CASE STUDY I

FINAL REPORT DEVELOPED FOR LARGE SCALE SOFTWARE SYSTEM DESIGN OF THE AN/TYC-39 STORE AND FORWARD MESSAGE SWITCH USING THE ADA PROGRAMMING LANGUAGE

U. S. ARMY CECOM CONTRACT NO. DAAK80-81-C-0108

VOLUME III OF IV

GENERAL DYNAMICS DATA SYSTEMS DIVISION CENTRAL CENTER P. O. BOX 748 FORT WORTH, TX 76101

DISTRIBUTION STATEMENT A
Approved for public release

83 01 12 031
Ada Capability Study: Design of the Message Switching System
AN/TYC-39 Using the Ada Programming Language

4. Title and Subtitle

An Ada oriented framework for the design and documentation of the U. S. Army TYC-39 store and forward message switch (military software) system is presented. This document package contains a Requirements, Design, Ada Integrated Methodology, and Final Report section. A methodology to use Ada in specifying requirements, design, and the implementation of a system was developed. This methodology was used to redesign the TYC-39 message switch system. A selected software module was programmed after the redesign.
CASE STUDY I

FINAL REPORT DEVELOPED FOR
LARGE SCALE SOFTWARE SYSTEM DESIGN
OF THE
AN/TYC-39 STORE AND FORWARD
MESSAGE SWITCH
USING
THE ADA PROGRAMMING LANGUAGE

U. S. ARMY CECOM
CONTRACT NO. DAAK80-81-C-0108

VOLUME III OF IV

GENERAL DYNAMICS
DATA SYSTEMS DIVISION
CENTRAL CENTER
P. O. BOX 748
FORT WORTH, TX 76101
SCOPE

The system requirements herein described are a subset of the U.S. Army TYC-39 store and forward message switch. In order to understand the terminology used in this document, Chapters 1 and 2 of the Ada Integrated Methodology (developed by General Dynamics under the same contract) must be thoroughly reviewed.

The purpose of this document is to describe an Ada oriented framework for the design and documentation of an existing large scale military software system. It is not a B5 or C5 specification, although it is anticipated that the Functional Decompositional Models and Concurrency Diagrams will form a basis for a B5, while the Ada Functional Requirements should serve as a basis for C5 equivalent specifications. Hardware and software have been specified functionally in Ada form. Thus, the optimum hardware/software partitioning and hardware description is to be a part of the design process (next phase).

Due to the complexity of the switch, certain considerations were assumed as follows:

1. Modems, cryptographic, and line multiplexor equipment are external to the switch.
2. The data adapter equipment is not supported.
3. Although discussed, the operator interface is not described here. A user interactive conversational approach is recommended, but not considered a truly real time aspect of the system.
4. Y-community traffic is considered only to the extent discussed in the non-classified documents used in the project.
5. Although journal and reference storage tapes are generated, the OFF LINE functions, such as message search, are not considered.
6. Mode III, VI, and higher line protocols are not included.
The following documents, in addition to the Ada Integrated Methodology developed by General Dynamics, were used in developing this specification:

1. Performance specification, central office, communications, automatic, TT-B1-1101-0001A.
3. Communication Instructions, Tape Relay-Procedures, ACP 127(D).
**TABLE OF CONTENTS**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. System Overview</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Message Switch System Description</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Message Switch System Diagram, Node A0-1</td>
<td>3</td>
</tr>
<tr>
<td>1.3 Top Level Data Flow Diagram (DFD), Node A0</td>
<td>6</td>
</tr>
<tr>
<td>2. Functional Decomposition Models and</td>
<td>7</td>
</tr>
<tr>
<td>Ada Functional Requirements</td>
<td></td>
</tr>
<tr>
<td>2.1 Startup/Recovery</td>
<td>8</td>
</tr>
<tr>
<td>2.2 Assemble Messages</td>
<td>22</td>
</tr>
<tr>
<td>2.3 Process Messages</td>
<td>97</td>
</tr>
<tr>
<td>2.4 Transmit Messages</td>
<td>124</td>
</tr>
<tr>
<td>3. Data Dictionary and Logical Data Structures</td>
<td>162</td>
</tr>
<tr>
<td>4. Non-functional Requirements</td>
<td>183</td>
</tr>
<tr>
<td>5. Concurrency Diagram</td>
<td>187</td>
</tr>
</tbody>
</table>
(This page intentionally left blank.)
System Overview
MESSAGE SWITCH SYSTEM DESCRIPTION

The purpose of the TYC-39 message switch is to act as an automatic relay for digital message data in a battlefield functional area (BFA). The equipment is van mounted, which allows the mobility required by ground forces. Although the messages originate at a keyboard terminal, the switch is capable of interfacing to nearly all digital communications equipment in the inventory of the U.S. Army and allied armed forces.

