDLC, LAPD, X.25 Layer 2)	
FS	Field Separator	
FSN	Forward Sequence Number (SS#7 Layer 2)	
GBM	Global Bus Module	
GE	Greater than or equal to	
GFI	Group Format Indicator (X.25 Layer 3)	
goto	Jump statement, C language	
GS	Group Separator	
GT	Greater than	
01		
HDLC	High Level Data Link Control procedure	
HDX	Half duplex (data cannot be transmitted in both directions	
IIDA	simultaneously)	
HEX	Hexadecimal number; also the hex key	
HRD	Hard disk	
Hz	Hertz	
112		
I statis	Indication (X.21 signal)	
I construction	Information (SDLC, LAPD, X.25 Layer 2)	
iAPX 286	Part number for Intel 80286 processor	
	Initial Explicit Route Number (SNA)	
IERN #if	Preprocessor directive, C language	
#ifdef	If defined, preprocessor directive, C language	
	If not defined, preprocessor directive, C language	
#ifndef	Interlayer (message buffer)	
IL	Increment	
INC #in shade	Preprocessor directive, C language	
#include	Indication	
IND	Information (SDLC, LAPD, X.25 Layer 2)	
INFO		
init	Initialize Integer data type, C language	
int	Integer data type, C language Interrupt (X.25 Layer 3)	
INT	Interrupt (X.25 Layer 5)	
I/O	Input/Output International Passenger Airlines Reservation System	
IPARS	International Passenger Annues Reservation System	

B-4

.

ISDN	Integrated Services Digital Network	
ISO	International Standards Organization	
ISOC	Isochronous	
JIS	Japanese Industrial Standard	
kana	Japanese syllabic alphabet	
Kbits	Kilobits	
Kbps	Kilobits per second	
Kbyte	Kilobyte	
LAF	Local Address Field (SNA)	
LAPD	Link Access Procedure on the D-channel	
LCG	Logical Channel Group (X.25 Layer 3)	
LCO	Logical Channel Number (X.25 Layer 3)	
	그는 것 같아요. 그는 것 같아요. 그는 것 같아요. 정말 해외에 집에 가지 않는 것 같아요. 이렇게 가지 않는 것 같아요. 이렇게 하는 것 같아요. 이렇게 하는 것 같아요. 이렇게 하는 것 같아요.	
LE	Less than or equal to	
LED	Light Emitting Diode	
LF	Line Feed	
LI	Length Indicator (SS#7 Layer 2)	
LOBJ	Linkable-object	
LRC	Longitudinal Redundancy Check	
LSU	Link Status Unit (SS#7 Layer 2)	
LT	Less than	
LTA	Link Test Acknowledge	
LTM	Link Test Message	
LU	Logical Unit (SNA)	
М	M bit (X.25 Layer 3, Bit 4 of first octet)	
macro	Macro replacement of text initiated by define pre-proces	ssor directive. C
	language	
Mbyte	Megabyte	
MOD	Modulus; maximum window size for frames or packets,	8 or 128
MPM	Main Processing Module	
msec	그 것 이 가지 그는 것 같은 것 가격 방법에 가지 않았다. 그는 것 그는 것 같은 것 같은 것 같은 것 같이 가지 않는 것 같이 하는 것 같이 있다.	
MSU	Millisecond Message Signal Unit (SS#7 Layer 2)	
	이 나는 것 같은 것 같	
mux	Multiplexer	
Ν	Network layer (OSI primitive)	
NAK	Negative Acknowledgment	
NC	Network Control (SNA)	
NE	Not equal to	
NETM	Network Management (SS#7 Layer 3)	
NI	Network Indicator	
NK	NAK: Negative Acknowledgment	
NP	Network Priority (SNA)	

INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

Nr	Number (next) receive frame (SDLC, LAPD, X.25 Layer 2); N(R)	; also NR and
NRZI	Non-Return to Zero Inverted (used with SDLC and ASYNC sometimes with clocked modems)	modems-
Ns	Number (frame) sent (SDLC, LAPD, X.25 Layer 2); also NS	S and N(S)
NT	Network Termination (ISDN)	
NTR	Network Test Regular (SS#7 Layer 3)	
NTS	Network Test Special (SS#7 Layer 3)	
NU	NUL: Null	
NUL	Null	
OBJ	Object code	
OEF	Origin Element Field (SNA)	
OPC	Originating Point Code (SS#7 Layer 3)	
OSAF	Origin Subarea Address Field (SNA)	
OSI	Open Systems Interconnection	
OUTSYNC	Out of synchronization	
	tipe ii	
pad	DEL or idle line character	
pal	Programmable array logic	
parens	Parentheses	
PCM	Peripheral Control Module	
PDU	Primitive Data Unit	
PERC	Percentage and believe and a second and a se	
P/F	Poll/Final bit used in control byte at frame level (SDLC, X.2	5)
PH	Physical layer (OSI primitive)	• )
PI	Pacing Indicator (SNA)	
PIU	Path Information Unit (SNA)	
PLU	Primary Logical Unit (SNA)	
	Position	
pos Pr	Packet (next) receive sequence number (X.25 Layer 3); also	PR and P(R)
	Preprocessor directive, C language and the set of the	
#pragma PRGM	Program	
PROG TR	Program Trace Run-mode screen	
PROM	Programmable Read-Only Memory	
Ps	Packet send sequence number (X.25 Layer 3); also PS and	P(S)
PU	Physical Unit (SNA)	(0)
FU	Enysical Onic (SIVA) Escure en (EIC), como situation	
0	Q bit (Bit 8 of first octet in packet-level X.25)	
Q	Queued Response Indicator (SNA) for the biometric	
QRI	Guened Response Indicator (SIVR) some site and sold	
R	Receive (X.21 signal) (1988) Journey and About 20	
R RAM	Random Access Memory	
	Received Data (RS-232/V.24 and V.35 signal)	
RD	Registration (X.25 Layer 3) (Additional and Strategy 2017)	9Z
REG	registration (r.25 Layer 5) to the contract of the second	

REJ	Reject (SDLC, LAPD, X.25 Layer 2)
REQ	Request
RESP	Response
Rev	Reverse
RGB	Red Green Blue (connector for color monitor)
RH	Request/Response Header (SNA)
RJ-11C	Standard for common telephone jack
RLSD	Received Line Signal Detect (RS-232/V.24 signal); same as CD: Carrier Detect
RNR	Receive Not-Ready (SDLC, LAPD, X.25 Layer 2, X.25 Layer 3)
ROM	Read-Only Memory (firmware/software storage)
RR	Receive Ready (SDLC, LAPD, X.25 Layer 2, X.25 Layer 3)
RS	Record Separator
RS-232/V.24	List of definitions for interchange circuit between data terminal equipment and
	data circuit termination equipment established by EIA
RS-449	EIA standard for 37-pin and 9-pin DTE-DCE interface
RTI	Response Type Indicator (SNA)
RTS	Request To Send (RS-232/V.24 and V.35 signal)
RU	Request/Response Unit (SNA)
SABM	Set Asynchronous Balanced Mode (LAPD, X.25 Layer 2)
SABME	Set Asynchronous Balanced Mode Extended (LAPD, X.25 Layer 2)
SAPI	Service Access Point Identifier (LAPD)
SB	SUB: Substitute
SC	Session Control (SNA)
SCCP	Signalling Connection Control Part (SS#7 Layer 3)
SCR	Signal Clock Receive (RS-232/V.24 and V.35 signal), used when DCE clock
	drives DTE
SCT	Signal Clock Transmit (RS-232/V.24 and V.35 signal), used when DCE clock drives DTE
SCTE	Signal Clock Transmit External (RS-232/V.24 and V.35 signal), used when
	DTE clock drives DCE
SDI	Sense Data Indicator (SNA)
SDLC	Synchronous Data Link Control (IBM)
SDU	Service Data Unit
SELECTRIC	IBM typewriter/printer code
SFO	Status Field Octet (SS#7 Layer 2)
SH	SOH: Start of Header
SI	Shift In
SIO	Sequenced Information Frame 0 (LAPD)
SI1	Sequenced Information Frame 1 (LAPD)
SIG	Signal
SIO	Service Information Octet (SS#7 Layer 3)
SLS	Signalling Link Selection (SS#7 Layer 3)
SLU	Secondary Logical Unit (SNA)
	- 특별 가지가 있는 것 같은 것 같은 것을 가지 않는 것을 수 있는 것이 가지 않는 것이 있는 것이 없는 것이 있는 것이 없는 것이 없다. 것이 없는 것이 없다. 것이 없는 것이 없다. 것이 없는 것이 없다. 것이 없는 것이 없다. 것이 없는 것이 없다. 것이 없는 것이 없는 것이 없는 것이 없는 것이 없는 것이 없다. 것이 없는 것이 없는 것이 없는 것이 없는 것이 없는 것이 없다. 것이 않은 것이 없는 것이 없는 것이 없는 것이 없다. 것이 없는 것이 없는 것이 없는 것이 없는 것이 없다. 것이 없는 것이 없는 것이 없는 것이 없는 것이 없는 것이 없는 것이 없다. 것이 없는 것이 없다. 것이 없는 것이 없는 것이 없는 것이 없는 것이 없다. 것이 없는 것이 없는 것이 없는 것이 없는 것이 없다. 것이 없는 것이 없는 것이 없다. 것이 않은 것이 없는 것이 없다. 것이 않은 것이 없는 것이 없다. 것이 없는 것이 없는 것이 없다. 것이 없는 것이 없는 것이 없는 것이 없다. 것이 없는 것이 없는 것이 없다. 것이 않은 것이 없는 것이 없다. 것이 않은 것 않이

SMP	Sample Sample	
SNA	System Network Architecture (IBM)	
SNAI	SNA Indicator (SNA)	
SNF	Sequence Number Field (SNA)	
SNRM	Set Normal Response Mode (SDLC)	
SO	Shift Out	1997
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 r	eferenced under
	the same name, C language	
STR	String and the constraint and creation that the	
STX	Start of Text	
SUB	Substitute	
SX	STX: Start of Text	
SY	SYN: Synchronization character	
SYN	Synchronization character	
SYS	System file	
sys	System directory	
	2007.10.2018年1月,2018年1月,1月18年1月。 1997.191.2018年1月,2018年1月,1月18年1月。	
Т	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	. The Constant
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 exis	ting data type, C
<i>.</i>	language	
	· 计标题	
UA	Unnumbered Acknowledgment (SDLC, LAPD, X.25 Layer	r 2) - 072
UI	Unnumbered Information (SDLC)	
UL	Underwriters' Laboratory	
#undef	Undefine, preprocessor directive, C language	
US	Unit Separator receiption 260 compared plant, units and	
usec	Microsecond	

.
USER TR usr	Run-mode User Trace screen User directory
VRCWI	Virtual Route Change Window Indicator (SNA)
VRCWRI	Virtual Route Change Window Reply Indicator (SNA)
VRID	Virtual Route Identifier (SNA)
VRN	Virtual Route Number (SNA)
VRPCI	Virtual Route Pacing Count Indicator (SNA)
VRPRQ	Virtual Route Pacing Request (SNA)
VRPRS	Virtual Route Pacing Response (SNA)
VRRWI	Virtual Route Reset Window Indicator (SNA)
VRSI	Virtual Route Support Indicator (SNA)
VRSSN	Virtual Route Send Sequence Number (SNA)
VRSTI	Virtual Route Sequence and Type Indicator (SNA)
VT	Vertical Tab
X.21	CCITT recommendation governing synchronous DTE-DCE operation on public data networks
X.25	CCITT recommendation governing the packet mode link connecting the user site with a public data network
XDRAM	Extended Dynamic Random Access Memory
XEQ	Execute
XFER	Transfer
XID	Exchange Identification (SDLC)
XMIT	Transmit, transmission

INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

Her Henrich and H

#### **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:

X1 = 3686400/2 X2 = 4096000/2 X3 = 4608000/2 X4 = 5376000/2

Divide the desired bps rate into each of the values of X to produce result Y.

Y1 = X1/bps Y2 = X2/bps Y3 = X3/bps Y4 = X4/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:

X1/Y1 = SPEED1 X2/Y2 = SPEED2 X3/Y3 = SPEED3 X4/Y4 = 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.

168000.0	12923.08	6582.9	4421.05	3339.1	2679.07	2215.38
144000.0	12800.0	6545.45	4413.79	3294.12	2666.67	2210.53
128000.0	12126.3	6461.54	4363.64	3291.43	2648.3	2206.810
115200.0	12000.0	6400.0	4347.2	3282.05	2625.0	- 2194.3
84000.0	11636.36	6260.87	4307.69	3272.73	2618.18	2181.82
76800.0	11520.0	6227.0	4266.67	3245.1	2612.24	2173.58
72000.0	11200.0	6222.22	4235.29	3230.77	2588.8	2169.49
64000.0	11076.92	6095.24	4200.0	3200.0	2584.62	2153.85
57600.0	10971.4	6063.16	4189.1	3169.81	2571.43	2149.25
56000.0	10666.67	6000.0	4129.03	3156.2	2560.0	2133.33
48000.0	10500.0	5907.7	4114.29	3130.43	2545.45	2126.58
46080.0	10472.73	5818.18	4097.56	3121.95	2526.32	2117.65
42666.67	10285.71	5793.10	4042.1	3113.51	2509.80	2113.8
42000.0	10017.4	5760.0	4000.0	3111.11	2507.46	2100.0
38400.0	9882.35	5619.5	3972.41	3072.0	2504.35	2098.36
36000.0	9846.15	5600.0	3906.98	3063.83	2482.76	2094.55
33600.0	9600.0	5565.22	3905.1	3054.55	2477.4	2086.96
32914.3	9333.33	5538.46	3891.89	3047.62	2470.59	2075.7
32000.0	9216.0	5485.71	3878.79	3031.58	2461.54	2074.07
28800.0	9142.86	5419.35	3840.0	3000.0	2451.06	2064.52
28000.0	9000.0	5358.1	3818.18	2992.2	2440.68	2057.14
25600.0	8861.54	5333.33	3789.47	2976.74	2434.78	2048.78
24000.0	8842.11	5250.0	3777.0	2953.85	2425.3	2038.9
23040.0	8533.33	5236.36	3764.71	2947.37	2415.09	2031.75
21333.33	8470.59	5142.86	3733.33	2938.78	2400.0	2028.17
21000.0	8400.0	5120.0	3716.13	2916.5	2375.3	2024.010
20945.5	8228.57	5090.91	3692.31	2909.09	2370.37	2021.05
20571.43	8000.0	5008.610	3657.14	2896.55	2366.110	2003.5
19200.0	7944.8	4965.52	3652.17	2880.0	2360.66	2000.0
18666.67	7680.0	4941.18	3600.0	2847.46	2351.02	1986.21
18285.71	7636.36	4923.08	3574.47	2844.44	2333.33	1976.47
18000.0	7578.95	4902.1	3555.56	2823.53	2327.27	1972.60
17723.1	7529.41	4800.0	3544.6	2809.76	2322.58	1969.23
16800.0	7432.3	4740.74	3512.110	2800.0	2304.0	1953.49
16457.14	7304.35	4702.0	3500.0	2782.61	2301.37	1952.54
16000.0	7200.0	4666.67	3490.91	2775.9	2285.71	1945.95
15360.0	7111.11	4645.16	3459.46	2769.23	2281.2	1939.39
15272.73	7000.0	4608.0	3438.8	2754.010	2270.27	1931.03
14400.0	6981.8	4571.43	3428.57	2742.86	2258.82	1920.0
14222.22	6857.14	4540.54	3388.24	2723.40	2250.0	1910.45
14222.22	6776.47	4517.6	3368.42	2716.98	2245.61	1909.09
13552.9	6736.84	4500.0	3360.0	2710.6	2240.0	1904.1
13090.91	6720.0	4430.77	3348.84	2709.68	2236.9	1894.74

INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

		Ta	ble C-1 (cont	inued)		an an ann an ann an Annaichtean a' fhair
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	1312.50	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
828.57	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 Pt 2	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

Appendix C Selectable Data Speeds

C−3

		्ार्ध	ole C-1 (contin	nued)	A subjective we can be a set of the	na na mana na m Na mana na mana n
954.55	893.02	835.82	786.89	743.23	702.93	669.32 <sup>88</sup>
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 ACOL	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 GOO	794.48	748.54	709.36	674.610	642.86
900.0	842.11	792.45	748.05	708.86	673.68	641.8
898.310	840.88 9201	791.8	746.67	707.18	672.810	641.22
896.5	840.0°. 2001	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

INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

				•		
		<b>.</b>	able C-1 (cont	inued)		
636.82	608.610	581.82	559.22	538.32	517.8	498.52
636.46	607.9	581.31	558.95	537.82	516.92	498.27
636.36	607.59	580.65	558.14	537.31	516.59	498.05
634.7	606.64	580.4	557.9	537.1	516.13	497.6
634.36	606.410	579.31	556.52	536.74	515.4	497.04
633.96	606.32	579.19	556.29	535.81	515.34	496.55
633.66	605.04	578.89	555.98	535.56	514.29	496.6
633.0	604.7	578.31	555.2	535.32	514.06	496.12
632.97	604.32	577.4	554.46	535.03	513.76	495.58
631.58	603.77	577.32	554.11	534.6	513.1	495.5
631.2	603.14	576.58	553.85	533.33	512.46	494.85
630.54	602.51	576.0	552.63	532.1	512.110	494.42
629.51	602.15	575.34	552.5	531.65	512.0	494.21
629.21	601.6	574.6	551.72	531.37	510.9	494.12
628.82	600.94	573.99	551.2	531.12	510.64	493.4
627.8	600.0	573.71	551.110	530.88	509.96	493.15
627.45	598.4	573.38	550.82	529.97	509.73	492.67
626.87	598.13	573.13	549.9	529.7	509.09	492.31
626.09	597.86	571.7	549.62	529.41	508.83	491.47
624.54	597.51	571.43	549.36	528.93	508.6	491.3
624.39	596.89	570.3	549.02	528.44	507.94	491.23
623.38	595.74	570.210	548.57	528.30	507.55	490.42
622.70	595.35	569.49	547.53	527.47	507.49	490.21
622.22	595.04	569.17	547.3	527.2	507.04	489.710
621.36	593.81	568.89	547.23	526.75	506.4	489.2
621.0	593.64	567.57	547.01	526.65	506.02	488.55
620.69	592.59	567.49	546.0	526.03	505.93	488.37
619.93	592.3	566.93	545.97	525.55	505.26	488.14
619.35	591.55	566.37	545.45	525.0	504.50	486.96
617.7	590.77	566.1	544.68	524.8	504.2	486.69
618.36	590.16	565.66	543.69	524.59	503.94	486.49
618.03	589.86	564.71	543.4	523.64	503.410	486.08
617.65	589.47	563.88	543.310	523.36	503.1	485.55
616.04	589.3	563.76	542.37	522.45	503.06	484.85
615.38	587.76	563.3	542.1	521.74	502.99	484.15
614.4	587.41	562.50	541.94	521.27	502.0	484.03
613.14	587.16	562.0	541.35	520.33	501.96	483.22
612.77	586.3	561.95	540.85	520.12	501.74	483.02
612.44	585.37	561.87	540.19	519.86	501.49	482.76
611.1	584.77	561.40	540.08	518.92	500.87	482.01
610.91	584.47	560.31	539.6	518.52	500.0	481.61
610.17	583.33	560.6	539.33	518.22	499.8	481.38
609.52	582.910	560.0	538.46	517.99	498.70	481.20

۰.