The simplified message switch environment considered in this project is depicted in the Node AO-1 diagram. The devices on the left side, such as paper tape, Autodin, and other message switches are regarded as input, whereas, devices on the right side of the diagram are regarded as outputs from the message switch. This does not preclude the fact that message channels are capable of simultaneous two-way transmission. The Autodin network on the left and right sides of the diagram could be the same network access over a two-way channel. Any combination of up to fifty channels can be connected to the switch. The switch requires the use of non volatile storage for history of message activity and as a backup in case of memory failure. Since this study did not consider cryptographic and modem equipment, none is shown in the diagrams.

Internal message switch processing is depicted by the top level functional decomposition model (Node AO). After switch startup, program load, and database initialization, incoming messages are received, validated for proper format, and forwarded to their final destination or other relay points. The details of this process are explained in the following paragraphs.

System startup (A1) consists of switch initialization and sending the start command to each of the other functions of the switch. Startup is a relatively simple subset of system recovery, which will be discussed later.

The "operator interface" function (A2) is used to communicate with the personnel operating the system. They must be able to enter, verify, and change database information which the switch uses to determine message routing and line or trunk characteristics. Other interface commands obtain information about switch performance, cause printouts of specific messages, trace messages through the system, or exhibit various information about messages passing through the switch. The information required for the printouts comes primarily from the reference and journal files. Other operator commands deal with system shutdown, re-introduction of undeliverable messages, diagnostic capability, etc.

Although the remaining functions of the switch operate simultaneously on up to fifty different messages, for a single message, processing may be viewed as a sequential process. Activity within the message switch is created by receipt of
multiple serial bit streams (message data) at the "assemble messages" function (A3, left side). The bit streams are converted to characters, lines, and blocks until a complete message is received. The function must be capable of dealing with several different protocols, ranging from fully asynchronous with no blocking and no acknowledgement or other control to rigidly blocked synchronous traffic with block by block acknowledgement. Received messages must be validated for proper format to eliminate garbled transmissions. High precedence messages may be passed even with some specified errors. The "assemble messages" function keeps a log of its operations on each message in the reference file. It also routes a complete copy of each message as it was received to the reference file, and queues received messages by precedence for "process message".

The "process messages" function (A4) obtains received messages from the input queue, determines output routing, performs any necessary translation, and queues for transmission. Once a message has been accepted and queued, a copy is routed to the "in-transit" file (process delineated as a subset of A4). If the "in-transit" file becomes filled beyond a certain threshold, incoming messages are diverted to the "overflow" file, and are automatically reentered into the system as the "in-transit" file lower threshold is reached.

Messages may have more than one destination. When a message must be transmitted over more than one channel, the portion of the message containing the routing information (the routing line) must be edited to remove destinations not applicable to each individual channel. In addition, different destinations may require differing formats (JANAP-128 vs. ACP-127) or different character codes (ASCII vs. ITA#2). If so, these transformations are provided at this point. Routed messages are then sent back to the in-transit file unless the channel for the message is marked as closed, in which case the message is sent to the intercept file. Messages in the intercept file may be re-introduced on operator command.

The "transmit messages" function (A5) converts the routed message into a bit stream and transmits it. This function must be able to handle simultaneous operation of up to fifty lines with several different protocols. Any blocking or control characters must be added at this stage. Final validation, to include a check that the security classification limit for this line is not exceeded, is performed at this time. The transmit function also keeps a log of its activities in the journal file.

The recovery function (A1) must insure that messages being processed at the time of a system failure are not lost. To do this, the journal file is used to perform an audit of the messages received by the switch. Any message which had not completed its final delivery at the time of the failure is re-introduced into the system from the reference file, and labeled as a "suspected duplicate".
In all operations the switch must maintain high standards of reliability and throughput. The switch must operate on a priority basis, with first-in-first-out operation within precedence.
Functional Decomposition Models and Ada Functional Requirements
procedure INITIALIZE_SYSTEM is
begin
  --> SET UP SYSTEM TABLES FROM PROGRAM LOAD FILE
  --> (OR OPERATOR CONSOLE)
  --> CLEAR BUFFERS
  --> INITIALIZE LINE HANDLERS
end INITIALIZE_SYSTEM;

EOT..
procedure AUDIT_JOURNAL is
begin
  for all LOG ENTRIES in HISTORY_FILE loop
    case ENTRY_TYPE is
      when EOM_IN =>
        -- ADD MESSAGE_ID TO CURRENT_LIST
      when SVC_GEN =>
        -- ADD MESSAGE_ID TO CURRENT_LIST
      when EOM_OUT =>
        if LAST COPY then
          -- REMOVE MESSAGE_ID FROM CURRENT_LIST
        end if;
      when OVERFLOW OUT =>
        -- REMOVE MESSAGE_ID FROM CURRENT_LIST
      when OVERFLOW IN =>
        -- ADD MESSAGE_ID TO CURRENT_LIST
      when INTERCEPT OUT =>
        -- REMOVE MESSAGE_ID FROM CURRENT_LIST
      when INTERCEPT IN =>
        -- ADD MESSAGE_ID TO CURRENT_LIST
      when others =>
        null;
    end case;
    -- COLLECT LAST CSN FOR EACH LINE
  end loop;
end AUDIT_JOURNAL;
EOT..
---A122 RECOVER INTRANSIT MESSAGES
---REV BAAA
---11/24/81 PD

procedure RECOVER_INTRANSIT_MESSAGES is
begin
  for all MESSAGES ON HISTORY TAPE loop
    if MESSAGE ID ON CURRENT LIST then
      -> ADD MESSAGE TO RECOVERED MESSAGES
      -> REMOVE MESSAGE_ID FROM CURRENT_LIST
    end if;
  end loop;
  if CURRENT LIST not EMPTY then
    -> NOTIFY OPERATOR
  end if;
end RECOVER_INTRANSIT_MESSAGES;

EOT..
procedure INTRANSIT_FAILURE is
begin
  if FIRST_FAILURE then
    -- ONE COPY OF INTRANSIT STORAGE REMAINING
    --> NOTIFY OPERATOR
    if FAILED_UNIT = CURRENT_UNIT then
      CURRENT_UNIT := BACKUP_UNIT;
    end if;
  else
    -- NO COPIES OF INTRANSIT STORAGE LEFT
    --> NOTIFY OPERATOR
    --> FAIL SYSTEM
  end if;
end INTRANSIT_FAILURE;

EOT..
procedure PERFORM_OVERFLOW_AUDIT is
begin
  for all LOG ENTRIES in HISTORY_FILE loop
    if ENTRY_TYPE = OVERFLOW_OUT then
      --> ADD MESSAGE_ID TO OVERFLOW_LIST
    elsif ENTRY_TYPE = OVERFLOW_IN then
      --> DELETE MESSAGE_ID FROM OVERFLOW_LIST
      else
        null;
      end if;
  end loop;
end PERFORM_OVERFLOW_AUDIT;

EOT.
-- A1322  RECOVER OVERFLOW MESSAGES
-- REV BAAAA
-- 11/24/81 PD.

procedure RECOVER_OVERFLOW_MESSAGES is
begin
    for all MESSAGES ON HISTORY TAPE loop
        if MESSAGE ID ON OVERFLOW_LIST then
            --> ADD MESSAGE TO NEW OVERFLOW FILE
            --> REMOVE MESSAGE_ID FROM OVERFLOW_LIST
            end if;
        end loop;
        if OVERFLOW LIST not EMPTY then
            --> NOTIFY OPERATOR
            end if;
    end loop;
end RECOVER_OVERFLOW_MESSAGES;

EOT..
procedure PERFORM_INTERCEPT_AUDIT is
begin
  for all LOG ENTRIES in HISTORY FILE loop
    if ENTRY TYPE = INTERCEPT OUT then
      --> ADD MESSAGE_ID TO INTERCEPT_LIST
    elsif ENTRY TYPE = INTERCEPT IN then
      --> DELETE MESSAGE_ID FROM INTERCEPT_LIST
    else
      null;
    end if;
  end loop;
end PERFORM_INTERCEPT_AUDIT;
EOT.
procedure RECOVER_INTERCEPT_MESSAGES is
    begin
        for all MESSAGES ON HISTORY_TAPE loop
            if MESSAGE_ID ON INTERCEPT_LIST then
                --> ADD MESSAGE TO NEW INTERCEPT_FILE
                --> REMOVE MESSAGE_ID FROM INTERCEPT_LIST
            end if;
        end loop;
        if INTERCEPT_LIST not EMPTY then
            --> NOTIFY OPERATOR
        end if;
    end RECOVER_INTERCEPT_MESSAGES;

EOT..
procedure HISTORY_FAILURE is
begin
    if A COPY of HISTORY REMAINS then
        NOTIFY OPERATOR
        RECOPY REMAINING HISTORY FOR NEW BACKUP
    else
        NOTIFY OPERATOR
        START NEW HISTORY FILE
    end if;
end HISTORY_FAILURE;