		Tab	ole C-1 (contir	nued)		
481.0	464.09	449.12	435.23	422.29	407.93	393.85
480.0	463.77	449.110	435.05	422.11	407.77	393.44
479.40	463.6	448.510	434.72	422.0	407.64	393.17
479.0	463.02	448.25	434.11	421.98	407.07	392.64
478.63	462.81	448.2	433.9	421.2	406.78	392.52
478.41	462.7	448.0	433.810	421.05	406.35	392.37
478.01	462.65	447.55	433.73	420.44	405.710	391.84
477.61	462.09	447.4	433.08	420.0	405.63	391.61
477.27	461.7	447.20	432.99	419.83	405.06	391.44
477.0	461.54	446.81	432.43	419.67	404.82	391.30
476.82	460.80	446.51	432.3	418.95	404.49	390.610
476.03	460.43	445.99	431.88	418.91	404.21	390.51
475.92	460.27	445.82	431.46	418.9	403.85	390.24
475.84	460.06	445.62	431.14	418.60	403.79	389.79
475.25	459.9	444.79	430.98	418.30	403.36	389.19
475.1	459.02	444.44	430.77	418.1	402.88	389.06
474.58	458.96	443.9	430.7	417.91	402.710	388.89
474.07	458.78	443.27	429.9	417.39	402.52	388.14
473.68	458.510	443.08	429.85	416.94	402.23	387.99
473.24	<b>458.1</b>	442.91	429.67	416.87	401.91	387.88
473.1	457.77	442.2	429.53	416.18	401.39	387.010
472.32	457.14	442.11	429.1	415.88	401.25	386.71
472.13	456.52	441.72	428.57	415.84	401.11	386.58
471.91	456.2	441.38	428.3	415.58	400.95	386.21
471.2	455.610	440.94	428.25	414.99	400.0	386.06
470.59	455.52	440.5	428.09	414.81	399.05	385.54
470.20	455.34	440.37	427.48	414.39	398.89	385.32
469.27	455.28	439.86	427.210	414.24	398.75	385.28
469.2	454.4	439.79	426.67	413.79	398.62	385.03
469.06	454.26	439.69	426.310	412.90	398.10	384.44
468.86	454.05	439.02	426.04	412.78	397.79	384.38
468.29	453.90	438.9	425.9	412.61	397.52	384.0
467.97	453.54	438.64	425.32	411.76	397.24	383.56
467.53	452.83	438.36	425.25	411.58	397.16	383.23
467.15	452.7	438.02	425.09	411.43	396.69	382.98
466.67	452.210	438.0	424.78	410.76	396.28	382.72
466.4	451.76	437.69	424.3	410.26	396.23	382.69
466.310	451.61	437.50	424.24	409.96	395.88	382.09
466.02	451.41	437.2	423.84	409.76	395.60	381.96
465.5	450.9	436.86	423.53	409.09	395.29	381.82
465.45	450.70	436.36	423.17	408.95	395.06	381.46
465.37	450.40	435.5	422.8	408.76	394.52	380.95
464.52	450.0	435.37	422.44	408.51	394.37	380.110

INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

		T	able C-1 (con	tinued)		
						an ann an tar thatairte an
380.09	367.61	355.93	344.91	334.20	324.32	315.110
379.95	367.35	355.56	344.410	334.11	324.05	315.010
379.82	366.88	355.18	344.26	333.91	323.610	314.75
379.23	366.81	354.68	344.09	333.33	323.510	314.61
378.95	366.76	354.57	343.88	332.95	323.23	314.410
378.610	366.41	354.46	343.68	332.67	323.08	314.02
378.38	366.01	354.43	343.56	332.56	322.87	313.810
377.95	365.71	353.81	343.16	332.47	322.69	313.73
377.70	365.48	353.68	342.86	332.02	322.46	313.43
377.58	365.22	353.59	342.25	331.99	322.42	313.04
377.53	364.67	353.37	342.16	331.710	322.15	312.96
376.96	364.56	352.94	342.04	331.61	321.84	312.85
376.68	364.43	352.62	341.84	331.36	321.79	312.36
376.47	363.64	352.29	341.46	331.03	321.61	312.27
375.98	363.41	352.20	341.33	330.75	321.43	312.110
375.84	362.85	352.08	341.23	330.71	321.22	311.69
375.37	362.72	351.65	340.83	330.28	320.89	311.44
375.24	362.61	351.46	340.77	330.09	320.80	311.35
375.0	362.26	351.22	340.43	330.06	320.71	311.11
374.27	362.07	350.73	340.08	329.810	320.61	311.02
374.16	361.81	350.68	339.82	329.52	320.0	310.68
374.03	361.58	350.36	339.62	329.41	319.39	310.54
373.33	361.29	350.15	339.52	329.14	319.29	310.51
373.18	361.13	350.0	339.39	329.05	319.20	310.34
373.06	360.90	349.73	338.82	328.77	319.11	309.96
372.82	360.56	349.51	338.71	328.21	318.79	- 309.93
372.51	360.52	349.27	338.62	328.13	318.58	309.68
372.09	360.0	349.09	338.03	328.02	318.41	309.39
371.68	359.74	348.77	337.83	327.49	318.23	309.18
371.61	359.55	348.67	337.73	327.37	318.18	309.01
371.13	359.10	348.55	337.35	327.27	317.88	308.85
371.01	358.97	348.04	337.24	326.85	317.62	308.82
370.86	358.88	347.83	336.84	326.53	317.58	308.43
370.42	358.54	347.11	336.67	326.35	317.36	308.35
370.18	358.21	346.99	336.45	326.21	317.18	308.26
370.04	357.76	346.88	336.0	325.79	316.98	308.02
369.94	357.54	346.39	335.96	325.610	316.83	307.69
369.23	357.45	346.15	335.86	325.58	316.48	307.20
368.88	357.32	345.95	335.66	325.42	316.38	307.13
368.42	356.69	345.68	335.33	325.06	316.05	307.04
368.29	356.66	345.32	335.08	324.95	315.79	306.95
368.05	356.55	345.01	334.88	324.87	315.62	306.57
367.82	356.44	344.97	334.66	324.51	315.27	306.38

C

(

			ble C-1 (conti	nued)		
306.22	303.710	301.18	150.0	100.0	<b>45.0</b>	16.0 PD 07 5
306.01	303.32	301.08	144.0	96.0	42.0	15.0
305.73	303.25	300.0	140.0	90.0	40.0	14.0
305.57	303.16	288.0	134.5	84.0	36.0	12.0
305.49	302.70	280.0	128.0	80.0	35.0	10.0 80.000
305.45	302.60	256.0	125.0	<b>75.0</b>	32.0	9.0 0.810
305.08	302.52	210.0	120.0	72.0	30.0	8.0 88 87 8
304.90	302.36	200.0	250.0	70.0	28.0	7.0 20.000
304.76	302.16	192.0	240.0	64.0	25.0	6.0
304.44	301.89	180.0	225.0	60.0	24.0	5.0
304.35	301.62	175.0	224.0	56.0	21.0	4.0
304.04 303.96	301.57 301.26	168.0 160.0	112.0 105.0	50.0 48.0	20.0 18.0	3.0 2.0
	in gravit Article Problem				in an	
			<ul> <li>株式201</li> <li>・・・・や おおく</li> </ul>			
지지 (11) 10 11 - 11 - 11 - 11 - 11 - 11 - 11 - 1						
	8.0					
6 6 - AV				ea, tee		
						· 新闻,名 教
						11. 二次表示学者要

INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

Appendix D Code Charts

## Appendix D: Code Charts

INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

Appendix 0: Code Charts

.

Appendix D1 Code Charts: Keyboard-to-Hex-Translation

# Appendix D1: Keyboard-to-Hex Translation

KEY	UNSHIFT CHAR <sup>1</sup>	FED HEX <sup>2</sup>	SHIFTE CHAR <sup>1</sup>	D HEX <sup>2</sup>	· · · · · · · · · · · · · · · · · · ·	CONT CHAR <sup>1</sup>	FROL HEX <sup>2</sup>
<b>⋖₿∁</b> Ѻ⋓⋤∁ <b>╂</b> ∟ј₭ <b>」</b> ጆઽѺ₽∁⋢⋈⋤⋃⋗⋛╳⋎⋈отаз45678	abcdefghijk  Mnopqrstuvwxyz012345678	8123345678912345678923456789012345678	<b>АВСДШҢӨН</b> ―ЈК <b>Ј</b> Х <b>20</b> РФ <b>г</b> め⊢ <b>Ј&gt;ЎХ</b> ≻N <i>→</i> @ <b>#\$%</b> - &*	C1223456678912345667892234566789DACBBC - 55		SOH STX EOQ ACEL BS HT LF FCR SI DC1 DC2 SI DC2 DC4 SYN SVN SVN SUB Sa CAN Sa CAN SA Sa CAN SA	
9 dāsh = `` ] ; ; , Space	9 dāsh = \ - ; ; , , , , , , , , , , , , , , , , ,	F9 60 7E 5E 5D 6B 4B 61 40	( underline + { } : ? Space	4D 4E 4E 4E 4E 4C 7A 7F 4E 6F 40 40		7 - DEL ESC NUL GS RS US FS Space	A1 79 07 27 00 1D 35 1F 22 40

INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

Table D1-1 Keyboard-to-EBCDIC

Untranslatable characters ("-" in the above table) that are entered in transmit strings will be replaced by NULL (hex 00) during transmission.

<sup>1</sup>CHAR displayed in Run mode

<sup>2</sup>HEX byte trapped/transmitted

<sup>3</sup>Enter the hex value for the  $\$  character.

Appendix D1 Code Charts: Keyboard-to-Hex-Translation

KEY	UNSHI CHAR <sup>1</sup>	HEX2	SHIFT CHAR <sup>1</sup>	ED HEX <sup>2</sup>	CONT CHAR <sup>1</sup>	ROL HEX
<b>⋖₿∁</b> ŪШĻŨĬ <sub>─</sub> JKJZZO₽Ø£%FJ>¥XYN0	a b c d e f gh i j k i ĥ n o p q r s t u v ¥ x y z o	61 62 63 64 65 66 67 68 66 66 66 66 66 66 66 70 71 73 74 75 76 77 78 79 70	<b>⋖₿∁</b> Ѻш⊩Ũ⊥_jĸ⊥≊≳Ѻ₽Ⴓҝѹ⊦∋>∛хүх→	41 42 43 445 46 47 48 45 46 47 48 47 48 40 40 40 51 23 45 55 55 55 55 55 55 55 55 55 55 55 55	SOH STX EDT ENQ ACK BEL BS HT LF VT FF CR SO SI DLE DC2 DC3 DC4 NAK SYN ECAN EM SUB	01 02 03 05 06 07 08 09 0A 0B 0C 0DE 0F 10 11 12 13 14 15 16 17 18 19 1A
1	1	7A 30 31 32 33		5A 29 21 40 23 24 25 5E 26	Sa	me
2 3 4 5 6 7 8	2 3 4 5 6 7 8 9	34 35 36 37 38 39 2D	@ # \$ ~	24 25 5E	8	S
7 8 9	7 8 9	37 38 39	& • •	26 2A 28 5F	Unst	nifted
g dāsh = \3 [ ] ; , , , , , , , , , , , , , , , , , ,	9 dāsh = [ ] ; ; Space	39 23D 55B 53B 27 2C 2E 20	( underline + - - - - - - - - - - - - - - - - - -	<ul> <li>20</li> <li>20</li> <li>20</li> <li>20</li> <li>20</li> <li>20</li> <li>20</li> </ul>	- DEL ESC NUL GS RS US FS Space	7E 60 7F 1B 00 1D 1E 1C 20

Table D1-2 Keyboard-to-ASCII

<sup>2</sup>HEX byte trapped/transmitted (space parity) <sup>3</sup>Enter the hex value for the  $\$  character.

un finition de la génération de la constant de la seconda de la seconda de la seconda de la seconda de la secon Seconda de la generation de la seconda de

	UN	ISHIFTED		naar oog Dan waato S	HIFTED		CON	TROL
KEY as a more	LOWER(\) <sup>1</sup>	UPPER(^) <sup>2</sup>	HEX <sup>3</sup>	LOWER(\) <sup>1</sup>	UPPER(^) <sup>2</sup>	HEX <sup>3</sup>	CHAR <sup>4</sup>	HEX3
A B C D	a b c d	A B C D	23 13 73 0B	a b c d	ABCD	23 13 73 0B	SOH - EOT	3E - 7C
<b>EFGH-JKLXZOPQRSTU&gt;`XYZO</b>	efghijki Enopqrstuvwxyzo	ш⊩ŨI_JKJZZO₽Q⊄%FJ>≷XYN~	6B 5B 3B 07 67 61 51 31 49 19 79 455 52 32 4A 2A 7A 26 54	e f ghijk   Mnopqrstuvwxyz0\$	<b>⋓</b> ⊩ ゚゚゚゚゚゚゚゚゚゚゚゚゚゚゚゚゚゚゚゚゚゚゚゚゚゚゚゚゚゚゚゚゚゚゚゚	6B 5B 3B 07 61 51 51 51 51 29 79 45 52 32 4A 24 7A 46 54	BS HT LF CR DC2 DC4 SYN ETB	
1 2 3 4 5 6 7	1 2 3 4 5 6	= <;;.%	20 10 70 08 68 58	\$ @# \$ 5	! 02 ! %	75 02 34 75 68 1C	Sar	
8 9	9.57 6.8 5.79		38 04 64	& 8 9		43 04 64	Unsh	ifted
dāsh = \6	dāsh 1	0 ∂ dāsh = \	01 20 1F	- &	- + -	43	-	-
Ì		84- 84-	 	-		-	DEL -	7F -
;	3		70 58	4 #	: ;	08 34	in the second	-
,	050	76 37	76 37	2	< >	10 38	RS	20
/ Space	; Space	37 ? Space	62 40	/ / Space	? Space	50 62 40	- 	-

Table D1-3 Keyboard-to-EBCD

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  $\setminus$  character.

Appendix D1 Code Charts: Keyboard-to-Hex-Translation

ΈY	UNSHII CHAR <sup>1</sup>	HEX <sup>2</sup>	SHIF CHAR <sup>1</sup>	TED HEX <sup>2</sup>	CON1 CHAR <sup>1</sup>	ROL HEX <sup>2</sup>
ABCDEFGH	Ą	54 15	Ą	54 15 16 57	-	-
в С	В С	15 16	B	15 16		
Ď	Ď	16 57 58	Ď	57	신물이다.	
E F	E	58 19	Ē	58 19		1953
G	A B C D E F G H	19 1A 5B 1C 64	<b>АВСДЕРОН</b>	iĂ	성 : 특별 : 이 : 1	Ξ.
H	H	5B 1C	H	5B	사람이 공 <b>식</b> 정이 가지요. - 1997년 2017년 - 1997	-
J	J	64		64	2012 특히 안전.	<u>-</u>
K	ĸ	25	K	25	1997년 <b>- 1</b> 997년 1997년 1997년 - 1997년 19	-
M	Ā	67	יארזעבסה <u>ס</u> בארח>	67	-	
N O	N	68 29	N	68		- 4M
P	P	ŽĂ	Ř	2A		2.4
Q	Q	6B	Q	6B		1
S	Š	75	S	75		학교 및
T	Ţ	76 97	Ţ	76 97		-
v	v	38	v	38		2013 <b>-</b> 202 1911 <b>-</b> 36
Ŵ	ŵ	79 7 A	W	79 7 A		는 것을 같은
Ŷ	Ŷ	3B	Ŷ	3B	이 귀구 아파 요리	0120
Z O	MZOPQRSTU>¥xyzo	26 67 68 29 2A 6B 2C 75 76 37 38 79 7A 38 79 7A 38 70 43 04	W Y Z	7C 3D	-	-
JKLMNOPQRSTUVWXYZ0123456789	1 2 3	45 46	! @ #	1A 5B 1C 62 26 67 68 20 76 76 37 38 7A 3B 7A 3D 20 F 20 F 20 F 20 73	Sar	ne
4 5 6	2 3 4 5 6 7 8 9	07 08 49 -	@ # \$ ~	62 6D 2F	as	
7 8	7 8	4A 0B	& *	73 61	Unshi	fted
		4C	(	31	0//3///	1160
dāsh =	dāsh =	02 5D	ùnderlin +	31 Ne 70 10		
<b>\</b> 3	$\sim 10^{-1}$	4A 0B 4C 02 5D 0D 4F				-
ļ	la III	4F 01	이 방법에는 물건했다. 사람이 이 물건이 되			-
1 (1) ;		ŎĖ		51	이 옷을 다 봐요.	- <b>-</b>
		6E		7F		-
•		01 0E 6E 32 52	< > . ?	51 7F 5E 3E 13 40		<b>.</b> .
1	1	34 40	?	13	-	40

Table D1-4 Keyboard-to-XS-3 (SYN=35; EOM=55)

Untranslatable characters ("-" in the above tables) that are entered in transmit strings will be replaced by NULL (hex 00) during transmission.

<sup>1</sup>CHAR displayed in Run mode

<sup>2</sup>HEX byte trapped/transmitted (odd parity)

<sup>3</sup>Enter the hex value for the  $\$  character.

A       A       31       A       31       -       -         B       B       32       B       32       -       -       -         C       O       334       D       334       D       334       -       -       -         D       D       345       D       344       D       345       -       -       -         E       E       356       F       366       F       367       -	KEY (	UNSHIFTED CHAR <sup>1</sup> HEX <sup>2</sup>	SHIF CHAR <sup>1</sup>	TED HEX <sup>2</sup>	CONT CHAR <sup>1</sup>	ROL HEX <sup>2</sup>
a 10 base 10 base 10	I JKLMNOPQRSTUVWXYZ0 123456789 dash	dāsh 1A = 0E  [ 1E    , 1F - 3B 	JKLMNOPQRSTU>WXYZ)-@#\$%*(-+*<	22 23 24 25 26 27 28 29 12 13 14 156 17 18 19 2 20 18 30 2 - 0 B 30 2 - 0 B 2 - 0 B 30 2 - 0 - 2 A 2 - 0 - 2 A 2 - 0 - 2 A 2 - 0 - 0 - 0 - 0 - 0 - 0 - 0 - 0 - 0 - 0	- - - - - - - - - - - - - - - - - - -	

Table D1-5 Keyboard-to-IPARS

<sup>1</sup>CHAR displayed in Run mode <sup>2</sup>HEX byte trapped/transmitted tes e otte add dd for "S englandedia' Affar (1990) affa en olaefau (60 gaals of 1991) yn honediade an ferr

show not at benefits 2.440

Aliang Divis) Barris and an international and a list?