EOT...
A311 SELECT_INPUT_DATA_TYPE
THE PURPOSE OF THIS ROUTINE IS TO DESCRIBE THE REQUIREMENTS
TO SELECT EITHER SYNCHRONOUS OR ASYNCHRONOUS DATA PROCESSING
DEPENDING ON THE MODE OF THE SPECIFIC MESSAGE SWITCH LINE
AS DESCRIBED IN THE FOLLOWING:
MANUAL PARAGRAPH(S)
DCAC 370-D175-1 CHAPTERS 5 AND 11
01/25/81 HF

procedure SELECT_INPUT_DATA_TYPE is
begin
  if CHANNEL_MODE for THIS_LINE = [2|4|5] then
    RECEIVE_VALIDASYNCHRONOUS_CHARACTERS;
  elsif CHANNEL_MODE for THIS_LINE = [1|3] then
    RECEIVE_VALID_SYNCHRONOUS_CHARACTERS;
  else
    DATABASE_INITIALIZATION_ERROR;
  end if;
end SELECT_INPUT_DATA_TYPE;
EOT..
procedure RECEIVE_VALIDASYNCHRONOUS_CHARACTERS is
  type PTY is (EVEN, ODD);
  PARITY : PTY;
  type BIT is (LOW, HIGH);
  BIT_SEQ : array(0..7) of BIT;
  pragma PACK(BIT);
  CLOCK_CYCLE : DURATION;
  FRAMING_ERROR : BOOLEAN;
begin
  -- ASSUMES CLOCK RATE IS 16 TIMES BAUD RATE
  -- FOR TIMING PURPOSES ASSUMES THAT INSTRUCTIONS EXECUTE IN
  -- ZERO TIME
  -- FOR IMPLEMENTATION, DELAYS MUST BE ADJUSTED FOR
  -- INSTRUCTION EXECUTION TIMES
  loop
    while ASYNCH_BIT_STREAM /= LOW loop
      delay 1.0 * CLOCK_CYCLE;
    end loop;
    delay 24.0 * CLOCK_CYCLE;
    BIT_SEQ := (0..7 => LOW);
    PARITY := EVEN;
    for I in reverse 8-LEVEL..7 loop
      BIT_SEQ(I) := ASYNCH_BIT_STREAM;
      if BIT_SEQ(I) = HIGH then
        if PARITY = EVEN then
          PARITY := ODD;
        else
          PARITY := EVEN;
        end if;
      end if;
      delay 16.0 * CLOCK_CYCLE;
    end loop;
    FRAMING_ERROR := FALSE;
    I := 1;
    loop
      if ASYNCH_BIT_STREAM = LOW then
        FRAMING_ERROR := TRUE;
      end if;
      I := I + 1;
      delay 8.0 * CLOCK_CYCLE;
      exit when I > NO_STOP_BITS;
    delay 8.0 * CLOCK_CYCLE;
  end loop;
  ASYNCH_CHAR := UNCHECKED_CONVERSION(BIT_SEQ(1..7));
  STROBE ASYNCH_CHAR;
end loop;
end RECEIVE_VALIDASYNCHRONOUS_CHARACTERS;
EOT..
-- A313 RECEIVE_VALID_SYNCHRONOUS_CHARACTERS
-- THE PURPOSE OF THIS ROUTINE IS TO DESCRIBE THE REQUIREMENTS
-- TO SYNCHRONIZE AND CONVERT A SERIAL BIT STREAM TO A CHARACTER
-- ORIENTED FORMAT, AND CHECK FOR PARITY ERRORS AS DESCRIBED IN
-- THE FOLLOWING:
-- MANUAL
-- DCAC 370-D175-1
-- 02/08/82 HF
procedure RECEIVE_VALID_SYNCHRONOUS_CHARACTERS is
begin
-- THIS REQUIREMENT IS SIMILAR IN NATURE TO A312, FOR WHICH THERE
-- IS AN EXAMPLE. IT IS EXPECTED THAT THIS FUNCTION WILL BE
-- IMPLEMENTED BY HARDWARE.
end RECEIVE_VALID_SYNCHRONOUS_CHARACTERS;
EOT.
procedure ANALYZE ASYNCH FRAMING is
begin
if START SYSTEM COMMAND or FRAMING_ERROR then
  GENERATE CONTROL CHAR;
  -- USE RT RCVD CODE (RT)
  START OF CONTROL SEQUENCE:=FALSE;
  ASYNCH_STATE:=INTERMESSAGE;
elsif STROBE ASYNCH_CHAR then
  if CHANNEL_MODE = 5 then
    if EXPIRED PAUSE TIMER then
      if PROTOCOL = ITA #2 or (ASCII and CHARACTER_PARITY = EVEN) then
        if RECEIVED CHARACTER = [ASYNCH RECEIVE CONTROL_CHAR | CANCEL CHAR | REPLY CHAR | START_CHAR] then
          START OF CONTROL SEQUENCE:=TRUE;
          -- Ctl Seq is effected if next consecutive
          -- Character is a repeat of this character.
          START_PAUSE_TIMER;
          -- If the pause timer expires before the next
          -- Character arrives, this sequence starts over.
          --> ACCEPT CONTROL CHARACTER
          --> Not to be intermixed with message characters.
        else
          ASYNCH_STATE:=INTERMESSAGE;
          -- IMPLIES THAT CHARACTER IS TO BE IGNORED.
        end if;
      else
        ASYNCH_STATE:=INTERMESSAGE;
        end if;
    else
      if START_OF CONTROL_SEQUENCE = TRUE then
        PROCESS ASYNCH CONTROL_SEQ;
      elsif ASYNCH_STATE = INTERMESSAGE then
        PROCESS INTERMESSAGE CHAR;
        START_PAUSE_TIMER;
      elsif ASYNCH_STATE = INTRAMESSAGE then
        PROCESS INTRAMESSAGE CHAR;
        if CHAR_BUFFER = UPPER_THRESHOLD then
          GENERATE CONTROL CHARs;
          -- USE 'SEND_STOP' REQUEST CODE
        end if;
      end if;
    end if;
  else
    if CHANNEL_MODE = 2 or 4 then
      if ASYNCH_STATE = INTERMESSAGE then

PROCESS_INTERMESSAGE_CHARACTER;
elself ASYNCH_STATE = INTRAMessage then
   PROCESS_INTRAMessage_CHAR;
end if;
end if;

e1s1f NO CHARACTERS RECEIVED for LAST 30 MINUTES then
   GEN SVC MSG;
   -- SVC_MESSAGE_TYPE = TRAFFIC_CHECK;
end if;
end ANALYZEASYNCH_FRAMING;

EOT..
procedure PROCESS_INTERMESSAGE_CHAR is
begin
  if PROTOCOL = ITA #2 or (ASCII and CHARACTER_PARITY = EVEN)
  then
    --> VALIDATE START OF MESSAGE SEQUENCE 'ZCZC'
    if SOM_SEQ = VALID then
      ASYNCH_STATE := INTRAMESSAGE;
      UPDATE_JOURNAL;
      -- SEND SOM_IN LOG ENTRY INFORMATION CODE
    end if;
    -- IMPLIES THAT ANY OTHER TEXT CHARACTERS WILL BE IGNORED.
  end if;
end PROCESS_INTERMESSAGE_CHAR;

EOT..
procedure PROCESS_INTRAMESSAGE_CHAR is

begin
    if PROTOCOL = ITA #2 then
        if CHARACTER = LETTERS CHAR then
            CHARACTER_CASE := LETTERS;
            --> ACCEPT CHARACTER
        elsif CHARACTER = FIGS CHAR then
            CHARACTER_CASE := FIGS;
            --> ACCEPT CHARACTER
        else
            --> ACCEPT CHARACTER
        end if;
    elsif PROTOCOL = ASCII then
        if CHARACTER_PARITY = ODD then
            --> ACCEPT CHARACTER
        end if;
    end if;

    if CHARACTER ACCEPTED then
        --> CHECK FOR VALID 'EOM' SEQUENCE (LINE FEED + NNNN)
        if VALID EOM SEQUENCE DETECTED then
            GENERATE_CONTROL_CHARS;
            --> USE EOMS_RCVD_CODE
            --> ACCEPT MESSAGE
            -- CAN BE OVERWRITTEN BY VALIDATION AND REF STORAGE.
            UPDATE_JOURNAL;
            --> SEND 'EOM IN' LOG ENTRY INFORMATION CODE
        end if;
        if CHANNEL_MODE = 5 then
            START_PAUSE_TIMER;
        end if;
    else
        ASYNCH_STATE := INTERMESSAGE;
        GENERATE_CONTROL_CHARS;
        --> USE SEND RT MESSAGE CODE. THIS WILL INITIALLY CAUSE
        -- STOP TO BE SENT OUT FROM THE XMITTER, CAUSING AN
        -- EVENTUAL REPLY TO BE RECEIVED. THEN RETRANSMIT MESSAGE
        -- (RT) WILL BE SENT OUT.
        UPDATE_JOURNAL;
        --> USE LOG_ENTRY_INFO = REJECT_ENTRY
    end if;
end PROCESS_INTRAMESSAGE_CHAR;

EOT..
-- A3144 PROCESS ASYNCH CONTROL SEQ
-- THE PURPOSE OF THIS ROUTINE IS TO DESCRIBE THE REQUIREMENTS
-- FOR PROCESSING A CONTROL CHARACTER SEQUENCE FOR A MODE 5
-- ASYNCHRONOUS TERMINAL CONNECTED TO THE MESSAGE SWITCH
-- AS DESCRIBED IN THE FOLLOWING:
-- MANUAL PARAGRAPH(S)
-- DCAC 370-D175-1 CHAPTER 11
-- 01/31/82 HF