Appendix D1 Code Charts: Keyboard-to-Hex-Translation

a b c d e f g h i j k l m n o p q r s t u v w x	UPPER(^) <sup>2</sup> ABCDEFGHIJKLMNOPQRSTUVWX	HEX <sup>3</sup> 31 32 73 34 75 76 37 87 9 61 62 23 64 25 66 89 22 13 54 51 51 51 51 51 51 51 51 51 51 51 51 51	LOWER(N) <sup>1</sup> a b c d e f f g h i j k l m n o p q r s t u v	<b>АВСОШНѼĬ</b> ╵ӮҞ <b>╵</b> ѮЅѺҎѼѤ҄Ѡ҅ҤӬჂ	HEX <sup>3</sup> 31 32 73 34 75 76 37 38 79 61 62 23 64 25 26 67 68 29 52 13 54 15	LOWER(\) <sup>1</sup> EOT BS HT LF - FF CR	UPPER(^) <sup>2</sup> - EOT - - BS HT LF - FF CR - - - - - - - - - - - - - - - -	HEX 
bcdefghijki Enopqrstuvw	<b>−→∀−1≤20</b> ₽0₽0₽₩₩	32 73 37 76 37 89 62 34 56 23 62 26 68 92 34 51 62 26 68 92 34 51 62 26 68 92 51 51 51 51 51 51 51 51 51 51 51 51 51	abcdef ghijk, Mnopqrstuv	<b>-                                    </b>	32 73 34 75 76 37 87 9 61 62 23 64 25 26 67 68 29 52 13 54 15	- BS HT LF - FF	- - BS HT LF - FF	- 6E 3D 5D 60 - - -
cdef ghijki Ê no pqrstu v V	<b>−→∀−1≤20</b> ₽0₽0₽₩₩	34 756 378 762 38 791 623 623 623 668 292 345 16 251 345 16	c d e f gh i j k i m o p q r s t u v	<b>-                                    </b>	34 75 76 37 38 79 61 62 23 64 25 26 67 68 29 52 13 54 15	- BS HT LF - FF	- - BS HT LF - FF	- 6E 3D 5D 60 - - -
efghijkl Enopqrstuvw	<b>−→∀−1≤20</b> ₽0₽0₽₩₩	34 756 378 762 38 791 623 623 623 668 292 345 16 251 345 16	ef ghijkl Mnopqr'stuv	<b>-                                    </b>	34 75 76 37 38 79 61 62 23 64 25 26 67 68 29 52 13 54 15	- BS HT LF - FF	- - BS HT LF - FF	- 6E 3D 5D 60 - - -
f ghijki Mnopqrstuvw	<b>−→∀−1≤20</b> ₽0₽0₽₩₩	76 37 38 76 23 62 26 68 25 26 68 25 23 45 68 25 23 45 15 15	f gh     k   Enopgristuv	<b>-                                    </b>	76 37 38 79 61 62 23 64 25 26 67 68 29 52 13 54 15		HT LF - FF	3D 5D 0D 60 - - - - - - -
i jkl mnopqrstuvw	<b>−→∀−1≤20</b> ₽0₽0₽₩₩	38 79 612 23 623 67 68 292 513 515 15 15	h i j k l M n o p q r s t u v	<b>-                                    </b>	38 79 61 23 64 25 26 67 68 29 52 13 54 15		HT LF - FF	3D 5D 0D 60 - - - - - - - -
i jkl mnopqrstuvw	<b>−→∀−1≤20</b> ₽0₽0₽₩₩	79 61 234 267 689 252 134 516 156	i jk i Mnopqrstuv	<b>-                                    </b>	79 61 62 23 64 25 26 67 68 29 52 13 54 15		HT LF - FF	3D 5D 0D 60 - - - - - - - - - - -
l m n o p q r s t u v w	ĸ⊣≥≥0₽₿₡%+⊃>≷	62 23 64 25 67 68 29 52 13 54 54 51 68	I Mnopqrstuv	К L Z Z O P O E O F D >	62 23 64 25 26 67 68 29 52 13 54 15	ĒF	FF	- 0D 60 
n o p q r s t u v W	<b>ろくに」のよのでのZ</b> る	23 64 25 67 68 9 52 13 54 51 54 51 6	m n o p q r s t u v	LMZOPØ£%FU>	23 64 25 67 68 29 52 13 52 13	FF CR - - - - - - - - - - - - - - - - - -	FF CR - - - - - - - - - -	60 
n o p q r s t u v W	<b>としてしのよのとつと</b>	25 26 67 68 29 52 13 54 15 16	n o p q r s t u v	NOPORSTUV	68 29 52 13 54 15		- - - - - - - -	
p q r s t u v W	v w	67 68 29 52 13 54 15 16	p q r s t u v	P Q R S T U V	68 29 52 13 54 15			
q r s t u v W	v w	68 29 52 13 54 15 16	q r s t u v	V	68 29 52 13 54 15			
s t u v w	v w	52 13 54 15 16	s t u v	V	54 15	- - - - -		
t u v w	v w	13 54 15 16	t U V	V	54 15	-	- - -	- 
V W	v w	15 16	v	V	15	말랐다. 그 아파 :		
	WX	16					a second s	1
~		57	W X	w X	16 57	ETB	ETB	5E
У	Ŷ	58	y i	ŵ X Y	58	1993년 - 1997년 - 1997년 - 1997년 1993년 <del>-</del> 1997년 - 1997년 1997년 - 1997년 - 1		-
Z O	X Y Z )	19 4A	Z O	Z	19 4A	- 	-	-
1 2 3		01 02	\$	10	6B 10		Same	
4	%	43 04 45	@ # \$ 5	! %	6B 10 0B 6B 45 0E		as	
5 6 7 8	\$	46 07	8	:	70		Unshifted	
9	( dāsh	49 20	9 -	<u>(</u>	49 -			
1	=	01	&	<u>+</u>	70			70 70 7F
ì	ì	4C	<u>}</u>	Ī	10	DEL	DEL	7F
Į	!	4C	}	. <b>(</b>	1C 04	11 전쟁 등 등 이 가지. 11 전 전 등 등 이 가지.		_
6		46	#	•	ÖB		-	-
	5B	5B 3B	27		02	en de la Solo - Constante Service - Constante - Const	- 1, <del>- </del> 2, - 1 2, 2, <b>-</b>	=
•	38				그는 것을 하는 것 같아요.			-
	8	8 * 9 ( daīsh daīsh 1 = \ \ [ ] 3 ; 6 ; 5 B	8     *     08       9     (     49       dāsh     dāsh     20       1     =     01       \     3E       [     ]     4C       [     ]     4C       3     ;     43       6     ;     46       ,     5B     5B       ,     3B     3B	$\begin{array}{cccccccccccccccccccccccccccccccccccc$	$\begin{array}{cccccccccccccccccccccccccccccccccccc$	$\begin{array}{cccccccccccccccccccccccccccccccccccc$	$\begin{array}{cccccccccccccccccccccccccccccccccccc$	$\begin{array}{cccccccccccccccccccccccccccccccccccc$

Table D1-6 Keyboard-to-REVERSE EBCD

Untranslatable characters ("-" in the above table) that are entered in transmit strings will be replaced by SPACE (hex 40) during transmission.

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>Enter the hex value for the  $\$  character.

	UN	SHIFTED			S	HIFTED		CON	TROL
KEY	LOWER(\) <sup>1</sup>	UPPER(^) <sup>2</sup>	HEX3	a ta a tito,	LOWER(\) <sup>1</sup>	UPPER(I <sup>^</sup> ) <sup>2</sup>	HEX <sup>3</sup>	CHAR <sup>4</sup>	HEX
ABCDEFGH-JKLMNOPQRSFJ>>XYNo1	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 o 1	<b>⋖₿℃ⅅ</b> ℍ₣ぴ <b>Ⴙ</b> ーյ₭⅃ <b>≌</b> ₽Ѻ₽ぴ <b>虎</b> ᅇ⊢⊃⋗ <b>℁</b> ╳≻⋈→ <u></u> –(	79 76 7A 2A 73 26 19 43 1A 65 25 25 25 25 25 25 25 25 25 25 25 25 25		a b c d e f g h i j k i M n o p q r s t u v W X y z 0 ! c	<b>⋖₿∁</b> ₽⋒⋕ <b>⋳</b> ⋣⋍⋺⋉ <b>⋷</b> ⋈⋗⋳⋩⋳⋩⋈⊢⋺⋋⋚х≻и∽⋳(	79 76 7A 2A 4A 73 23 26 19 43 1A 61 52 45 0B 52 45 0B 52 25 02 31 75 62 67 54 64 01 0	- EOT - BS HT LF - - - - - - - - - - - - - - - - - -	- - - - - - - - - - - - - - - - - - -
2 3 4 5 6 7 8 9 dash	2 3 4 5 6 7 8 9 4 āst	@ # \$ 58 & * ( u <u>n</u> derline	10 70 04 58 68 38 34 37		2 3 4 5 7 8 9 dash	@ # \$ % ~ & * ( underline	10 70 08 1C 68 38 38 34 37		ame as ifted
≤ 5 [ ; ,			13 1F 20 - 6B 49 3B 51			+ Gro + Gro - Stat - St	13 - - 6B 49 -	- DEL - - RS -	- 7F - 2C
/ Space	/ Space	? Space	07 40		/ Space	? Space	07 40	- Space	- 40

Table D1-7 Keyboard-to-SELECTRIC

Untranslatable characters ("-" in the above table) that are entered in 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.

D1-8

Appendix D1 Code Charts: Keyboard-to-Hex-Translation

		NSHIFTED		S	HIFTED		C	ONTROL	
KEY	LOWER(\)	<sup>1</sup> UPPER(^) <sup>2</sup>	HEX3	LOWER(\) <sup>1</sup>	UPPER (^) <sup>2</sup>	HEX3	LOWER(\)1	UPPER(^) <sup>2</sup>	HEX3
ABCDEFGH-JKLMNOPQRSTUVWXYZ0123456789d=\[]	АВСОШҢ ÜI-JKJZZO Р QR %+J>\$XYNPQ\$UR+YJ-04 -/ ->JZZX	dāsh ? : \$ 3 ! & # 8 () 9 0 1 4 B5 7 ; 2 / 6 " 0 1 2 3 4 5 6 7 8 9 dāsh	039E910DA460BF2CC8677A507E3D516776683 F EBCCCD	АВС <b>ОШҢӨТ—</b> ЈКЈХZОРФЯЯ%ТЭ>ЎХУМЈҢ   ТО   ^ Ө   К	dash ?:\$3!()9014B57:2/6=)!-#\$-^&-(:?	039E910DA460BF2CC8677A507E3D5120_49_BA_F01_19	- - - - - - - - - - - - - - - - - - -	- - - - - - - - - - - - - - - - - - -	- - - 05 - 02 - - - - - - - - - - - - - - - - -

Table D1-8 Keyboard-to-BAUDOT

Untranslatable characters ("-" in the above table) that are entered in transmit strings will be replaced by NULL (hex 00) during transmission.

<sup>1</sup>CHAR displayed in Run mode if latest case-control character was letter

<sup>2</sup>CHAR displayed in Run mode if latest case-control character was figure

<sup>3</sup>HEX byte searched for/transmitted

.

<sup>4</sup>Enter the hex value for the  $\setminus$  character.

	UN	SHIFTED		S	HIFTED		CO	NTROL	
KEY	LOWER( <sup>S</sup> I) <sup>1</sup>	UPPER (ዄ )²	HEX3	LOWER( <sup>S</sup> I) <sup>1</sup>	UPPER(ዄ) <sup>2</sup>	HEX <sup>3</sup>	LOWER( <sup>S</sup> I) <sup>1</sup>	UPPER(₺)²	HEX
A	a	61	61	A	f	41	SOH	SOH	01
в	Ь	62	62	В	ッ	42	STX	STX	02
С	C	63	63	С	テ	43	ETX	ETX	03
D	d	64	64	D	k	44	EOT	EOT	04
E	e	65	65	Е	7	45	ENQ	ENQ	05
F	f	66	66	F		46	ACK	ACK	06
G	g	67	67	G	<b>X</b>	47	BEL	BEL	07
н	h	68	68	н	ネ	48	BS	BS	08
I .	i	69	69	I	J	49	HT	НТ	09
J	j	6A	6A	J	11	4A	LF	LF	0A
к	k	6B	6B	к	F	4B	VT	VT	0B
L	I	6C	6C	L	フ	4C	FF	FF	0C
м	m	6D	6D	м	~	4D	CR	CR	0D
N	n	6E	6E	N	Т	4E	SO	so	0E
0	o	6F	6F	0	੨	4F	SI	SI	0F
Р	р	70	70	- P	Ĩ	50	DLE	DLE	10
Q	q	71	71	Q	6	51	DC1	DC1	11
R	r	72	72	R	×	52	DC2	DC2	12
s	S	73	73	S	Ŧ	53	DC3	DC3	13
т	t	74	74	т	ተ	54	DC4	DC4	14
U	u	75	75	U	2	55	NAK	NAK	15
v	v	76	76	v	Э	56	SYN	SYN	16
w	w	77	77	w	ラ	57	ETB	ETB	17
x	×	78	78	x	IJ	58	CAN	CAN	18
Y	У	79	79	Y	JU	59	EM	EM	19

Table D1-9 Keyboard-to-JIS7

<sup>1</sup>CHAR displayed in Run mode if latest case-control character was Shift In (<sup>5</sup><sub>1</sub>).

<sup>2</sup>CHAR displayed in Run mode if latest case-control character was Shift Out (5).

<sup>3</sup>HEX byte trapped/transmitted (space parity)

	UNS	HIFTED				SHIFTED		C	ONTROL	
(EY	LOWER( <sup>S</sup> I) <sup>1</sup> L	JPPER(ᢒ)²	HEX3	L	OWER ( <sup>S</sup> I	)¹ UPPER(ᢒ)²	HEX3	LOWER( <sup>S</sup> I)	<sup>1</sup> UPPER(ጜ) <sup>2</sup>	; HEX3
z	z	7A	7A		z	ŀ	5A	SUB	SUB	1A
0	0	-	30		).		29			
1	1	ア	31		i i	•	21			
2	2	1	32	•	@	5	40	Sa	me	
3	3	ゥ	33		#	L	23			
4	4	I	34		\$	•	24			
5	5	オ	35		%	•	25	8	as	
6	6	カ	36		•	n	5E			
7	7	+	37		&	E	26			
8	8	ጋ	38		•	I	2A	Uns	hifted	
9	9	ታ	39		(	4	28			
dash	dāsh	L	2D		underli	ne *	5F			
=	=	ス	3D		+	オ	2B	-	7E	7E
∖4	¥	ס	5C		-	7C	7C	1 1	60	60
I	I		5B		{	7B	7B	DEL	DEL	7F
]	]	J	5D		}	7D	7D	ESC	ESC	1B
;		<del>'</del>	3B		:	D	3A	NUL	NUL	00
,	,	т	27			Г	22	GS	GS	1D
,	,	Ŧ	2C		<	Ð	<b>3</b> C	RS	RS	1E
		3	2E		>	t	3E	US	US	1F
1	1	۳	2F		?	У	3F	FS	FS	1C
Space	Space	Space	20		Space	Space	20	Space	Space	20

Table D1-9 (continued)

<sup>1</sup>CHAR displayed in Run mode if latest case-control character was Shift In (<sup>5</sup>r). <sup>2</sup>CHAR displayed in Run mode if latest case-control character was Shift Out (<sup>5</sup>b). <sup>3</sup>HEX byte trapped/transmitted (space parity)

<sup>4</sup>Enter the hex value for the ¥ and  $\square$  characters.

KEY	UNSHIF CHAR <sup>2</sup>	TED HEX <sup>3</sup>	(	SHIFTI CHAR <sup>2</sup>	ED HEX <sup>3</sup>	CONT CHAR <sup>2</sup>	'ROL HEX <sup>3</sup>
A B C D H F G H – J K L M N O P Q R S F U > 🛛 X Y N O 1 2 3 4 5 6 7	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 0 1 2 3 4	61 62 63 64 56 66 66 66 66 66 66 77 77 77 77 77 77 77		ABCDWFGH−JK∟MZOPQRSTU>¥XYN)!@#\$	41 42 44 44 44 44 44 45 55 55 55 55 55 55 55	SOH STX EONCK BBST LF FFR SSI DCC2 34 KN ECAN BSUB Sar Sar	
8 9	5 6 7 8 9	35 36 37 38 39		% ^ & * ( underline	25 5E 26 2A 28 5F	as Unsh	
dāsh = \4 [ ] ; , , , , , , , , , , , , , , , , , ,	dāsh = ¥ [ ] ; , , , , , , , , , , , , , , , , , ,	2D 3D 5C 5B 5D 3B 27 2C 2E 2F 20		+ { } : ? Space	5F 2B 7C 7D 3A 22 3C 3E 3F 20	- DEL ESC NUL GS RS US FS Space	7E 60 7F 1B 00 1D 1E 1F 1C 20

Table D1-10 Keyboard-to-JIS8 (space parity)<sup>1</sup>

'Hex data-entry will override parity

<sup>2</sup>CHAR displayed in Run mode

<sup>3</sup>HEX byte trapped/transmitted <sup>4</sup>Enter the hex value for the ¥ character.

Appendix D1 Code Charts: Keyboard-to-Hex-Translation

, G. 1997 - 1997 - 1997 - 1997 - 1997 - 1997 - 1997 - 1997 - 1997 - 1997 - 1997 - 1997 - 1997 - 1997 - 1997 - 1 1997 - 1997 - 1997 - 1997 - 1997 - 1997 - 1997 - 1997 - 1997 - 1997 - 1997 - 1997 - 1997 - 1997 - 1997 - 1997 -	UNSHI	FTED	SHIF		CONT	
KEY	CHAR <sup>2</sup>	HEX3	CHAR <sup>2</sup>	HEX <sup>3</sup>	CHAR <sup>2</sup>	HEX3
2			f	<b>.</b>		0.4
A	E1	E1	1	C1	81	81
В	E2	E2		C2	82	82
C	E3	E3	テ	C3	83	83
D	E4	E4	_ <u>↓</u>	C4	84	84
E	E5	E5	ナニヌネ	C5	85	85
F	E6	<b>E6</b>	Ξ	C6	86	86
G	E7	E7	×	C7	87	87
Ĥ	E8	E8	7	C8	88	88
l d	E9	E9	J	C9	89	89
J -	EA	EA	1	CA	8A	8A
K	EB	EB	en e	СВ	8B	8B
L	EC	EC	フ	CC	8C	8C
М	ED	ED	$\sim$	CD	<b>8</b> D	8D
N	EE	EE	東	CE	8E	8E
0	EF	EF	2	CF	8F	8F
Р	F0	F0	n n	D0	90	90
Q	F1	F1	マミムメモヤュヨラリ	D1	91	91
R	F2	F2	×	D2	92	92
S	F3	F3	Ð	D3	93	93
Т	F4	F4	<b>Þ</b>	D4	94	94
U	F5	F5	2	D5	95	95
V	F6	F6	Э	D6	96	96
w	F7	F7	ラー	D7	97	97
X	F8	F8	j j	D8	98	98
Y	F9	F9	JU	D9	99	99
Z	FA	FA	L L	DA	9A	9A
0		BO	~	A9		
1	ア	B1		A1		
2		B2	5	C0	Sai	ne
3	÷	B3	۔ د	A3		
4	Ť	B4		A4		
4 5	イウエオカキりケ	B5		A5	a	\$
6	5	B6	n	DE	<u>u</u>	-
7	÷	B7	7	A6		
8	л	B8	ヨエ	AA	linet	hifted
9	, ж	B9		A8	0/13/	
	- Э	AD	1	DF		
dash	-					

Table D1-11Keyboard-to-JIS8 (mark parity)1

<sup>1</sup>Hex data-entry will override parity <sup>2</sup>CHAR displayed in Run mode <sup>3</sup>HEX byte trapped/transmitted

	UNSHI	FTED	SHIF	TED		CON	TROL
ΕY	CHAR	HEX	CHAR	HEX		CHAR	HEX
= .	z	BD	*	AB	a f	FE	FE
14	0	DC	FC	FC		EO	E0
[		DB	FB	FB		FF	FF
j .	J	DD	FD	FD		9B	9B
;	サ	BB	<u>ت</u>	BA		80	80
•	T	A7	Г	A2		9D	9D
,	Ŧ	AC	シ	BC		9E	9E
•	Э	AE	った	BE		9F	9F
1	ש	AF		BF		9C	9C
Space	A0	A0	A0	A0		A0	A0

Table D1-11 (continued)

<sup>4</sup>Enter the hex value for the  $\mathcal{I}$  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:

#### INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

1

Table D2-1 Hex-to-Display Translation

INPUT HEX	ASCII¹ (₽)\$1≵184211	EBCDIC <sup>1</sup> ୳୳ଽୄୄୄୄୄଽୄୄୄ ୳ୠୄୄୄଽୄୄୄୄୄ	EBC (P)124 LOWER	8 631	XS-3¹ (₱)રું₀8421।	IPARS <sup>2</sup> (≩⊌8421	REV E (1248 LOWER	1e32(P)	BAUI %84 LETTERS	1211	(P)12	CTRIC <sup>1</sup> 48 \{ \style 1 UPPER	JIS (୩୨୫୬୫ LOWER		JIS8 <sup>1</sup> ୳ୢୣୢୢୢୢୢୢୢୢୢୢୢୢୄ୳ <sub>8</sub> 8421
						hav			hav	hav	00	SP	NU	NU 1	NU
00	NU	NU	space	space	space	hex	space	space	hex E	hex 3	SP		SH	NU SH	
01	SH	SH	dāsh	dash		1	2	=	LF		•	hex			SH
02	SX	SX	@	hex	daish	2 3	2	<	A	LF	t	Т	SX EX	SX EX	SX EX
03	EX	EX	&	+	0	3	4			-		J			
04	ET	hex	8		1	4	-	:	space	hex	4	\$	ET	ET	ET
05	EQ	HT	q	Q	2	5	5	%	S	hex	0	0	EQ	EQ	EQ
06	AK	hex	У	Y	3	6	6			8	ļ	L	AK	AK	AK
07	BL	pad	h	н	4	7	7	>	U	7	/	?	BL	BL	BL
08	BS	hex	4	:	5	8	8	*	CR	CR	5	%	BS	BS	BS
09	нт	hex	m	M	6	9	9	(	D	\$	•		HT	HT	HT
0A	LF	hex	u	U	7	0	0	)	R	4	е	E	LF	LF	LF
0B	VT	VT	d	D	8	-	# .		J	1,000	р	Р	VT	VT	VT
0C	FF	FF	D2	D2	9	CR	1	<b>]</b>	N	:	hex	hex	FF	FF	FF
0D	CR	CR	hex	hex	N	hex	FF	FF	F	. I.	hex	hex	CR	CR	CR
0E	SO	SO	hex	hex		=	^		С	:	hex	hex	SO	SO	SO
0F	SI	SI	D4	D4	E E	hex	ET	ET	к	1	hex	hex	SI	SI	SI
10	DL	DL	2,	<	+	hex	@	hex	Т	5	2	@	DL	DL	DL
11	D1	D1	k	ĸ	:	hex	1	?	Z	hex	· .		D1	D1	D1
12	D2	D2	S	S		S	S	S	L	, <b>, , , )</b> ,	n	Ν	D2	D2	D2
13	D3	D3	b	В	?	т	t	Т	W	2	=	+	D3	D3	D3
14	D4	hex	0	)	A	U	u	U	н	#	z	Z	D4	D4	D4
15	NK	hex	hex	hex	В	V	v	V	Y	6	hex	hex	NK	NK	NK
16	SY	BS	•	hex	С	w	w	w	P	0	hex	hex	SY	SY	SY
17	EB	hex	hex	hex	D	x	X	х	Q	1	hex	hex	EB	EB	EB
18	CN	CN	6	•	E	Y	У	Y	0	9	6	hex	CN	CN	CN
19	EM	EM	o	0	F	z	Z	Z	В	?	1	1	EM	EM	EM
1A	SB	hex	Ŵ	Ŵ	G	dash	hex	hex	G	&	k	к	SB	SB	SB
1B	EC	hex	f	F	н	#	•	hex	•	^	q	Q	EC	EC	EC
10	FS	hex		2	. 1	space	}	{	Μ	•	2	-	FS	FS	FS
1D	GS	GS	BS	BS	=	hex	ĹF	LÈ	X		BS	BS	GS	GS	GS
1E	RS	hex	EB	EB	<	1	EB	EB	V		EB	EB	RS	RS	RS
1E 1F	US	US	1	1	#		hex	hex	No. 1	Ń	1	1	US	US	US

<sup>1</sup>Select Bit Order/Polarity: NORMAL

<sup>2</sup>Select Bit Order/Polarity: REV-INVERT

<sup>3</sup>Select Bit Order/Polarity: REVERSE-NORM

D2-3

INPUT HEX	ASCII	EBCDIC	EB0 LOWER		XS-3	IPARS	REV I			IDOT FIGURES		CTRIC RUPPER	LOWER		JIS8
20	space	hex	1	=	@	@	dash	dash	hex	hex	1	[	space	space	space
21	l.	hex	J	J	*	J	j	J	hex	hex	m	М	1		1.
22		FS	1	?	\$	к	k	к	hex	hex	x	х		Г	1 <b>P</b>
23	#:	hex	а	Α	1	L	1	L	hex	hex	g	G	#	1	#
24	\$	hex	9	G	J	м	m	м	hex	hex	0	• )	\$	,	\$
25	%	LF	r	R	к	N	n	Ν	hex	hex	S	S	%	•	%
26	&	EB	z	Z	L	0	ο	0	hex	hex	h	Н	&	E	&
27	•	EC	I	L	М	Р	р	Р	hex	hex	У	Y	•	- ד	•
28	(	hex	5	%	N	Q	q	Q	hex	hex	7	&	(	4	(
29	)	hex	n	N	0	R	r	R	hex	hex	r	R	)	•	)
2A	*	hex	V <sup>1</sup>	V	Р	:	hex	hex	hex	hex	d	D	*	I	*
2B	+	hex	е	Е	Q	<	\$	!	hex	hex	;	:	+	オ	+
2C	•	hex	RS	RS	R	+	hex	hex	hex	hex	RS	RS	,	Ť	,
2D	_	EQ	CR	CR	%	hex	CR	CR	hex	hex	CR	CR	dash	L	dash
2E	-	AK	LF	LF	,	)	BS	BS	hex	hex	LF	LF	•	Э	•
2F	1	BL	HT	HT		(	hex	hex	hex	hex	HT	ΗT	1	ש	1
30	0	hex	3	•	underline	\$	&	+	hex	hex	3	#	0	-	0
31	1	hex	1	L	(	Α	a	Α	hex	hex	v	V	1	ア	1
32	2	SY	t	Т	•	В	b	в	hex	hex	u	U	2	1	2
33	3	hex	С	C	&	С	С	С	hex	hex	f	F	3	ゥ	3
34	4	hex	#	-	1	D	d	D	hex	hex	9	(	4	I	4
35	5	RS	\$	!	S*	Е	е	Е	hex	hex	w	w	5	オ	5
36	6	hex	•	hex	Т	F	f	F	hex	hex	b	в	6	カ	6
37	7	ET	•	hex	U	G	g	G	hex	hex	-	_	7	+	7
38	8	hex	7	>	V	н	h	н	hex	hex	8	*	8	ጋ	8
39	9	hex	p	Ρ	W	1	ing in the second	1	hex	hex	а	Α	9	ካ	9
3A	:	hex	X	X	×	?	hex	hex	hex	hex	С	C	:	D	:
3B	1. <b>1</b>	hex	g	G	Y	•	•	hex	hex	hex	•		;	サ	;
3C	<	D4	ET	ET	Z	%	~	•	hex	hex	ET	ET	<	シ	<
3D		NK	SY	SY	. )	hex	HT	HT	hex	hex	hex	hex		2	= -
3E	>	hex	SH	SH	>	•	Ν.	Ν.	hex	hex	hex	hex	>	セ	>
3F	?	SB	pad	pad	*	hex	pad	pad	hex	hex	pad	pad	?	У	?

Table D2-1 (continued)

\*SYNC = even parity S  $(35_{16})$ .

.

INPUT HEX	ASCII	EBCDIC	EB0 LOWER		XS-3	IPARS	REV I LOWER			IDOT FIGURES		CTRIC R UPPER	JIS LOWER		JIS8
40	@	space	space	space	space	hex	space	space	hex	hex	SP	SP	@	5	@
41	A	hex		l. s.	1	hex	1	=	hex	hex	1	hex	Ă	÷	A
42	В	hex	ē	hex	dash	hex	2	<	hex	hex	t	Т	В	ý	В
43	С	hex	&	+	0	hex	3	;	hex	hex	1	J	С	Ŧ	С
44	D	hex	8	+	1	hex	4	:	hex	hex	4	\$	D	- K	D
45	E	hex	q	Q	2	hex	5	%	hex	hex	0	0	E	+	E
46	F	hex	ý	Y	3	hex	6	•	hex	hex	1	L	F	<u> </u>	Ŕ
47	G	hex	h	н	4	hex	7	>	hex	hex	1	?	G	7	G
48	н	hex	4	:	5	hex	8	*	hex	hex	5	%	н	*	H
49	1	hex	m	Μ	6	hex	9	(	hex	hex	•	•	1	)	1
4A	J	hex	u	U	7	hex	0	)	hex	hex	е	Е	J	Л	J
4B	к		d	D	8	hex	#	"	hex	hex	р	Р	к	E	к
4C	L	<	D4	D2	9	hex	· [	1	hex	hex	hex	hex	L	7	L
4D	М	(	hex	hex	N	hex	FF	FF	hex	hex	hex	hex	M	$\sim$	М
4E	N	+	hex	hex	;	hex	•		hex	hex	hex	hex	N	<b>太</b>	N
4F	ο	hex	D4	D4	Ţ	hex	ET	ET	hex	hex	hex	hex	ο	<b>२</b>	0
50	Р	&	2	<	+	hex	@	hex	hex	hex	2	@	Р	R	Р
51	Q	hex	k	ĸ	•	hex	1	?	hex	hex	•	124 1.1	Q	6	Q
52	R	hex	S	S		hex	s	S	hex	hex	n	N	R	×	R
53	S	hex	b	В	?	hex	t	T	hex	hex	=	+	S	Ŧ	S
54	Т	hex	0	)	Α	hex	u	U	hex	hex	Z	Z	T	Þ	Т
55	U	hex	hex	hex	B*	hex	V	V	hex	hex	hex	hex	U	2	U
56	V	hex	hex	hex	С	hex	w	W	hex	hex	hex	hex	V	Э	V
57	W	hex	hex	hex	D	hex	×	x	hex	hex	hex	hex	W	ラ	W
58	X	hex	6	•	E	hex	У	Y	hex	hex	6	hex	х	IJ	x
59	Ŷ	hex	0	0	F	hex	Z	Z	hex	hex	T	1	Y	ΙL	Y
5A	Z	1	w	W	G	hex	hex	hex	hex	hex	k	к	Z	L	Z
5B	a sen 🛔 sur ante ar ar		<b>f</b>	F		hex		hex	hex	hex	P	Q	ſ		ſ
5C	Λ	*	6.1 <b>.3</b> 65.	<b>^</b>	1	hex			hex	hex	<b>^</b>	<b>^</b>	¥	יי	¥
5D	]	e de <b>j</b> ever	BS	BS	= 1	hex	LF	LF	hex	hex	BS	BS	1	C	] 1.24
5E	ana 🖍 na sa atao	t on a state to a state of	EB	EB	<	hex	EB	EB	hex	hex	EB	EB	<ul> <li>An entering</li> </ul>	n	en en en en <b>a</b> rrenek en en en
5F	_	hex	N	Ν.	#	hex	hex	hex	hex	hex	N	Ν.	underline	•	underline

\*EOM = even parity B  $(55_{16})$ .

Table D2-1 (continued)

HEX	ASCII	EBCDIC EBCD XS-3 IPARS REV EBCD BAUDOT LOWER UPPER LETTERS FIGURES		SELECTRIC LOWER UPPER		JIS7 LOWER UPPER		JIS8							
60	<b>i</b> ,	dash	1	=	@	hex	daish	dash	hex	hex	1	1		hex	•
61	a	1	j	J	*	hex	J	J	hex	hex	m	Ň	а	hex	а
62	b	hex	1	?	\$	hex	k	ĸ	hex	hex	х	Х	b	hex	b
63	C	hex	a	Α	1	hex	1	Ľ	hex	hex	g	G	С	hex	с
64	d	hex	9	(	J	hex	m	М	hex	hex	0	)	d	hex	d
65	e	hex	. <b>r</b> -	R	к	hex	n	N	hex	hex	S	S	е	hex	e
66	f	hex	z	Z	L	hex	0	ο	hex	hex	h	н	f	hex	f
67	g	hex	I	1	Μ	hex	р	Р	hex	hex	У	Y	g	hex	g
68	ĥ	hex	5	%	Ν	hex	q	Q	hex	hex	7	&	ĥ	hex	ĥ
69	I	hex	n	Ν	0	hex	r	R	hex	hex	r	RI	i.	hex	i
6A	J	:	v	V	Р	hex	hex	hex	hex	hex	d	D	J	hex	J
6B	k.		e	E	Q	hex	\$	1	hex	hex	;	:	k	hex	k
6C	I	%	RS	RS	R	hex	hex	hex	hex	hex	RS	RS	1	hex	1
6D	m	_	CR	CR	%	hex	CR	CR	hex	hex	CR	CR	m	hex	m
6E	n	5	LF	LF	,	hex	BS	BS	hex	hex	LF	LF	n	hex	n
6F	0	?	нт	HT	^	hex	hex	hex	hex	hex	HT	нт	0	hex	0
70	Р	hex	3	;	_	hex	8.	+	hex	hex	3	#	р	hex	р
71	q	hex	I.	L	(	hex	a	Α	hex	hex	V	V	q	hex	q
72	r	hex	t	Т	,	hex	b	в	hex	hex	u	U	r	hex	r
73	S	hex	C	С	&	hex	С	С	hex	hex	f	F	S	hex	S
74	t	hex	#		1	hex	d	D	hex	hex	q	(	t	hex	t
75	u	hex	\$	1	S	hex	е	Е	hex	hex	w	Ŵ	u	hex	u
76	V	hex	,	hex	т	hex	f	F	hex	hex	b	В	V	hex	v
77	w	hex	• 1	hex	U	hex	g	G	hex	hex	daish	underline	w	hex	w
78	x	hex	7	>	V	hex	h	н	hex	hex	8	*	x	hex	x
79	y	•	P	P	W	hex	I	1	hex	hex	а	A	У	hex	У
7A	z	:	×	X	X	hex	hex	hex	hex	hex	C	C	Z	hex	z
7B	{	#	g	G	Y	hex		hex	hex	hex	,	•	{	hex	{
7C	L.	@	ĔŤ	ET	Z	hex	~	•	hex	hex	ET	ET	1	hex	1
7D	}:	7	SY	SY	)	hex	HT	HT	hex	hex	hex	hex	}	hex	}
7E	-	=	SH	SH	>	hex	Ν.	Ν.	hex	hex	hex	hex	-	hex	~
7F	pad	e na se 🗰 🗰 e e e e e e e e e e e e e e e e	pad	pad		hex	pad	pad	hex	hex	pad	pad	pad	hex	pad

D2-6

NPUT HEX	ASCII	EBCDIC	EBCDIC	EBCDIC	EBCDIC	EBCD LOWER UPPER	XS-3	IPARS	REV EBCD LOWER UPPER	BAUDOT LETTERS FIGURES	SELECTRIC	JIS		JIS8
80	NU	hex		hex	hex				NU	NU	hex			
81	SH	а		hex	hex				SH	SH	hex			
82	SX	b		hex	hex				SX	SX	hex			
83	EX	C		hex	hex				EX	EX	hex			
84	ET	d		hex	hex				ET	ET	hex			
85	EQ	е		hex	hex				EQ	EQ	hex			
86	AK	f		hex	hex				AK	AK	hex			
87	BL	g		hex	hex				BL	BL	hex			
88	BS	ĥ		hex	hex				BS	BS	hex			
89	HT	A17		hex	hex				HT	ĤΤ	hex			
8A	LF	hex		hex	hex				LF	LF	hex			
8B	VT	hex		hex	hex				VT	VT	hex			
8C	FF	hex		hex	hex				FF	FF	hex			
8D	CR	hex		hex	hex				CR	CR	hex			
8E	SO	hex		hex	hex				SO	SO	hex			
8F	SI	hex		hex	hex				SI	SI	hex			
90	DL	hex		hex	hex				DL	DL	hex			
91	D1	J		hex	hex				D1	D1	hex			
92	D2	k		hex	hex				D2	D2	hex			
93	D3	1		hex	hex				D3	D3	hex			
94	D4	m		hex	hex				D4	D4	hex			
95	NK	n		hex	hex				NK	NK	hex			
96	SY	0		hex	hex				SY	SY	hex			
97	EB	р		hex	hex				EB	EB	hex			
98	CN	q		hex	hex				CN	CN	hex			
99	EM	r		hex	hex				EM	EM	hex			
9A	SB	hex		hex	hex				SB	SB	hex			
9B	EC	hex		hex	hex				EC	EC	hex			
9C	FS	hex		hex	hex				FS	FS	hex			
9D	GS	hex		hex	hex				GS	GS	hex			
9E	RS	hex		hex	hex				RS	RS	hex			
9F	US	hex		hex	hex				US	US	hex			

HEX	ASCII	EBCDIC	EBCDIC	EBCD LOWER UPPER	XS-3	IPARS	REV EBCD LOWER UPPER	BAUDOT LETTERS FIGURES	SELECTRIC	JIS		JIS8
A0		hex		hex	hex				space	space	hex	
A1	1	-		hex	hex				!			
A2		s		hex	hex					Г	Г	
A3	#	t		hex	hex				#	L.	L	
A4	\$	u		hex	hex				\$			
A5	%	V		hex	hex				%	•	•	
A6	&	w		hex	hex				&	E	Э	
A7	<b>!</b>	x		hex	hex				•	г	יד	
A8	(	У		hex	hex				(	1	ſ	
A9	)	z		hex	hex				)	•	•	
AA	•	hex		hex	hex				*	I	I	
AB	+	hex		hex	hex				+	オ	オ	
AC	,	hex		hex	hex				,	Ť	Ŧ	
AD	dash	hex		hex	hex				dash	ב	L	
AE	•	hex		hex	hex				•	Э	Э	
AF	1	hex		hex	hex				1	ש	ש	
B0	0	hex		hex	hex				0	<u>.</u>	<u> </u>	
B1	1	hex		hex	hex				1	ア	ア	
B2	2	hex		hex	hex				2	1	1	
B3	3	hex		hex	hex				3	ウ	ゥ	
B4	4	hex		hex	hex				4	I	I	
B5	5	hex		hex	hex				5	オ	エオカ	
B6	6	hex		hex	hex				6	カ	カ	
B7	7	hex		hex	hex				7	+	<b>+</b> 1	
B8	8	hex		hex	hex				8	Л		
B9	9	hex		hex	hex				9	ታ	ታ	
BA	5 <b>1</b> 1	hex		hex	hex				:	C	D	
BB		hex		hex	hex				;	サート	サー	
BC	<	hex		hex	hex				<	Ð	Э	
BD	= -	hex		hex	hex				=	Z	Z	
BE	>	hex		hex	hex				>	t	t	
BF	?	hex		hex	hex				?	ÿ	ÿ	

Table D2-1 (continued)

i

INPUT HEX	ASCII	EBCDIC	EBCD LOWER UPPER	XS-3	IPARS	REV EBCD LOWER UPPER	BAUDOT LETTERS FIGURES	SELECTRIC LOWER UPPER	JIS LOWER		JIS8
CO	@	{		hex	hex				@	5	5
C1	Ā	À		hex	hex				Ā	f	÷
C2	В	В		hex	hex				В	9	ÿ
C3	С	С		hex	hex			•	С	Ŧ	テ
C4	D	D		hex	hex				D	k	K
C5	Е	E		hex	hex				Е	ナ	+
C6	F	F		hex	hex				F		<u> </u>
C7	G	G		hex	hex				G	- R	ニヌネ
C8	н	н		hex	hex				н	ネ	ネ
C9	1	1		hex	hex				I	1	)
CA	J	hex		hex	hex				J	1	Л
СВ	К	hex		hex	hex				к	E	ヒフ
CC	L	hex		hex	hex				L	7	フ
CD	М	hex		hex	hex				М	~	$\sim$
CE	N	hex		hex	hex				N	<b>太</b>	東
CF	0	hex		hex	hex				0	マ	7
D0	Р	•		hex	hex				Р	R	R
D1	Q	J		hex	hex				Q	6	6
D2	R	к		hex	hex				R	×	ミムメモヤ
D3	S	L		hex	hex				S	ŧ	Ŧ
D4	Т	М		hex	hex				Т	Þ	
D5	U	N		hex	hex				U	2	2
D6	V	0		hex	hex				v	и Г Г	ユ ヨ ラ
D7	W	Р		hex	hex				W		ラ
D8	x	Q		hex	hex				x	IJ	IJ
D9	Y	R		hex	hex				Y	JU	JU
DA	Z	hex		hex	hex				Z	L	L
DB	and free second as	hex		hex	hex				Ţ		D
DC	Ν.	hex		hex	hex				¥	ינ	כ
DD	]	hex		hex	hex	<ul> <li>A second sec second second sec</li></ul>			]	ĩ	Ĵ
DE	•••••	hex		hex	hex					n •	N •
DF	_	hex		hex	hex				underline	•	•

D2-9

NPUT HEX	ASCII	EBCDIC	EBCD LOWER UPPER	XS-3	IPARS	REV EBCD LOWER UPPER	BAUDOT LETTERS FIGURES	SELECTRIC	JIS LOWER	JIS8	
E0	4			hex	hex				4	hex	hex
E1	а	hex		hex	hex				а	hex	hex
E2	b	S		hex	hex				b	hex	hex
E3	C	т		hex	hex				С	hex	hex
E4	d	U		hex	hex				d	hex	hex
E5	е	V		hex	hex				е	hex	hex
E6	f	W		hex	hex				f	hex	hex
E7	g	х		hex	hex				g	hex	hex
E8	ĥ	Y		hex	hex				ĥ	hex	hex
E9	1	Z		hex	hex				i	hex	hex
EA	J	hex		hex	hex				]	hex	hex
EB	k	hex		hex	hex				k	hex	hex
EC	1	hex		hex	hex				1	hex	hex
ED	m	hex		hex	hex				m	hex	hex
EE	n	hex		hex	hex				n	hex	hex
EF	0	hex		hex	hex				0	hex	hex
F0	Р	0		hex	hex				р	hex	hex
F1	q	1		hex	hex				q	hex	hex
F2	r	2		hex	hex				r	hex	hex
F3	S	3		hex	hex				S	hex	hex
F4	t	4		hex	hex				t	hex	hex
F5	u	5		hex	hex				u	hex	hex
F6	V.	6		hex	hex				V	hex	hex
F7	w	7		hex	hex				W	hex	hex
F8	x	8		hex	hex				x	hex	hex
F9	y	9		hex	hex				У	hex	hex
FA	Z	hex		hex	hex				Z	hex	hex
FB	{	hex		hex	hex				{	hex	hex
FC	l.	hex		hex	hex				1	hex	hex
FD	}	hex		hex	hex				}	hex	hex
FE	-	hex		hex	hex				<b></b>	hex	hex
FF	pad	hex		hex	hex				pad	hex	hex

Table D2-1 (continued)

동생은 영국에서 가지 않는 것을 하는 것이 없다.
## **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.
- 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 EM, BLOCK (F1), IN/OUT (F7), READ/U (F3). Enter the name of the file when prompted. The absolute pathname of the standard ASCII code file is HRD/sys/codes/ASCII. Press EM. 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 [19]. 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 жж Name of file: HRD/sys/codes/ASCII Code for ASCII Version: з; Name: "ASCII"; To Graphic: nu sx bl sh ex et eq ak lf ff bs ht si vt so cr d 1 d 1 d2 dЗ d4 eb nk sy em sb fs <u></u>gs cn ec rs us MOVE CO

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 °<sub>1</sub>. 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 '\' with 080. Figure D3-2 shows the set after the first replacement. Next, locate and edit position DC.

Appendix D3 Code Charts: User-Defined Codes

** Protoco Name of file: HRD/sys/codes/A Name: "ASCII";	ol Spreadsheet ** ASCII
To Graphic:	
bs ht lf vt ff cr so s dl dl d2 d3 d4 nk sy e	blender en son en s s i eb
	us ,,, ,/, ,7,
'8' '9' ':' ';' '<' '=' '>' ' '@' 'A' 'B' 'C' 'D' 'E' 'F' '	, <sup>5</sup> , ,0,
	'Ŵ' '_'
F1 F2 F3 F4 BEGIN END CLEAR DELET	4 F5 F6 F7 F8 TE MOVE COPY IN/OUT

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.

Character	Code-Table Entry	Character	Code-Table Entry
۲J	000	Ēc	01b
ъ. Ъ	001	Fs	01c
\$ <sub>x</sub>	002	G	01d
۶	003	P <sub>S</sub>	01e
Ę,	004	u u	01f
50	005	(space)	020
<b>%</b>	006	1999년 1999년 1997년 19	021
E.	007	n de la construcción de la constru La construcción de la construcción d	022
B5	008	#	023
4	009	\$	024
۶.	00a	* %	025
¥	00b		
F	00c	&	026
Ср.	00d	•	027
5	00e		028
s <sub>I</sub>	00f		029
P	010	*	02a
D <sub>1</sub>	011	+	02b
<b>D</b> 2	012	· · · · · · · · · · · · · · · · · · ·	02c
<b>B</b> 3	013		02d
D <sub>4</sub>	014	n de la tradición de la companya de La companya de la comp	02e
۲k	015	/	02f
Sr .	016	Ø	030
EB C	017 018	1	031
SN E	019	2	032
۶	015	3	033

Table D3-1 Code-Set Characters

Character	Code-Table Entry	Character	Code-Table Entry
4	034	Ρ	050
5	035		051
6	036	R	052
7	037	S S	053
8	038	. Т	054
9	039		055
:	03a		056
;	03b	Ŵ	057
<	03c		
<b>=</b>	03d	<b>X</b> <sub>1,1</sub> <sup>1</sup> <sup>1</sup> <sup>1</sup> <sup>1</sup>	058
>	03e	Y	059
?	03f	Z	05a
@	040		05b
A	041	$\mathbf{N}$ . The second se	05c
В	042		05d
С	043	<b>^</b>	05e
D	044	ана 1997 — Алариана 1997 — <b>—</b>	05f
E	045	- 10 m m - 10 m m ■	060
F	046	a	061
G	047	d	062
Н	048	C	063
I	049	d	064
J	04a		065
к	04b	e	
L state	04c	f <sub>state</sub>	066
M	04d	9	067
<b>N</b> (230)	04e	h aga	068
	04f	<b>i</b>	069

Table D3-1 (continued)

.

D3-5

Character	Code-Table Entry	Character	Code-Table Entry
j	06a	Ð	086
k	06b	ד	087
1	06c	4	088
m	06d	<b>`</b>	089
n	06e	I	08a
D	. 06f	7	08b
þ	070	Ť	08c
q	071		08d
r	072	ے _	08e
S	073	3	
t	074	ש	08f
u	075	-	090
V	076	ア	091
ω	077	1	092
×	078	Ċ	093
У	079	I	094
z	07a	*	095
{	07b	カ	096
:	07c 07d	+	097
) ~	07e	Л	098
~	07f	ካ	099
¥	080	ے ا	09a
	081	<del>''</del>	09b
•	082	Ð	09c
	082	<b>Z</b>	09d
	084	t	09e
•	085	у У	09f

Character	Code-Table Entry	Character	Code-Table Entry
5	0a0		Obb
<b>f</b>	0a1		0bc
<u>پ</u>	0a2	د. الانتقال ال	0bd
<b>7</b>	0a3	n (1997) N	0be
k - Maria	0a4		Obf
7	0a5	Ç	
<b>1</b> tae	0a6		0c0
R	0a7	<b>ü</b> (1987) 1	0c1
*	0a8	é i said	0c2
J Contraction	0a9	â	0c3
Л	Oaa	ä	0c4
E	0ab	à	0c5
כ ייי	Oac	a	0c6
	0ad	ç	0c7
π	Oae	ê	0c8
ू २	Oaf	ë	
R	060		0c9
2	061	è	0ca
		ï	0cb
× ×	0b2	î	0cc
ŧ	0b3	ì	0cd
4	0b4	Ä	0ce
2	0b5	Â	Ocf
Э	0b6	É	0d0
ラ	0Ь7	æ	0d1
IJ	0b8		
JJ.	0b9 •au angestikken de best van de skriver in de skriver en de skriver de skriver de skriver de skriver de skriver	Æ	0d2
L	0ba	Ô	0d3

Table D3-1 (continued)

D3-7

Character	Code-Table Entry	Character	Code-Table Entry
ö	0d4	ú	0e3
ò	0d5	ñ	0e4
û	0d6	ñ	0e5
ù	0d7	<u>a</u>	0e6
ÿ	0d8	Q	0e7
Ö	0d9	ذ	0e8
ü	0da	E and	0e9
¢	0db	<b>-</b>	0ea
£	0dc	1/2	0eb
ß	0dd	1/4	0ec
<b>R</b>	0de	i	0ed
£	0df	••	0ee
á	0e0	5	0ef
í	0e1	•	OfO †
Ó	0e2		

Table D3-1 (continued)

+ Values 0f1-0ff are undefined.

Appendix E Communications with the AR Division Factory

# 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 <u>must</u> post this RA number on the outside of the package in order for AR Division to honor the terms of the Maintenance Agreement.

MAY 1990 PAGE E-1

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.

MAY 1990 PAGE E-2

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.

#### ADDENDUM

#### INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

MAY 1990 PAGE E-4

.

# 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.

F-1



Figure F-1 Soft Pack Travel Bag, Option 99.



Figure F-2 Hard Shell Travel Case, Option 95.

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).

and the state of the

. Količej – Merije u roku postala postu – Dijek eko kolje Elektrone jednom te kolje mostale konjem postatkoj u tele

# 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 *lsys/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 *lsys/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.

	** File Maintenance **				
Include DIR 32 08/30/88 16:09	Command: CHANG Drive: FD1 Name:	DIR <u>/sys/codes</u>	Push XEQ T	o Perform	Command
	Current Director	y: HRD/usr		`	11 11 12 12 12 12 12 12 12 12 12 12 12 1
					16:09 09:21

Press x to access the directory containing the JIS7 and JIS8 files.

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 J/S7 and J/S8 Files into /sys/codes Directory

Press COPY. Leave the source pathname on the From line blank: we will make the From selections via the we key in the body of the current directory listings themselves. Press the 4 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 *lsys/codes*. Be sure to type a slash (*l*) 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  $\blacksquare$ . Move the cursor down over JIS8 and mark this file as well.

Your screen should resemble the screen drawn in Figure H-2. Press  $\overline{}$  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 M	laintenance **		
To: H	D1 Name: _	sys/codes FD1/sys/coo		To Perform	Command
JIS7 JIS8		ASCII ASCII	2179 1113	10/01/88 10/01/88	14:44 14:46
F 1	F 2	F3 F4	F 5 F	6 F 7	F 8

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.

	** Line
LINE SETUP	
Mode: MONITOR	
Source: LINE	
Code: JIS7	
Bits: 8 Parity: Format: BOP	NONE
Format: BOP	

Figure H-3 Files loaded into the /sys/codes directory are selectable in the Code field on the Line Setup menu.

After bootup, press man, SETUP, LINE, to access the Line Setup menu. Move the cursor down to the **Code** field. Press is or the to rotate the selections in this field until you have verified that 357 and 358 are available as new code selections. Figure H-3 shows a line setup with 357 selected in the **Code** field.

### H.4 Testing with JIS7/JIS8

In your line setup, be sure to change Mode: <u>AUTOMON</u> to <u>MONITOR</u> 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 <u>ASCH</u>.

Figure H-4 shows a screen display for JIS7, a shifted code. Note that the messages with Katakana text begin with Shift Out  $(\mathfrak{F}, hex \mathfrak{o}_{\mathbf{E}})$ .

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.

<u>JIS7/8/NONE/BOP</u> え、二回点点 私二 2 5 ツ 5 バロー・ つレバ、ワラント れれれれれれれれれれれれれれれれれれれれれれ れれれれれれれれれれれれ	ስለስለስለስስስስስስስስስስስስስ ስለስስስስስስስስስስስስስስስ ኦህኒ፲ጋ-፡ታፖኦታ" 2.ፍኣ፴ልስ rom Washington DC t ስለስስስስስስስስስስስስስስስ echo test from Was
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA	nnnnnnnnnnnnh ハノ、IDー・テストデ ス。ない回れれ rom Washington DC t nnnnnnnnnnnnnnn echo test from Was
<u>ለለመለከለ በ በ በ በ በ በ በ በ በ በ በ በ በ በ በ በ በ በ </u>	AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
<u></u>	<u>echo test from Was</u>
	コレハ、ワシントンカラトウキョウヘノ、
44(回をもしちきもハロー。 コレハ、ワシント	NANANANANANANANANANANANANANANANANANANA
	<u>・ファラトウキョウヘノ、エコー・テストテ</u>
алаалалалалалалалалалалал CENTER¥%	лалалалалалалалал <u>** ST</u>

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** 

	13 00000 25 (DB-25, female)	
Pin No.	Pin Name	Signal Description
1 2 3 4 5 6 7 8 9 10 11 12 13 14	Frame Ground TD RD RTS CTS Signal Ground CD	Ground RS-232/V.24 Output RS-232/V.24 Input RS-232/V.24 Output RS-232/V.24 Input Ground RS-232/V.24 Input
15 16 17 18	SCT SCR	RS-232/V.24 Input RS-232/V.24 Input
19 20 21 22 23 24 25	DTR	RS-232/V.24 Output

Table I-1 Remote Connector

	13	1
	1000	
	1100	000/
	25	14
	(DB-25, r	nale) and a second state of the
Pin No.	Pin Name	Signal Description
	an (an an ann) 1994 ann an an Anna an A	
1	Frame Ground	Ground
2	TD RD	RS-232/V.24 Input RS-232/V.24 Output
4	RTS	RS-232/V.24 Unput
5	CTS	RS-232/V.24 Mput
6	DSR	RS-232/V.24 Output
7	Signal Ground	Ground
8	CD	RS-232/V.24 Output
9		
10		
11		
12		
13		
14		
15	SCT	RS-232/V.24 Input
16		
17	SCR	RS-232/V.24 Input
18		
19	rente presi	
20	DTR	RS-232/V.24 Input
21		
22 23		
23		
24		
20		

Table I-2 Printer Connector

Tab	le I-3
Auxiliary	Connector



(16-pin Bi-directional	TTL	Input/Output.	DB-25.	female)

Pin No.	Pin Name		
1 . <b>1</b>	PA0		
2	PB0		
3	PA1		
	PB1		
- <b>5</b>	PA2		
6	PB2		
7	PA3		
8	PB3		
9	PA4		
10	PB4		
11	PA5		
12	PB5		
13	PA6		
14	PB6		
15	PA7		
16	PB7		
17 · · · ·	Signal Ground		
18	Reserved		
19	Signal Ground		
20	Reserved		
21	Signal Ground		
22	Reserved		
23	Signal Ground		
24	Reserved		
25	Signal Ground		

RGB N	e I-4 Monitor	
	female)	
Pin No.	Pin Name	
1 2 3 4 5 6 7 8 9	Signal Ground Signal Ground Red Green Blue Brightness Reserved Horizontal Sync Vertical Sync	

I-5



Figure I-1 RS-232/V.24 Interface Module.

Appendix I Interface Specifications

			Signal Description		
Pin No.	Pin Name	Monitor Mode	To DCE (Em DTE) (DB-25, male)	To DTE (Em DCE) (DB-25, female)	
1	Frame Ground	Frame Ground	Frame Ground	Frame Ground	
2	TD	High Impedance Input	RS-232/V.24 Output	RS-232/V.24 Input	
3	RD	High Impedance Input	RS-232/V.24 Input	RS-232/V.24 Output	
4	RTS	High Impedance Input	RS-232/V.24 Output	RS-232/V.24 Input	
5	CTS	High Impedance Input	RS-232/V.24 Input	RS-232/V.24 Outpu	
6	DSR	High Impedance Input	RS-232/V.24 Input	RS-232/V.24 Outpu	
7	Signal Ground	Signal Ground	Signal Ground	Signal Ground	
8	CD	High Impedance Input	RS-232/V.24 Input	RS-232/V.24 Outpu	
9		Test Point	Test Point	Test Point	
10		Test Point	Test Point	Test Point	
11		Test Point	Test Point	Test Point	
12	SCD	Test Point	Test Point	Test Point	
13	SCTS	Test Point	Test Point	Test Point	
14	STD	Test Point	Test Point	Test Point	
15	SCT	High Impedance Input	RS-232/V.24 Input	RS-232/V.24 Outpu	
16	SRD	Test Point	Test Point	Test Point	
17	SCR	High Impedance Input	RS-232/V.24 Input	RS-232/V.24 Outpu	
18	LL	Test Point	Test Point	Test Point	
19	SRTS	Test Point	Test Point	Test Point	
20	DTR	High Impedance Input	RS-232/V.24 Output	RS-232/V.24 Input	
21	SQ	Test Point	Test Point	Test Point	
22	RI	Test Point	Test Point	Test Point	
23	DSRS	Test Point	Test Point	Test Point	
24	SCTE	High IMpedance Input	RS-232/V.24 Output	RS-232/V.24 Input	
25	ТІ	Test Point	Test Point	Test Point	

Table I-5 RS-232 Test Interface Module

.



Figure I-2 V.35 Interface Module.

Pin No.	Circuit	Signal	Monitor Mode	To DTE (Em DCE) (34-pin, female)	To DCE (Em DTE) (34-pin, female)
Α	101	Frame Ground	Frame Ground	Frame Ground	Frame Ground
В	102	Signal Ground	Signal Ground	Signal Ground	Signal Ground
С	105	RTS	High Impedance Input	V.35 Input	V.35 Output
D	106	CTS	High Impedance Input	V.35 Output	V.35 Input
Е	107	DSR	High Impedance Input	V.35 Output	V.35 Input
F	109	CD	High Impedance Input	V.35 Output	V.35 Input
н	108	DTR	High Impedance Input	V.35 Input	V.35 Output
J	125	RI	Test Point	Test Point	Test Point
R	104A	RD	High Impedance Input	V.35 Output	V.35 Input
Т	104B				
v	115A	SCR	High Impedance Input	V.35 Output	V.35 Input
х	115B				
Y	114A	SCT	High Impedance Input	V.35 Output	V.35 Input
AA	114B			v.oo Output	v.55 input
Р	103A	TD	High Impedance Input	V.35 Input	V.35 Output
S	103B			v.oo mput	
U W	113A 113B	SCTE	High Impedance Input	V.35 Input	V.35 Output
к	F1		_	_	_
М	F1			<b>—</b>	_
L	F2	Test Point	Test Point	Test Point	- Test Point
N	F2			_	
z	F3	i An ann an thairtean	en e	: 	
BB	F3		n 📥 oo soo tahu dhada		
cc	F4		_	_	_
EE	F4	<u> </u>	_	_	
DD	F5	·	<b></b>	_	
FF	F5	en la constante de la constante Esta de la constante de la const	y energia y anticipation de la competencia de la competencia. Energia	n general and a strengt a program and a strengt and a s The strengt and a strengt an	na ana ao amin'ny tanàna mandritry dia mampika dia mandritry dia mandritry dia mandritry dia mandritry dia mand Ny kaodim-paositra dia mandritry dia mandritry dia mandritry dia mandritry dia mandritry dia mandritry dia mandr
нн КК	N1 N1	Test Point Test Point	Test Point Test Point	Test Point Test Point	Test Point Test Point
JJ	N2	∫ Test Point	Test Point	Test Point	Test Point
LL	N2	<b>1</b> Test Point	Test Point	Test Point	Test Point
NANA.	F				
MM NN	F			_	-

Table I-6 V.35 Test Interface Module

·

1-9



Figure I-3 X.21 Interface Module.