procedure PROCESS_ASYNCH_CONTROL_SEQ is
begin
    if PROTOCOL = ITA_#2 or (ASCII and CHARACTER_PARITY = EVEN) then
        if CHARACTER /= PREVIOUS CONTROL_CHAR then
            ASYNCH_STATE:=INTERMESSAGE;
            -- IMPLIES THAT PAUSE TIMER WILL EXPIRE AND SEQ MUST
            -- RESTART.
            elsif CHARACTER = [STOP_CHAR | ACK_CHAR | RETRANSMIT_CHAR] then
                generate control chars;
                -- USE CODE REQUIRED BY THE CONTROL SEQUENCE, SUCH AS
                -- STOP_RCVD, ACK_1_RCVD, ACK_2_RCVD, OR RT_RCVD.
                start pause timer;
            elsif CHARACTER = CANCEL_CHAR then
                generate control chars;
                -- USE CANCEL_RCVD CODE. THIS ALSO IMPLIES THAT ACK
                -- ALTERNATION SEQUENCE IS SET TO TRANSMIT AN
                -- ACK 2 OUT OF THE TRANSMITTER.
                -- IN ADDITION, IF A STOP SEQUENCE IS IN PROGRESS, IT
                -- IS TO BE RESET.
                update journal;
                -- USE LOG_ENTRY_INFO = CANCEL_REC_ENTRY
                start pause timer;
            elsif CHARACTER = REPLY_CHAR then
                generate control chars;
                -- USE REPLY_RCVD CODE (MESSAGE STATUS ASSUMED TO
                -- BE RT UNLESS APPROPRIATELY ACKED.
                start pause timer;
            elsif CHARACTER /= START_CHAR then
                ASYNCH_STATE:=INTERMESSAGE;
                -- IMPLIES THAT CHARACTER SEQUENCE IS TO BE IGNORED.
                end if;
        else
            ASYNCH_STATE:=INTERMESSAGE;
        end if;
    end if;
    START_OF_CONTROL_SEQUENCE:=false;
end PROCESS_ASYNCH_CONTROL_SEQ;

EOT..
-- A3151 ANALYZE BLOCK_FRAMING
-- THE PURPOSE OF THIS ROUTINE IS TO DESCRIBE THE REQUIREMENTS
-- TO ANALYZE SYNCHRONOUS PROTOCOL AFTER BIT AND CHARACTER
-- SYNCHRONIZATION HAS BEEN PREVIOUSLY ESTABLISHED.
-- AS DESCRIBED IN THE FOLLOWING:
-- MANUAL PARAGRAPH(S)
-- DCAC 370-D175-1 CHAPTER 5
-- 01/19/82 HF

procedure ANALYZE_BLOCK_FRAMING is
begin
    if CHAR STROBE then
        if DATA SCATE LATCH = TRUE and DATA_STATE=FALSE then
            DATA_STATE:=TRUE;
            CHAR_COUNT:=1;
            START OF CONTROL SEQUENCE:=FALSE;
        elsif DATA_STATE LATCH = FALSE then
            if not PROCESS BLOCK FRAMING STATE and
               SYNCH DATA MODE::BLOCK BY_BLOCK then
                DATA_STATE:=FALSE;
                exit;
            end if;
            exit;
        end if;
        if START OF CONTROL SEQUENCE = TRUE then
            PROCESS SYNCH_CONTROL_SEQ;
            exit;
        end if;
        case CHAR_COUNT is
            when 1 =>
                PROCESS_FIRST_SYNCH_CHAR;
            when 2 =>
                PROCESS_SECOND_SYNCH_CHAR;
            when 3 .... 82 =>
                PROCESS SYNCH TEXT;
            when 83 or END OF MEDIUM SEQUENCE::TRUE =>
                PROCESS BLOCK_FRAMING;
            when 84 =>
                PROCESS_BLOCK_PARITY;
        end case;
    else
        if NO CHARACTERS RECEIVED for LAST 30 MINUTES then
            GEN SVC_MSG;
            -- SVC MESSAGE TYPE::TRAFFIC CHECK;
            -- IMPLIES RECEIVER IS EITHER OUT OF SYNC OR
            -- RECEIVING ONLY SYNC CHARACTERS.
        end if;
    end if;
end ANALYZE_BLOCK_FRAMING;

EOT..
procedure PROCESS_FIRST_SYNCH_CHAR is
begin
  if CHARACTER_PARITY = ODD then
    exit;
  end if;
  if CONTROL_CHARACTER = (REQUEST FOR ANSWER_CHAR | RECEIVE CONTROL CHAR | INVALID CHAR) then
    START_OF_CONTROL_SEQUENCE := TRUE;
    -- THE CONTROL SEQUENCE IS EFFECTED IF THE NEXT CONSECUTIVE
    -- CHARACTER IS A REPEAT OF THIS CONTROL CHARACTER.
    --> ACCEPT CHARACTER
    exit;
  end if;
  if SYNCH_DATA_MODE = BLOCK_BY_BLOCK then
    VERIFY_THAT_WE_HAVE_ANSWERED FOR THE LAST BLOCK RECEIVED;
    if WE_HAVE Not_ANSWERED then
      exit;
    end if;
  else
    if SYNCH_DATA_MODE = CONTINUOUS then
      if FIRST CHARACTER AFTER BLOCK PARITY of BLOCK in ERROR
      then
        exit;
      end if;
    end if;
    if CONTROL_CHARACTER = START_OF_HEADER then
      if ETX_BF or CANCEL not LAST_PROCESSED then
        exit;
      end if;
    elsif CONTROL_CHARACTER = START_OF_TEXT then
      if ETB_BF not LAST_CHARACTER_PROCESSED then
        exit;
      end if;
    else
      exit;
    end if;
    --> ACCEPT FIRST CHARACTER
    UPDATE_JOURNAL;
    -- USE SOM_IN_ENTRY.
    CHAR_COUNT := CHAR_COUNT + 1;
  end if;
end PROCESS_FIRST_SYNCH_CHAR;