					Signal Description	
Pin No. Circu	Circuit	it Pin ID	Pin Name		To DCE (Em DTE) (15 Pin, male)	To DTE (Em DCE) (15 Pin, female)
1		Shield	Shield	Frame Ground	Frame Ground	Frame Ground
2,9	103	Т	Transmit Data	High Impedance Input	X.21 Output	X.21 Input
3,10	105	С	Control	High Impedance Input	X.21 Output	X.21 Input
4,11	104	R	<b>Receive</b> Data	High Impedance Input	X.21 Input	X.21 Output
5,12	109	1	Indicator	High Impedance Input	X.21 Input	X.21 Output
6,13	114	S	Signal Timing	High Impedance Input	X.21 Input	X.21 Output
7,14		в	Byte Strobe	High Impedance Input	X.21 Input	X.21 Output
15		-		Test Point	Test Point	Test Point
Patch F	anel:	U/A A,B*		High Impedance Differential Input	High Impedance Differential Input	High Impedance Differential Input
		+5V		Output	Output	Output
		-5V		Output	Output	Output
		GND	Ground	Ground	Ground	Ground

Table I-7 X.21 Test Interface Module

\* UA A and B can be used for balanced or unbalanced signals. (Do not connect B when you are looking at unbalanced signals.)

.



Figure I-5 RS-485 Interface Module.
Pin No.				Signal Description	
	Circuit	Pin ID	Monitor Mode	15 Pin, Male	15 Pin, Female
1	_	Shield	Frame Ground	Frame Ground	Frame Ground
2 9	103 103	A Bus +	High Impedance Input	RS-485 Output RS-485 Output	RS-485 Input RS-485 Input
3,10	103	A Bus -	High Impedance Input	H3-465 Output	no-405 input
4	104	B Bus +	High Impedance Input	RS-485 Input	RS-485 Output
11	104	B Bus -	High Impedance Input	RS-485 Input	RS-485 Output
5,12				_	
6,13					
7,14				- 199 <u>-</u> - 1	
15					_

Table I-8 RS-485 Test Interface Module



Figure I-6 RS-449/V.36/V.37 Interface Module.



Figure I-7 RS-449/V.36/V.37 LED Overlay.

			Signal Description		
Pin No.	Pin ID	Pin Name	– Monitor Mode	To DCE (Em DTE) (36 Pin, male)	To DTE (Em DCE) (36 Pin, female)
2	SI	Signalling Rate Indicator	Test Point	Test Point	Test Point
4.22	SD	Send Data	High Impedance Input	RS-449 Output	RS-449 Input
5,23	ST	Send Timing	High Impedance Input	RS-449 Input	RS-449 Output
6,24	RD	Receive Data	High Impedance Input	RS-449 Input	RS-449 Output
7.25	RS	Request to Send	High Impedance Input	RS-449 Output	RS-449 Input
8,26	RT	Receive Timing	High Impedance Input	RS-449 Input	RS-449 Output
9.27	CS	Clear to Send	High Impedance Input	RS-449 Input	RS-449 Output
10	LL	Local Loopback	High Impedance Input	Test Point	Test Point
11.29	DM	Data Mode	High Impedance Input	RS-449 Input	RS-449 Output
12,30	TR	Terminal Ready	High Impedance Input	RS-449 Output	RS-449 Input
13,31	RR	Receiver Ready	High Impedance Input	RS-449 Input	RS-449 Output
13,31	RL	Remote Loopback	High Impedance Input	High Impedance Input	High Impedance Input
15		· ·	High Impedance Input	High Impedance Input	High Impedance Input
15	SF/SR	Incoming Call	Test Point	Test Point	Test Point
10	SF/SH	Select Frequency/	lest Point	Test Point	lest Point
47.05	<b>*</b> *	Signaling Rate Selector	I Bala Jana adama a Jamint		DC 440 Immut
17,35	TT	Terminal Timing	High Impedance Input	RS-449 Output	RS-449 Input
18	TM	Test Mode	High Impedance Input	High Impedance Input	High Impedance Input
19	SG	Signal Ground	Signal Ground	Signal Ground	Signal Ground
28	IS	In Service	Test Point	Test Point	Test Point
32	SS	Select Standby	High Impedance Input	High Impedance Input	High Impedance Input
33	SQ	Signal Quality	High Impedance Input	High Impedance Input	High Impedance Input
36	SB	Standby Indicator	High Impedance Input	High Impedance Input	High Impedance Inpu
	y Patch Pa			· · · · · · · · · · · · · · · · · · ·	
	UA A,B	Unassigned Input	High Impedance Input	High Impedance Input	High Impedance Inpu
	+5	+5 Volts	Output	Output	Output
	-5	-5 Volts	Output	Output	Output
19	GND	Ground	Signal Ground	Signal Ground	Signal Ground
	AUX0 A,B	Auxiliary	Output	Output	Output
	AUX1	Auxiliary	Output	Output	Output
	AUX2	Auxiliary	Output	Output	Output
37	SC	Send Common	Send Common	Send Common	Send Common
20	RC	Receive Common	Receive Common	Receive Common	Receive Common
34	NS	New Signal	Test Point	Test Point	Test Point
• ·	N A.B	National A, B	Reserved	Reserved	Reserved
-	SHIELD	Shield	Frame Ground	Frame Ground	Frame Ground

Table I-9 RS-449/V.36/V.37 Test Interface Module



Figure I-8 T1 Interface Module.

			Signal Description	
Pin No.	Pin Name	Monitor	To DCE (Em DTE)	To DTE (Em DCE)
1	Send Data Tip	High Impedance Input	Output	Input
2	Frame Ground	Ground	Ground	Ground
3	Receive Data Tip	High Impedance Input	Input	Output
5	Remote Test Make	High Impedance Input	Test Point	Test Point
6	Remote Test Break	High Impedance Input	Test Point	Test Point
9	Send Data Ring	High Impedance Input	Output	Input
11	Receive Data Ring	High Impedance Input	Input	Output
13	Remote Test Common	High Impedance Input	Test Point	Test Point

Table I-10 T-1 Test Interface Specifications 1

(1) Unlisted connectors are wired 1-for-1 through the two connectors. Test points are connected to switches and test points only.

I-17



Figure I-9 G.703 Interface Module.

		Signal Description		
Pin No.	Pin Name	Monitor	To DCE (Em DTE)	To DTE (Em DCE)
Pin No.	Pin Name	Monitor	To DCE (Em DTE)	To DTE (Em DCE)
1	Receive Data Tip	High Impedence Input	Input	Output
2	Frame Ground	Ground	Ground	Ground
5	Send Data Tip	High Impedence Input	Output	Input
6	Receive Data Ring	High Impedence Input	Input	Output
9	Send Data Ring	High Impedence Input	Output	Input

Table I-11 G.703 Test Interface Specifications 1

(1) Unlisted connectors are wired 1-for-1 through the two connectors.

INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108





ISDN BASIC RATE S/T INTERFACE

INTERVIEW REMOTE FREEZE

Figure I-11 ISDN LED overlay.

			Signal Description	
Pin No.	Pin Name	Monitor	To DCE (Em DTE)	To DTE (Em DCE)
3	Send Data Tip	High Impedence Input	Output	Input
4	Receive Data Tip	High Impedence Input	Input	Output
5	<b>Receive Data Ring</b>	High Impedence Input	Input	Output
6	Send Data Ring	High Impedence Input	Output	Input
7 <sup>2</sup>	- voltage	Output	Output	Output
8 <sup>2</sup>	+ voltage	Output	Output	Output

Table I-12 ISDN Test Interface Specifications 1

(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 1.430.

INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

.

# 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.

J-1

수영화관 영상도 사고 전 가장 있는 것이다. 우리 확장 관계 영감 강감 강리는 것은 것은 데너지

> tin de la constante de la const La constante de la constante de

> > .

J-2

Appendix J1 Field Service: Eliminating Static Electricity

## Appendix J1: Eliminating Static Electricity



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.

#### INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

(2) The Letter of the second state of the s

#### 이 제품 가격 문화를 가지 않는 것을 하는 것

(a) A set is a set of the set

,

Appendix J2 Field Service: Removing Logic Boards

### 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 <u>must</u> 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.

J2-6

Appendix J3 Field Service: Installing Logic Boards

### Appendix J3: Installing Logic Boards

INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108



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  $\mathbf{\overline{M}}$  in the diagram represents a bi-directional buffer.



)

#### (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.

(a) All all contracts of the theory of content seconds on the All Marson end of the SUGDer data of the context is pullicly for a transmission of 22 meaningly on the SUGDer reference. Such the contract contracts are the All the simple and of the incorrects once detected to only be given in the Chieve disk intermedie was been as in a sector that are the given into the content of the intermedie and an intervention in a sector to the given into the Chieve disk intermedie and intervention in the association of the sector of the methods intermedie and direct of the sector that can be reacted by reaction promoted with the operation of the content and the sector of the sector of the sector of the content of the sector methods in the sector of the sector
Appendix J3 Field Service: Installing Logic Boards

S1-Switch Settings for MPM Boards in the INTERVIEW 7000 Series										
MPM	Slot 7			Slot 8				Slot 9		
Pin		2	<b>3</b> 4		2	3	1	2	3	
Units with 3 MPMs	81						an a			
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	2:									
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	

Table J3-1

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\_rcrd 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 :CPMO[MOM1M2]" 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.

Figure (1999) Figure (2007) 20 - 000 Control (2007) 20 - 10000 Esta (2007) 20 - 20000 Control (2007) 20 - 200 Control (2007) 20 - 200 Control (2007) 20 - 2000 Control (2007) 2000 Control (2007) 2000 El (2007) 2000 Control (2007) 2000 Control (2007) 2000000 Esta (2007) 2000 Control (2007) 2000 Control (2007) 2000 Control (2007) 2000000 Esta (2007) Control (2007) 2000 Control (2007) 2000 Control (2007) 2000000 Esta (2007) Control (2007) 2000 Control (2007) 2000 Control (2007) 20000 Esta (2000) Control (2007) 2000 Control (2007) 2000 Control (2007) 20000 Control (2000) Control (2007) 2000 Control (2007) 2000 Control (2007) 2000 Control (2007) 20000 Control (2007) 2000 Control (2007) 2007) 2000 Control (2007) 2007) 2000 Control (2007) 20

Persit that even at of the submarners are "cound" to stay connect algorithms. Those that are not "keyed" aded to reversive cours course areas ad fine quilles stres with one product populating holds. See Neuro 19-17

the set issi a CL

, Betono trati overplanely apoural or inaray, it is deginered that your 1999, pour 1999 . Index cartala drat in functions when Cale Dravel recomment. Press, analare iss course. Appendix J3 Field Service: Installing Logic Boards

** 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.



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.

INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

Ale (1497年1月) Start Fail Ale (1497年1日) Ale (1997年1日) Ale (1997年1日) Ale (1997年1日) Ale (1997年1日) Ale (1997年1日) Ale (1997年1日) Ale (1997年1月) Ale (199

金融资源 计成本分词 计信息分词存在目标 化模糊

navoD král shupači 18.00

Place the white the brack came and controls the His horeway to that the cover showed

# **Appendix J4: Installing Multiplexer Board**

CAUTION: STATIC ELECTRICITY CAN DAMAGE THE UNIT WHEN THE COVER IS REMOVED. Before you begin this section, you <u>must</u> 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.





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)

one parte di Lebre a reactadi di barcati con stil

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.

INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108



Figure J4-2 All PROMs are factory installed on the T1 mux board, represented above by the most prominent components.

# 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.

## INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

## 为"就会做我了你帮助"。 化基料

en forme encountry the summary one of the one of the sectors well from the full encode all the manuations contractly. All board procedures to the other 18 to employed the context (relation scilledy meaning balloes procedures on the or will be an onso the more demonstration of yer according beliefs was sectore when one of the context of the contex

The must beauti opporte special to deplete the masses of matters in the CRA C18: Bald. If there are a type theories of the TLE THAT I ANTHON OPEN mystak the spaceStress from the metrology of our sequence and the execution the fram hourd. Replace the enversion of one-on opport and the secondary the set of our cases there. If welf and other points contains to the second of second

## ent onuced - Ale

h han daa siijan 1937 Beheresta () - Caraa Arkiis, pirree sine ware na (e insele and repisee timpets eerimaa shirekii wa arkiin a arkii

# 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 <u>must</u> 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.



#### INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

# 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.

nat na skulter and character.

#### (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.





Figure J5-2 PROM GA and PROM GB are located on the upper left side of the CPM beaution of the CPM beaution of the CPM beaution of the compared bea

# J5.3 Exchange the PROMs on the CPM Board

## (A) Locate the PROMs

and the capitosaness and result

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.

• edge of the PROME and requirement of an one observer in a receiver, water-about motion to access to be Conflict the externality as required. Place the old PROME in the rectionance of the contract flue reprincement. For our sense for the contract of the contract of the explorable for Packing and Shipping instruction.

- 22069 and add inten (G)

(a) Havali accheracia econtro Place Control Place Control Place Active respectivity, the PROMS to achie superificative reacted of ROS is practice previous that active that and and of anthe ROOA: The tractice for the control that a control place data and these PROMS spatial contributions. State The Structure as well as the piny and their tracteres active to be a state of the State Control and active State and their tracteres and active to be a state of the State Control and the Manther prestate theory pages actors and active to the tracteres with GN in the manther prestate into the formation.

## based entropalqpA A.S.

#### The state of the second of the second second second (A).

Replace the board in the Line the Line of the second by which you construct it. Take our to held it this is the edited. To do sittle it into its constant.

estériC entribisariaceA (B)

The exhibit spread is the concerned in the riverse order in which you distommental theory. Next, repay a be excillent torrider carder while from the PCM being on the contract same

based to the COMPERCESS (Sector Sector Sector) and according to the test and the firm of the firm the firm the based of the device of the SCI Proved in the Science of the

## State Test the Unit

Sefore replacing the cover, review processes to be concern you have followed all the instantions correctly. Follow procedures is Section 15.4 in replace the caver (as a rately measure before you tota the unit) as well as to easy the new dimensive an all process up before you secure the cover.

If the SELF TEST STRORS: has shows anything other their NORE and you are insisting the CRM board, my restabiling the PRONE. Go to the heginning of this section (Appendix 35) and batefully tallow the same protocolous and matrixations given.) Replace the oover and priver up the unit upon such that and matrixations once more.) If the self test will gives protos, tail Unitation fervice. Appendix J6 Field Service: Installing Hard Disk Drive

gir hinister seiter ein Fillenden. Saar hie eine Swo 12410 der Hellenden als Statistiche Swo

# 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 <u>must</u> 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.

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.

kuere is an unacaened power carle win-a tote pin remain connects cerning on bracket. This cable will give dower to rise hard drive once it is installed. For now, move it towards the teach and outside of rom mit, clear of the bracket.

There are five strike holding this brocket in the unit-two small Phillips screws toward the front outside of the brocket and three standard-head captive screws holding the bracket to the bottom of the unit, as shown in Figure 16-1. Remove the



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.

문

# 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.

4

Phillips-head

(top and bottom)

# J6.4 Connect the Cables

Captive Screws

(back)

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.

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 :CPMO<u>MOM1M2</u>" 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: Ø1-1					
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.

INTERVIEW 7000 Series Advanced Programming: ATLC-107-951-108

Appendix J7 Field Service: Servicing Other Components

# 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 slots. 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 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 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. 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

#### operator punctuator

#### K.1.1.2 Keywords

[These words (entirely in lower-case) are reserved due to their special meanings.]

keyword: one of

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float *	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

[\* The reserved word float is not used in the INTERVIEW implementation of C.]

[The following two words are reserved in the INTERVIEW implementation.]

task 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.]

identifier:

nondigit identifier nondigit identifier digit

nondigit: one of

\_ 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

digit: one of

0 1 2 3 4 5 6 7 8 9

## 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

escape-sequence: one of

\' \" \? \\ \o \oo \ooo 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]) 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 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

declarator [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

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; break; 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

Clock

signal, 1-13

# 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

# С

Capture Memory, field on Record Setup menu, 12-9

Character buffer, storage capacity, 1-12

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 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** 

# Ε

EIA, trigger conditions, fails to come true, 2-19

EIA leads, storage of, 1-12

Emulation, connectors used, 1-10

Errors, recoverable, 2-16

External monitors RGB color video connector, 1–9 RS-170 video connector, 1–10

### F

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 - 8From 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

# G

General operation, boot-up program, creating a user interface, 2 - 4changing 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

Disk Maintenance, menu selection, 12–10 installing new system software via the DUPDISK command, 2–7

#### ADDENDUM

### Η

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

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

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, 2-17 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

### Μ

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, 1-5Miscellaneous Utilities, overview, 2-14 Mode, test mode field on Line Setup menu, 1 - 11Modem connector external. 1-9 internal, 1-9 MPM errors, 2-16

### 0

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

### Ρ

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 Power connector, 1–7

Power switch, back panel, 1-8

Power up, 1-14, 2-1self 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

### R

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

### S

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

## T

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 - 20Trigger conditions, EIA, fails to come true, 2-19 Trigger Setup menus overview, 2-11 transmit string, does not appear on screen, 2 - 20Trouble-shooting data plus leads, failure of leads to transition, 2 - 20data-plus-leads display, failure of leads to transition, 2-20

layers, passing data between, 2-20

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 /usr/user\_intrf affect on Start Up screen, 2-2 creating a user interface, 2-4

Utilities menus, overview, 2-14 See also Separate entry under name of each menu

### 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, microfloppies, 1-5

#### ADDENDUM

# Index A Technical Manual Part I and Part II

### 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
- M bit mask symbol, in Receive string condition, 21-7
- » close double parens symbol, 3-5, 22-4, 25-5
- \*/ closing delimiter for comment, 24-13
- I don't care symbol in Receive string condition, 21-7 in spreadsheet string search, 33-35
- E flag, 7E flag symbol, 5−13 C coding, 59-6
- G goodbcc, overlay for good BCC or FCS, 5-13 C coding, 59-6
- highlighted plus symbol, indicates wrap in logical line, 26-3
- Inot equal flag symbol in Receive string condition, 21-7 in spreadsheet string search, for beginning of frame, 33-35
- « open double parens symbol, 3-5, 22-4, 25-5
- /\* opening delimiter for comment, 24-13
- # pad, in Outsync Char field, 4-8
- S 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

### Α

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 - 15Accumulate timer, layer-independent action, 27 - 15Accumulator 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 - 8comments in, 24-13 record on/off, 11-10 ADDR, 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 ASCII 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 - 17AUX 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, 4-3stage in autoconfiguration displayed in Status field, 5-3

updates Line Setup screen, 5-4with no clock, speed defaults to 168 kbps, 5-4

Autosync, subfield on Line Setup menu, 4-7, 4-8

Auxiliary connector, I-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

# В

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 S\$#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 SDLC, 35-27 X.25 Layer 2, 33-28 adjunct to send-string action, Layer 1, 28-9 appended to transmit string, 8-5 as condition, 8-5fevar\_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, 28-4overlay, 8-3 Basic Rate ISDN, 48-3 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

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, 9-7initiating 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 Bisvnc 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, 5-14in 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, 4-6per 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 Blnk, 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

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 - 4running 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, 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 %m, 61-12, 61-27, 61-30, 61-32 %0, 61-11 octal, 57-8 %p, 61-11 %s, 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

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 - 16locating 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 true 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, 57-16th\_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 example, 67-15 #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, 53-14test, 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, 58-4-58-6display\_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 - 15syntax 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, 34-6CALLING, 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 multiframes, G.703 frame structures, 50 - 31Cable length, T1 Interface Control screen, 49-21 Cable type, T1 Interface Control screen, 49-20 Cables 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 - 6as 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 - 12Capture 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

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, 20-5-20-8See 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 - 10Channel mode G.703, field on Interface Control menu, 50 - 22G.703 BERT, 9-40 T1 BERT, 9-27 Channel number G.703, field on Interface Control menu, 50 - 23T1 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, 5-26display 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

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 character, 16-5 trigger 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 - 24Comment, 24-13 debugging tool, 24-14 delimiters, 24-13 in C region, 53-10 length of, 24-13location 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** naming 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, 25-6Protocol 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, 5-12Control 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 laver-independent action, 27-9 maximum value vs. maximum stat display, 27 - 6may 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, 27-9transmitted, 29-4 Cover removal. See Field Service replacement. See Field Service CPM board block diagram, 2 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 - 10CTS-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 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, 59-15 Data event-word, data display, 59-15 Data Path G.703, field on Interface Control menu, 50 - 22T1, field on Interface Control menu, 49-24 T1 Interface Control selection, 49-23 Data packet monitor/receive condition, X.25 Layer 3, 34 - 15translates 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, 12-7Data to Record, field on Record Setup menu, 11 - 6Data 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, 15-3set 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 /svs, 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, 5-2traces, 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, 30-9DL\_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, 55-4sent up automatically by Give Data action at Layer 2, 33-28, 33-34, 35-27, 35-32, 39-26, 39-31 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 - 10DTE, 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 - 10Dual 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

### Ε

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 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 trigger 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, 36-3field 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, 10-4Emulate 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

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 BCC Setup menu, 8–12 Enhance field on Display Setup menu, 5-12, 11-5 field on Miscellaneous Utilities menu, 16-4, 16 - 5field on Trigger Setup menu, 22-7 Enhance character data as Laver 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 12 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 13 enhance, C variable, 78-4 SDLC action, 35-30 l2\_enhance, C variable, 73-8 SNA action, 36-3 12\_enhance, C variable, 74-4 SS#7 Layer 2 action, 42-10 l2\_enhance, C variable, 79-4 SS#7 Layer 3, 13\_enhance, C variable, 80-7 SS#7 Layer 3 action, 43-10 X.25 Layer 2 action, 33-32 12 enhance, C variable, 71-8 X.25 Layer 3 action, 34-42 13\_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 - 13used 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 interactive messages, A1-1-A1-15 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

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 - 12setup 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, 59-2monitor/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 - 17Front 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

Index to Part I and Part II

#### 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 - 4changing default menus, 2-6 common problems, 2-20-2-22data 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, 12 give data, C routine, 77-9 SDLC action, 35-27 12\_give\_data, C routine, 73-9 X.25 Layer 2 action, 30-10, 33-28 12\_give\_data, C routine, 71-9 X.25 Layer 3 action, 34-39 13\_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, 8-5default BCC for frames, 33-28, 35-26, 39-25 monitor/receive condition, Layer 1, 28-4

operational only when Rcv Blk Chk

parameters defined on BCC Setup menu, 8-6

enabled, 28-4

overlay, 8-3

Goto, C statement, 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

Graphical Statistics menu, color graphics, 16-6

### Η

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 - 8Hex 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 - 10displaying synchronous idle, 4-8 idle\_action, C routine, 59-13 retained/discarded on FEB Setup menu, 7-3, 7 - 4selecting 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 - 3field 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 - 1string 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

**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 - 32In/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 - 3Include channel 16, G.703, field on Interface Control menu, 50-26 Increment counter, layer-independent action, 27 - 9Increment 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 - 9Initial 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 - 7Instrument, 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 - 10D 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, 48-8NT, 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 - 10statistics 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

# Κ

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, 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

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 - 14Load 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, 49-9Loop 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

M, trace column, X.25 Layer 3, 34-14 M bit adjunct to monitor/receive-packet condition, 34 - 18adjunct 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 - 18X.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, 47-6Message 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, 16-4-16-5background color, 16-5 blink, 16-5 character, 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 - 6setting 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

### Ν

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 Laver 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 - 12Nonliteral 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

Index to Part I and Part II

SDLC, 35-24

X.25 Layer 2, 33-26

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

# 0

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 - 8placement 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 - 22not as versatile as source-code files, 13-15 use disk-space intensively, 13-15

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, 59-141 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 interactive messages, A1-1-A1-15 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 - 5Order 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

LAPD, 39-11

SDLC, 35-12

Other frame

OSI primitives. See Primitives

monitor/receive condition

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 - 17send 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 Laver 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

### Ρ

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
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 adjunct to send-packet action, X.25 Layer 3, 34-32, 34-33, 34-34 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, J2-2PDU. See Primitive data unit; Primitives

Perc(entage), 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, 27-10transfer 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

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 - 41trace 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

56 - 12in linkable-object files, 56-11 system-generated during Compile spreadsheet, 56-11 using multiple hooks, 56-12 #pragma layer, C directive, used to declare a laver, 53-1 #pragma nowarn, C directive, used to suppress compiler warnings, A3-1 #pragma object, C preprocessor directive, 56-8 format of, 56-8 placement of, 56-8 #pragma tracebuf, C directive, used to configure size of trace-buffer arrays, 5-18, 5-24, 61-26 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, 63-2being 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, 30-3automatic, 31-1 at Layer 1, 30-3 below the top layer, 31-1monitor primitives, 31-1 varies with protocol package, 31-1 currently not accessible at Layer 1, 30-3 indications versus requests, 30-10

hook text added to top-level main function,

Index to Part I and Part II

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-independent action, 27-16 used to check status of print buffer, 64-1 Print accumulator, layer-independent action, 27 - 16Print buffer overrun, 64–1 minimize by suspending playback, 64-2 queues unprinted text from print actions, 64-1 Print counter, layer-independent 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-independent action, 27-18 Print server, transfers output from print buffer to printer port, 64-1 Print timer, layer-independent 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, 5-20selecting state names from via Display Setup, 5-19, 27-19 specific to Layer/Test selected on Display Setup, 5-19, 27-19

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, 61-3printing, 27-18 prompt line never accessed by trace routines, 61-4 using backslash and double-quote characters inside of, 27-12Protocol 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, 6-3Protocol Spreadsheet comments, 24–13 constants, 20-8, 25-3-25-7 creating and editing, 19-4-19-5 See also Protocol Spreadsheet editor 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, 6-4how to select and load, 6-3 Ps calculated automatically, adjunct to send-data-packet action, X.25 Layer 3, 34-35 reset, emulate-mode action, X.25 Layer 3, 34 - 41same 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 - 35trace column 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 character 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 - 24via REMOTE RS-232 port, 67-1 Receive path, upward path of IL buffer in emulate mode, 55-3 Receiver, Conditions, Trigger Setup menus, 21 - 5Receiver gain, G.703, Interface Control selection, 50-19Record layer-independent action, 27-20 start\_rcrd\_play, C routine, 14-6, 65-3, 69-16 suspend\_rcrd\_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

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, I-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 - 27in 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 reset\_ns, C routine, 77-11 SDLC, 35-30 reset\_ns, C routine, 73-11 X.25 Layer 2, 33-31 reset\_ns, C routine, 71-11 Reset Pr and Ps, emulate-mode action, X.25 Layer 3, 34-41 reset\_pr\_ps, 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

Index to Part I and Part II

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 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

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

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 11\_il\_tansmit, C routine, 59-12

11 tansmit, C routine, 59-11

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 SIO monitor/receive condition, LAPD, 39-10 send action, LAPD, 39-21 SI1 monitor/receive condition, LAPD, 39-10 send action, LAPD, 39-21 Sig Channel Polarity, T1 Interface Control selection, 49-6Sign 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, 49-6Signal channel idle char, T1 Interface Control
- screen, 49–27 Sizzal abarral sumbar T1 Istarford Control
- 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-recieve 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, 41-3Layer 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 - 8comments in, 24-13 traced along with layers and tests on Program Trace, 5–19 Static electricity anit-static packing, J1-3 elimination, J1-3 Static Leads, field on Interface Control menu, 10 - 17Statistics 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 12\_suppress, C variable, 77-8 Q.931 action, 40-12, 40-13

13\_suppress, C variable, 78-4

SDLC action, 35–30, 35–32 12 suppress, C variable, 73-8 SNA action, 36-3 12\_suppress, C variable, 74-4 SS#7 Layer 2 action, 42-10, 42-11 12\_suppress, C variable, 79-4 SS#7 Layer 3 action, 43-10, 43-11 13\_suppress, C variable, 80-7 X.25 Layer 2 action, 33-32, 33-33 12\_suppress, C variable, 71-8 X.25 Layer 3 action, 34-42, 34-43 13\_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, 4-9Synchronization 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

## Τ

T, type, field on Graphical Statistics menu,

18 - 5T1, 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 emulation modes, 49-5, 49-7 FAS, 49-35 FEB Setup screen, 9-25 FPS (framing pattern sequence), in ESF transmissions, 49-35 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, 49-30sync 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, I-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

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 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 11\_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 condition, 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 DTE, test connector, 1-11 To Disk Number, subfield on Disk Maintenance menu, 12-9 Trace as component of custom protocol analysis, 5 - 20as 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 LAPD, 39-5 Q.931, 40-3 SDLC, 35-5 SNA-SDLC, 36-4 X.25 Layer 2, 33-5, 42-3 X.25 Layer 3, 34-8, 43-4 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 sample 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 G.703, Interface Control selection, 50-20 T1, Interface Control selection, 49-21 Transmit string complete 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 control of color display, 16-5-16-6 in C. 53-8 Trouble-shooting data plus leads, failure of leads to transition, 2 - 21data-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, 2-20Protocol 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 - 21trigger conditions, EIA, fails to come true, 2 - 20Twisted 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, 61-23buffer 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, I-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

Voltages RS-232 detected by receivers, 10-9 generated by drivers, 10-9 generated by special output jacks, 10-8 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, 44-6 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 Waitfor, C statement, 53-2, 53-4, 53-5, 53-6, 53-7, 53-9, 53-13, 54-1, 56-3 placed inside of state loop, 52-16 Wall clock accurate to one millisecond, 7-6 controls timers when time ticks are disabled, 27 - 10drives 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 defined, 33-29, 34-39, 35-27, 39-26

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 - 3field on SDLC Frame Level Setup screen, 35 - 3field 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 - 6monitoring T and R for valid status, 32-5 sending from Layer 2, 32-5, 32-11 Test Interface Module, 46-2, I-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

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

## Υ

Yellow alarm, T1 transmissions, 49-24

## Ζ

Zero, transmitting steady zero, set\_tcr\_b, C routine, 59-14

# Index B C Structures, Variables, and Routines

### Α

- 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\_il\_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

## В

bcc\_error LAPD event, defined, 77-3 SDLC event, defined, 73-3 X.25 Layer 2 event, defined, 71-3

# С

Counter events, counter\_name\_change, 62-3 structures, counter\_struct, 62-2

clearerr, disk I/O routine, 65-2, 65-4 defined, 65-10

convert\_tick\_count, timer routine, 62-11, 62-12, 62-18 defined, 62-18 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 ctl\_enhance\_rd, data-display routine, defined, 59-8 ctl 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, 70-2defined, 70-3 current\_line, Display Window variable, 61-4 defined, 61-5

# D

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, 78-2SS#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

display window index buffer, Display Window structure, 61-7, 61-35 defined, 61-8 displayc, Display Window routine, 61-1 defined, 61-9 displayf, Display Window routine, 56-14, 56-19, 61-1, 61-3, 61-8, 61-13, 61-32, 61-33, 65-7, 65-19, 67-6, 67-11 defined, 61-9 displays, Display Window routine, 56-19, 56-22, 57-10, 58-1, 58-3, 61-1, 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, 79-3 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, 78-2SS#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

#### Ε

EIA events, fevar\_eia\_changed, 60-2 routines, ctl\_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 enter\_data\_phase, X.21 routine, 70-10 defined, 70-9

## F

Flag events, flag\_name\_change, 69-4 structures, flag struct, 69-2 fclose, disk I/O routine, 65-1, 65-3, 65-6, 65-7, 65-10, 65-11, 65-13, 65-14, 67-16 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\_rd DDCMP event, defined, 75-2 line event, 59-2, 59-6 defined, 59-3 fevar\_bd\_bcc\_td DDCMP event, defined, 75-2 line event, 59-2 defined, 59-3 fevar\_bd\_bcc2\_rd, DDCMP event, 59-6 defined, 75-2fevar\_bd\_bcc2\_td, DDCMP event, defined, 75-2 fevar eia changed EIA event, 53-15, 54-2, 60-1, 60-3 defined, 60-2X.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 rd DDCMP event, defined, 75-2 line event, 54-2, 59-6 defined, 59-3

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 fgets, 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 fopen, disk I/O routine, 65-2, 65-3, 65-5, 65-13, 67-15 defined, 65-4 fprintf, disk I/O routine, 65-2, 65-7, 65-31 defined, 65-28 fputc, disk I/O routine, 65-2, 65-11, 65-27 defined, 65-26 fputs, 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 fwrite, disk I/O routine, 65-1, 65-2, 65-15, 65-24, 65-36 defined, 65-23

### G

defined, 63–37

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

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

get\_wall\_time\_ticks, timer routine, 62-17 defined, 62-16

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.

invalid\_packet, X.25 Layer 3 event, defined, 72-3 OSI routine, 63-12

# Κ

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

#### 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

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 rcvd\_frame, 77-3 routines 12\_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 12\_enhance, 77-6 12 lower window edge, 77-5 l2\_resend\_edge, 77-5 12\_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\_cntrl\_byte\_1, 77-4 m\_frame\_nr, 77-4 m\_frame\_ns, 77-4 m\_frame\_pf, 77-4 m\_frame\_type, 77-4 rcvd\_frame\_addr\_cr, 77-4 rcvd\_frame\_addr\_sapi, 77-4 rcvd\_frame\_addr\_tei, 77-4 rcvd\_frame\_bcc\_type, 77-5 rcvd\_frame\_buff\_seg, 77-5 rcvd\_frame\_cntrl\_byte\_1, 77-5 rcvd\_frame\_nr, 77-5 rcvd\_frame\_ns, 77-5 rcvd\_frame\_pf, 77-5 rcvd\_frame\_sdu\_offset, 77-5 rcvd\_frame\_sdu\_size, 77-5 rcvd\_frame\_type, 77-4

#### Line

events 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\_buffer\_full, 59-3

fevar rcvd char rd, 59-3 fevar rcvd char td, 59-3 routines, outsync\_action, 59-10 variables rcv\_buffer\_full, 59-4 rcvd\_char\_rd, 59-4 rcvd\_char\_td, 59-4 Lock, routines lock, 69-13 unlock, 69-14 l1\_il\_transmit, transmit routine, 55-1, 55-4, 55-8, 63-39 defined, 59-12 11 tick count, timer variable, 62-8, 62-9, 62-11, 62-18, 63-10, 63-16, 63-19, 63-22, 63-25, 63-28, 63-30 defined, 62-1011\_transmit, transmit routine, 59-1 defined, 59-11 11\_trbuf, trace buffer structure, defined, 61-25 12\_current\_window\_edge LAPD variable, defined, 77-5 SDLC variable, 73-8 defined, 73-5 X.25 Layer 2 variable, 71-8, 77-8 defined, 71-5 l2 enhance LAPD variable, 77-8 defined, 77-6 SDLC variable, 73-8 defined, 73-6 SNA variable, 74-4 SS#7 Layer 2 variable, 79-4 defined, 79-3 X.25 Layer 2 variable, 71-8 defined, 71-5 12 give data LAPD routine, defined, 77-9 SDLC routine, defined, 73–9 X.25 Layer 2 routine, defined, 71-9 12\_lower\_window\_edge LAPD variable, 77-8 defined, 77-5 SDLC variable, 73-8 defined, 73-5 X.25 Layer 2 variable, 71-8 defined, 71-5 12\_resend\_edge LAPD variable, 77-8 defined, 77-5

SDLC variable, 73-8 defined, 73-5 X.25 Layer 2 variable, 71-8 defined, 71-512\_suppress LAPD variable, 77-8 defined, 77-6 SDLC variable, 73-8 defined, 73-6 SNA variable, 74-4 SS#7 Layer 2 variable, 79-4 defined, 79-3 X.25 Layer 2 variable, 71-8 defined, 71-5 l2 T1 LAPD event, defined, 77-3 SDLC event, defined, 73-3 X.25 Layer 2 event, defined, 71-3 l2\_tick\_count, OSI Layer 2 variable, defined, 63-16 12\_trbuf, trace buffer structure, defined, 61-25 12\_upper\_window\_edge LAPD variable, 77-8 defined, 77-5 SDLC variable, 73-8 defined, 73-5 X.25 Layer 2 variable, 71-8 defined, 71-5 l2pp trbuff, Protocol Trace variable, defined, 61-39 l2pp\_trbuff\_ctl, Protocol Trace structure, defined, 61-4112pp trbuff end, Protocol Trace variable, defined, 61-39 13\_clear\_path, X.25 Layer 3 routine, defined, 72-11 13 enhance Q.931 variable, 78-4 defined, 78-3 SS#7 Layer 3 variable, 80-7 defined, 80-6 X.25 Layer 3 variable, 72-9 defined, 72-7 13\_give\_data, X.25 Layer 3 routine, defined, 72-10 13\_more\_to\_resend, X.25 Layer 3 routine, defined, 72-1113\_suppress Q.931 variable, 78-4 defined, 78-3