EOT..
-- A3153 PROCESS SECOND SYNCH CHAR
-- THE PURPOSE OF THIS ROUTINE IS TO DESCRIBE THE REQUIREMENTS
-- FOR PROCESSING THE SECOND CHARACTER OF A SYNCHRONOUS INPUT
-- TRUNK TO THE MESSAGE SWITCH AS DESCRIBED IN THE FOLLOWING:
-- MANUAL PARAGRAPH(S)
-- DCAC 370-D175-1 CHAPTER 5
-- 01/17/82 HF

procedure PROCESS_SECOND_SYNCH_CHAR is
begin
  if CHARACTER_PARITY = ODD then
    CHAR_COUNT:=1;
    exit;
  end if;
  if FIRST_CHARACTER = START_OF_HEADER_CHAR then
    if CONTROL_CHARACTER = SELECT_CHAR then
      -- > ACCEPT SECOND_CHARACTER
    else
      GENERATE_CONTROL_CHARS;
      -- USE SEND_RM CODE
    end if;
  else
    if FIRST_CHARACTER = START_OF_TEXT_CHAR then
      if AUTODIN_INTERSWITCH TRUNK then
        if VALID_SECURITY_CHARACTER then
          -- > ACCEPT SECOND_CHARACTER
        else
          GENERATE_CONTROL_CHARS;
          -- USE SEND_RM CODE
        end if;
      elsif DELETE_CHAR then
        -- > ACCEPT SECOND_CHARACTER
      else
        GENERATE_CONTROL_CHARS;
        -- USE SEND_RM CODE
      end if;
    end if;
  end if;
  if SECOND_CHARACTER ACCEPTED then
    CHAR_COUNT:=CHAR_COUNT+1;
    BP_CALC:=SECOND_CHARACTER;
  end if;
end PROCESS_SECOND_SYNCH_CHAR;
EOT..
procedure PROCESS_SYNCH_TEXT is
begin
  if CHARACTER_PARITY = EVEN then
    if CONTROL_CHARACTER = MODE_CHANGE_CHAR or
       END_OF_MEDIUM_CHAR then
      --> ACCEPT CHARACTER
      CHAR_COUNT := CHAR_COUNT + 1;
      BLOCK_PARITY_CALC := BLOCK_PARITY_CALC xor CONTROL_CHAR;
      if CONTROL_CHARACTER = END_OF_MEDIUM then
        END_OF_MEDIUM_SEQ := TRUE;
      end if;
    elseif CONTROL_CHARACTER = [RECEIVE_CONTROL_CHAR ;
        INVALID CHAR] then
      START_OF_CONTROL_SEQ := TRUE;
      -- THE CONTROL SEQUENCE IS EFFECTED IF THE NEXT
      -- CONSECUTIVE CHARACTER IS IDENTICAL TO THIS ONE.
      --> ACCEPT CHARACTER
    else
      CHAR_COUNT := 1;
      GENERATE_CONTROL_CHARS;
      --> USE SEND_NAK CODE (TO BE OUTPUT ONLY AFTER BLOCK
      -- FRAMING, REPLY, OR CANCEL)
    end if;
  elseif TEXT_CHARACTER then
    --> ACCEPT CHARACTER
    CHAR_COUNT := CHAR_COUNT + 1;
    BP_CALC := CHAR_COUNT + 1;
    BP_CALC := BP_CALC xor TEXT_CHARACTER;
  end if;
end PROCESS_SYNCH_TEXT;
EOT.
-- A3155 PROCESS BLOCK_FRAMING
-- THE PURPOSE OF THIS ROUTINE IS TO DESCRIBE THE REQUIREMENTS
-- FOR PROCESSING THE BLOCK FRAMING CHARACTER POSITION OF A
-- SYNCHRONOUS INPUT TRUNK TO THE MESSAGE SWITCH
-- AS DESCRIBED IN THE FOLLOWING:
-- MANUAL PARAGRAPHS
-- DCAC 370-D175-1 CHAPTER 5
-- 01/19/82 HF