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

- 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

- lo\_t\_pdu\_seg, OSI Layer 5 variable, defined, 63-23
- lo\_t\_prmtv, OSI Layer 5 event, defined, 63-23
- lo\_t\_prmtv\_code, OSI Layer 5 variable, 63-55, 63-59 defined, 63-23
- lo\_t\_prmtv\_path, OSI Layer 5 variable, 63-59 defined, 63-24
- lo\_t\_sdu, OSI Layer 5 variable, 63-59 defined, 63-24
- load\_program, program-chaining routine, 69-13 defined, 69-12
- lock

disk I/O routine, 65-4, 65-15 defined, 65-14 lock routine, 69-14 defined, 69-13

#### Μ

- m\_bib, SS#7 Layer 2 variable, defined, 79-2
- m\_call\_ref\_flag, Q.931 variable, defined, 78-2
- m\_call\_ref\_len, Q.931 variable, defined, 78-3
- m cic, SS#7 Layer 3 variable, defined, 80-6
- m\_code\_type, SS#7 Layer 3 variable, defined, 80-2
- m\_fib, SS#7 Layer 2 variable, defined, 79-2
- m\_frame\_addr SDLC variable, defined, 73-4 X.25 Layer 2 variable, defined, 71-4
- m\_frame\_addr\_cr, LAPD variable, defined, 77-4
- m\_frame\_addr\_sapi, LAPD variable, defined, 77-4
- m\_frame\_addr\_tei, LAPD variable, defined, 77-4
- m\_frame\_bcc\_type LAPD variable, defined, 77-4 SDLC variable, defined, 73-4 SS#7 Layer 2 variable, defined, 79-3 X.25 Layer 2 variable, defined, 71-4
- m\_frame\_cntrl\_byte\_1 LAPD variable, defined, 77-4 SDLC variable, defined, 73-4 X.25 Layer 2 variable, defined, 71-4

m\_frame\_nr, LAPD variable, defined, 77-4

- m\_frame\_ns, LAPD variable, defined, 77-4
- m\_frame\_pf LAPD variable, defined, 77-4 SDLC variable, defined, 73-4 X.25 Layer 2 variable, defined, 71-4
- m\_frame\_type LAPD variable, defined, 77-4 SDLC variable, defined, 73-4 SNA variable, 74-1 X.25 Layer 2 variable, defined, 71-4
- m\_info\_element\_len, Q.931 variable, defined, 78-3
- m\_label\_dpc, SS#7 Layer 3 variable, defined, 80-6
- m\_label\_opc, SS#7 Layer 3 variable, defined, 80-6
- m\_label\_sls, SS#7 Layer 3 variable, defined, 80-6
- m li, SS#7 Layer 2 variable, defined, 79-3

m\_lo\_dl\_il\_buff, OSI Layer 3 variable, 55-3, 63-5, 63-7, 63-8, 63-9, 63-10, 63-11, 63-52 defined, 63-18

- m\_lo\_dl\_pdu\_seg, OSI Layer 3 variable, 63-8 defined, 63-17
- m\_lo\_dl\_prmtv, OSI Layer 3 event, 55-3, 63-7 defined, 63-17
- m\_lo\_dl\_prmtv\_code, OSI Layer 3 variable, 63-8, 63-48 defined, 63-18
- m\_lo\_dl\_prmtv\_path, OSI Layer 3 variable, 63-8 defined, 63-18
- m\_lo\_dl\_sdu\_offset, OSI Layer 3 variable, 55-3, 63-8, 63-10, 63-52 defined, 63-18
- m\_lo\_dl\_sdu\_size, OSI Layer 3 variable, 55-3, 63-8, 63-52 defined, 63-18
- m\_lo\_n\_il\_buff, OSI Layer 4 variable, 63-56 defined, 63-21
- m\_lo\_n\_pdu\_seg, OSI Layer 4 variable, defined, 63-20
- m\_lo\_n\_prmtv, OSI Layer 4 event, defined, 63-20
- m\_lo\_n\_prmtv\_code, OSI Layer 4 variable, 63-52 defined, 63-21

- m\_lo\_n\_prmtv\_path, OSI Layer 4 variable, defined, 63-21
- m\_lo\_n\_sdu\_offset, OSI Layer 4 variable, 63-56 defined, 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\_du\_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\_dsaf, SNA variable, defined, 74-2
- m packet fi, SNA variable, defined, 74-3
- m\_packet\_fid\_type, SNA variable, 74-1 defined, 74-2

72-9 defined, 72-5 m\_packet\_info\_offset, X.25 Layer 3 variable, defined, 72-5m\_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\_lcn\_grp, 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\_oef, 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 rri, SNA variable, defined, 74-3 m\_packet\_rti, 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 info length, X.25 Layer 3 variable,

- 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

# Ν

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

# 0

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, 63-20 m\_lo\_p\_prmtv, 63-29 m\_lo\_ph\_prmtv, 63-29 m\_lo\_ph\_prmtv, 63-26 m\_lo\_t\_prmtv, 63-23 up dl prmtv, 63-14

up n prmtv, 63–17 up\_p\_prmtv, 63-26 up\_s\_prmtv, 63-23 up\_t\_prmtv, 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\_prmtv\_above, 63-46 send\_dl\_prmtv\_below, 63-53 send m dl prmtv above, 63-48 send\_m\_n\_prmtv\_above, 63-52 send\_m\_p\_prmtv\_above, 63-64 send\_m\_s\_prmtv\_above, 63-60 send\_m\_t\_prmtv\_above, 63-56 send\_n\_prmtv\_above, 63-50 send n prmtv below, 63-57 send\_p\_prmtv\_above, 63-62 send\_p\_prmtv\_below, 63-66 send\_ph\_prmtv\_below, 63-49 send\_ph\_to\_above, 63-44 send\_s\_prmtv\_above, 63-58 send\_s\_prmtv\_below, 63-65 send\_t\_prmtv\_above, 63-54 send t prmtv 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 12\_tick\_count, 63-16 13\_tick\_count, 63-19 14\_tick\_count, 63-22 15 tick count, 63-25 16\_tick\_count, 63-28 17\_tick\_count, 63-30 lo\_dl\_il\_buff, 63-18 lo\_dl\_pdu\_seg, 63-17 lo\_dl\_prmtv\_code, 63-17 lo\_dl\_prmtv\_path, 63-18 lo\_dl\_sdu, 63-18 lo\_n\_il\_buff, 63-21 lo\_n\_pdu\_seg, 63-20 lo\_n\_prmtv\_code, 63-20 lo\_n\_prmtv\_path, 63-21 lo\_n\_sdu, 63-21 lo\_p\_il\_buff, 63-30 lo\_p\_pdu\_seg, 63-29

lo\_p\_prmtv\_code, 63-29 lo\_p\_prmtv\_path, 63-29 lo\_p\_sdu, 63-30 lo\_ph\_il\_buff, 63-15 lo\_ph\_pdu\_seg, 63-14 lo\_ph\_prmtv\_code, 63-14 lo\_ph\_prmtv\_path, 63-15 lo\_ph\_sdu, 63-15 lo\_s\_il\_buff, 63-27 lo s pdu seg, 63-26 lo\_s\_prmtv\_code, 63-26 lo\_s\_prmtv\_path, 63-27 lo\_s\_sdu, 63-27 lo\_t\_il\_buff, 63-24 lo\_t\_pdu\_seg, 63-23 lo\_t\_prmtv\_code, 63-23 lo\_t\_prmtv\_path, 63-24 lo\_t\_sdu, 63-24 m\_lo\_dl\_il\_buff, 63-18 m\_lo\_dl\_pdu\_seg, 63-17 m\_lo\_dl\_prmtv\_code, 63-18 m\_lo\_dl\_prmtv\_path, 63-18 m\_lo\_dl\_sdu\_offset, 63-18 m\_lo\_dl\_sdu\_size, 63-18 m\_lo\_n\_il\_buff, 63-21 m\_lo\_n\_pdu\_seg, 63-20 m\_lo\_n\_prmtv\_code, 63-21 m\_lo\_n\_prmtv\_path, 63-21 m\_lo\_n\_sdu\_offset, 63-21 m\_lo\_n\_sdu\_size, 63-21 m\_lo\_p\_il\_buff, 63-30 m\_lo\_p\_pdu\_seg, 63-29 m\_lo\_p\_prmtv\_code, 63-29 m\_lo\_p\_prmtv\_path, 63-30 m\_lo\_p\_sdu\_offset, 63-30 m\_lo\_p\_sdu\_size, 63-30 m\_lo\_ph\_il\_buff, 63-15 m\_lo\_ph\_pdu\_seg, 63-14 m\_lo\_ph\_prmtv\_code, 63-14 m\_lo\_ph\_prmtv\_path, 63-15 m\_lo\_ph\_sdu\_offset, 63-15 m\_lo\_ph\_sdu\_size, 63-15 m\_lo\_s\_il\_buff, 63-27 m\_lo\_s\_pdu\_seg, 63-26 m\_lo\_s\_prmtv\_code, 63-26 m\_lo\_s\_prmtv\_path, 63-27 m\_lo\_s\_sdu\_offset, 63-27  $m_{lo_s}sdu_{size}$ , 63–27 m\_lo\_t\_il\_buff, 63-24 m\_lo\_t\_pdu\_seg, 63-23 m\_lo\_t\_prmtv\_code, 63-23 m\_lo\_t\_prmtv\_path, 63-24 m\_lo\_t\_sdu\_offset, 63-24 m\_lo\_t\_sdu\_size, 63-24 ph\_prmtv\_type, 63-13 up\_dl\_il\_buff, 63-16

up dl pdu seg, 63-15 up\_dl\_prmtv\_code, 63-16 up\_dl\_prmtv\_path, 63-16 up dl sdu, 63-16 up\_n\_il\_buff, 63-19 up\_n\_pdu\_seg, 63-19 up\_n\_prmtv\_code, 63-19 up\_n\_prmtv\_path, 63-19 up n sdu, 63-19 up\_p\_il\_buff, 63-28 up\_p\_pdu\_seg, 63-27 up\_p\_prmtv\_code, 63-28 up p prmtv path, 63-28 up\_p\_sdu, 63-28 up\_s\_il\_buff, 63-25 up\_s\_pdu\_seg, 63-24 up\_s\_prmtv\_code, 63-25 up\_s\_prmtv\_path, 63-25 up\_s\_sdu, 63-25 up\_t\_il\_buff, 63-22 up\_t\_pdu\_seg, 63-22 up\_t\_prmtv\_code, 63-22 up t prmtv path, 63-22 up\_t\_sdu, 63-22 \_open\_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

## Ρ

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

printc, 64–4 printf, 64–4 prints, 64–9

set print header, 64-8 sprintf, 64–7 structures, print buffer, 64-2 Program chaining, routines, load program, 69 - 12Protocol Trace structures l2pp\_trbuff\_ctl, 61-41 l3pp\_trbuff\_ctl, 61-42 variables l2pp\_trbuff, 61-39 l2pp trbuff end, 61–39 13pp\_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 pos\_cursor, Display Window routine, 58-1, 58-3, 61-1, 61-3, 61-8, 61-9, 61-22, 61-27, 65-22 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, 69-4prev 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

Index to C Structures, Variables, and Routines

defined, 70-3

defined, 64-2

defined, 64–4

defined, 64-4

defined, 64-9

\_print\_buffer, print structure, 64-1

printc, print routine, 64-1, 64-4

printf, print routine, 64-1, 64-6

prints, print routine, 56-22, 64-1

used to check status of print buffer, 64-1

prog\_trbuf, 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

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

routines rmt flushi, 67–8 rmt\_flusho, 67-16 rmt\_get\_baud\_rate, 67-25 rmt get bits, 67-25 rmt\_get\_mode, 67-26 rmt\_get\_parity, 67-26 rmt getc, 67-4 rmt\_getl, 67-5 rmt\_gets, 67-6 rmt\_lock, 67-9 rmt\_putb, 67-14 rmt putc, 67-11 rmt\_puts, 67-13 rmt\_resumeo, 67-18 rmt send break, 67-19 rmt\_set\_baud\_rate, 67-21 rmt\_set\_bits, 67-22 rmt\_set\_mode, 67-24 rmt\_set\_parity, 67-23 rmt\_suspendo, 67-17 rmt\_unlock, 67-10 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

SDLC variable, 73-8 defined, 73-5 X.25 Layer 2 variable, 71-8 defined, 71-5rcvd frame cntrl 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, 77-8 defined, 77-5 SDLC variable, 73-8 defined, 73-5 X.25 Layer 2 variable, 71-8 defined, 71-5 rcvd\_frame\_sdu\_size LAPD variable, 77-8 defined, 77-5 SDLC variable, 73-8 defined, 73-5 X.25 Layer 2 variable, 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\_buff\_seg, X.25 Layer 3 variable, 72-8 defined, 72-6

rcvd pkt cause, X.25 Layer 3 variable, defined, 72-4 rcvd\_pkt\_d, X.25 Layer 3 variable, defined, 72-4 rcvd\_pkt\_diagn, X.25 Layer 3 variable, defined, 72 - 4rcvd\_pkt\_info\_length, X.25 Layer 3 variable, 72-9 defined, 72-6 rcvd\_pkt\_info\_offset, X.25 Layer 3 variable, defined, 72-6rcvd\_pkt\_info\_ptr, X.25 Layer 3 variable, 72-8 defined, 72-6 rcvd\_pkt\_info\_seg, X.25 Layer 3 variable, defined, 72-6rcvd pkt lcn, X.25 Layer 3 variable, defined, 72 - 4rcvd\_pkt\_length, X.25 Layer 3 variable, 72-8 defined, 72-6rcvd\_pkt\_m, X.25 Layer 3 variable, defined, 72-4 rcvd\_pkt\_pr, X.25 Layer 3 variable, defined, 72 - 4rcvd\_pkt\_ps, X.25 Layer 3 variable, defined, 72-4 rcvd\_pkt\_q, X.25 Layer 3 variable, defined, 72 - 4rcvd\_pkt\_sdu\_offset, X.25 Layer 3 variable, 72-8 defined, 72-6 rcvd\_pkt\_type\_byte, X.25 Layer 3 variable, defined, 72-5rd\_modifier, data-display variable, 59-7, 59-18 defined, 59-5 remove, disk I/O routine, 65-3, 65-33 defined, 65-32rename, 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
- rmt\_flushi, remote port I/O routine, 67-9 defined, 67-8
- 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
- rmt\_getl, 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
- rmt\_putc, 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

routines 12 give data, 73-9 resend\_frame, 73-10 reset\_nr, 73-11 reset\_ns, 73-12 send\_frame, 73-12 structures, send\_frame\_structure, 73-2 variables 12\_current\_window\_edge, 73-5 12 enhance, 73-6 12\_lower\_window\_edge, 73-5 l2\_resend\_edge, 73-5 12\_suppress, 73-6 l2\_upper\_window\_edge, 73-5 m\_frame\_addr, 73-4 m frame bcc type, 73-4 m\_frame\_cntrl\_byte\_1, 73-4 m\_frame\_pf, 73-4 m frame type, 73-4 rcvd\_frame\_addr, 73-4 rcvd\_frame\_bcc\_type, 73-5 rcvd\_frame\_buff\_seg, 73-5 rcvd\_frame\_cntrl\_byte\_1, 73-4 rcvd frame nr, 73–5 rcvd\_frame\_ns, 73-5 rcvd frame pf, 73-4 rcvd\_frame\_sdu\_offset, 73-5 rcvd\_frame\_sdu\_size, 73-5 rcvd\_frame\_type, 73-4

#### SNA

events. See SDLC, events routines. See SDLC, routines structures. See SDLC, structures variables See also SDLC, variables m packet daf, 74-2 m\_packet\_def, 74-2 m\_packet\_dsaf, 74-2 m\_packet\_fi, 74-3 m packet fid type, 74-2 m\_packet\_length, 74-2 m\_packet\_lsid, 74-2 m packet oaf, 74-2 m packet oef, 74-2 m packet osaf, 74-2 m\_packet\_rri, 74-3 m\_packet\_rti, 74-3 m\_packet\_ru\_category, 74-3 m packet sdi, 74-3 rh\_ptr, 74-3 ru ptr, 74-3 th\_ptr, 74-3

SS#7 Layer 1, variables dce\_flags, 79-5 dce\_frames\_suppressed, 79-5 dte flags, 79-5 dte\_frames\_suppressed, 79-5 SS#7 Layer 2 events dce abort, 79-2 dce bad bcc, 79-2 dce\_frame, 79-2 dce\_good\_bcc, 79-2 dte abort, 79-2 dte\_bad\_bcc, 79-2 dte frame, 79-2 dte\_good\_bcc, 79-2 variables 12\_enhance, 79-3 12\_suppress, 79-3 m bib, 79-2 m\_fib, 79-2 m\_frame\_bcc\_type, 79-3 m\_li, 79-3 m so0, 79-3 m\_unit\_type, 79-2 SS#7 Layer 3 events dce\_packet, 80-2 dte packet, 80-2 variables 13 enhance, 80-6 13\_suppress, 80-6 m\_cic, 80-6 m\_code\_type, 80-2 m label dpc, 80-6 m\_label\_opc, 80-6 m\_label\_sls, 80-6 m\_sio\_ni, 80-2 m\_sio\_priority, 80-2 m\_sio\_si, 80-2 Stats display routines get\_68k\_phys\_addr, 62-14 send\_stat\_message, 62-15 structures, stat\_msg, 62-5 Status events, display\_screen\_changed, 61-2 structures mpm info, 66-5unit\_config, 66-3 unit setup, 66–2 variables crnt\_display\_screen, 61-2 prev\_display\_screen, 61-3

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 prmtv above, OSI Layer 2 routine, 55 - 4defined, 63-46 send\_dl\_prmtv\_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. 69-17send m dl prmtv above, OSI Layer 2 routine, defined, 63-48 send m n prmtv above, OSI Layer 3 routine, defined, 63-52 send\_m\_p\_prmtv\_above, OSI Layer 6 routine, defined, 63-64 send\_m\_s\_prmtv\_above, OSI Layer 5 routine, defined, 63-60 send m\_t\_prmtv\_above, OSI Layer 4 routine, defined, 63-56 send\_n\_prmtv\_above, OSI Layer 3 routine, defined, 63-50 send\_n\_prmtv\_below, OSI Layer 4 routine, 55-7, 55-8, 63-41, 63-58 defined, 63-57

send p prmtv above, OSI Layer 6 routine, defined, 63-62 send p prmtv 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\_prmtv\_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\_prmtv\_above, OSI Layer 5 routine, defined, 63-58 send\_s\_prmtv\_below, OSI Layer 6 routine, defined, 63-65 send\_stat\_message, stats-display routine, 62-15, 62-16 defined, 62-15 send\_t\_prmtv\_above, OSI Layer 4 routine, defined, 63-54 send\_t\_prmtv\_below, OSI Layer 5 routine, 63-41 defined, 63-61 set\_aux\_ctl\_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 routine, 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

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 buff 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 rcrd 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 rcrd 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, 61-28traces, 61-31structures l1 trbuf, 61-25 l2\_trbuf, 61-25 13\_trbuf, 61-25 14\_trbuf, 61-25 15 trbuf, 61-26 16 trbuf, 61-26 17\_trbuf, 61-26 prog trbuf, 61-25 trace\_buf, 61-24 trace\_buffer\_header, 61-24 Transmit events, fevar xmit cmplt, 59-4 routines idle\_action, 59-13 11\_il\_transmit, 55-4, 59-12 11\_transmit, 59-11 set\_tcr\_b, 59-14 structures, xmit\_list, 59-1 td\_modifier, data-display variable, 59-7, 59-17 defined, 59-4temporary\_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 - 4timeout 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, 61-24trace buffer header, trace buffer structure, 61-23, 61-32 defined, 61-24 tracec, trace buffer routine, 61-1, 61-4, 61-28 defined, 61-27tracef, trace buffer routine, 61-1, 61-4, 61-27, 61-29, 61-32, 61-34, 61-35, 65-18 defined, 61-28

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
- window\_modifier, 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

## 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 12 T1, 71-3 nr\_error, 71-3 ns error, 71-3 rcvd\_frame, 71-3 routines 12\_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 12\_enhance, 71-5 l2\_lower\_window\_edge, 71-5 12\_resend\_edge, 71-5 12 suppress, 71-5 12\_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 13\_clear\_path, 72-11 13 give data, 72-1013 more to resend, 72-11 13\_window\_empty, 72-13 13\_window\_full, 72-12 resend\_packet, 72-14 reset\_pr\_ps, 72-15 send\_packet, 72-16 structure, send\_packet\_structure, 72-2 variables 13 enhance, 72-7 13\_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, 72-6 rcvd\_pkt\_cause, 72-4 rcvd\_pkt\_d, 72-4 rcvd\_pkt\_diagn, 72-4 rcvd\_pkt\_info\_length, 72-6 rcvd pkt info offset, 72-6

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\_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

•