procedure PROCESS_BLOCK_FRAMING is
  begin
    if CHARACTER_PARITY = EVEN then
      if CONTROL_CHARACTER = ETX CHAR or ETB CHAR then
        if PREVIOUSLY RECEIVED BLOCK HAS BEEN ACKNOWLEDGED then
          -> ACCEPT CHARACTER
          BP_CALC:=BP_CALC xor CONTROL_CHAR;
          if ETX_CHAR then
            END OF MESSAGE_SEQ:=TRUE;
          end if;
        else
          CHAR_COUNT:=1;
          -- IMPLIES THAT TRANSMITTER IS NOT KEEPING UP.
          -- GOOD AREA TO APPLY AUTOMATIC ERROR DETECTION.
        end if;
      else
        CHAR_COUNT:=1;
        -- IMPLIES THAT CHARACTER WAS NOT ACCEPTED; WAIT FOR
        -- REPLY OR CANCEL.
      end if;
    else
      CHAR_COUNT:=1;
      -- IMPLIES THAT CHARACTER WAS NOT ACCEPTED; WAIT FOR
      -- OR CANCEL.
    end if;
  end if;
  END OF MEDIUM_SEQ:=FALSE;
end PROCESS_BLOCK_FRAMING;

EOT.
procedure PROCESS_SYNCH_CONTROL_SEQ is
begin
  if CONTROL_CHARACTER /= PREVIOUS CONTROL_CHARACTER then
    if CHAR_COUNT > 1 then
      GENERATE_CONTROL_CHARS;
      CHAR_COUNT := 1;
    end if;
  elsif CONTROL_CHARACTER = INVALID_CHAR then
    GENERATE_CONTROL_CHARS;
    USE INVALID_RCVD CODE
  elsif CONTROL_CHARACTER = RECEIVE_CONTROL_CHAR then
    GENERATE_CONTROL_CHARS;
    USE CODE REQUIRED BY CONTROL CHAR, SUCH AS WBT_RCVD,
    NAK_RCVD, RM_RCVD, ACK_1_RCVD OR ACK_2_RCVD.
  elsif PROCESSING_SYNCH_TEXT_STATE then
    GENERATE_CONTROL_CHARS;
    USE INVALID_CHAR CODE
    CHAR_COUNT := 1;
    CHANGES STATE TO PROCESS FIRST SYNCH CHAR
  elsif CONTROL_CHARACTER = REQUEST_FOR_ANSWER_CHAR then
    GENERATE_CONTROL_CHARS;
    USE CODE AS REQUIRED BY CONTROL CHARACTER, SUCH AS
    [REPLY RECEIVED | CANCEL RECEIVED | ENQUIRY RECEIVED]
    AND REPORT THAT AN ANSWER IS REQUIRED.
  if CONTROL_CHARACTER = CANCEL_CHAR then
    UPDATE_JOURNAL;
    USE CANCEL_RECEIVED CODE
  end if;
end if;
START_OF_CONTROL_SEQUENCE := FALSE;
end PROCESS_SYNCH_CONTROL_SEQ;

EOT.
-- A3157 PROCESS BLOCK PARITY
-- THE PURPOSE OF THIS ROUTINE IS TO DESCRIBE THE REQUIREMENTS
-- TO CHECK THE BLOCK PARITY CHARACTER OF A SYNCHRONOUS
-- MESSAGE AND TAKE APPROPRIATE ACTION BASED ON THE RESULT
-- AS DESCRIBED IN THE FOLLOWING:
-- MANUAL PARAGRAPH(S)
-- DCAC 370-D175-1 CHAPTER 5
-- 01/21/82 HF

procedure PROCESS_BLOCK_PARITY is
begin
  if BP CALC = CONTROL_CHAR then
    if END OF MESSAGE SEQ = TRUE then
      -- ACCEPT CHARACTER
      -- ACCEPT MESSAGE
      -- COMMENT FOR ACCEPT BLOCK ALSO APPLIES HERE.
      END_OF_MESSAGE_SEQ:=FALSE;
    else
      -- ACCEPT CHARACTER
      -- ACCEPT BLOCK
      -- CAN BE OVERRIDEN BY MESSAGE VALIDATION ROUTINES
      -- AND REFERENCE STORAGE. TIME CONSTRAINTS EXIST FOR
      -- A REPLY, AS WELL AS SYNCHRONIZATION OF THE PROCESSES.
    end if;
    CHAR_COUNT:=1;
    GENERATE_CONTROL_CHARS;
    -- USE EOMS RCVD_CODE
    UPDATE_JOURNAL;
    -- USE EOM_IN CODE
  else
    CHAR_COUNT:=1;
    GENERATE_CONTROL_CHARS;
    -- USE SEND_RM_CODE
    -- WAIT FOR REPLY OR CANCEL.
  end if;
end PROCESS_BLOCK_PARITY;

EOT..
procedure DECIDE_CHECK_SEQUENCE is
begin
  if PRECEDENCE = [W|Y|Z] then
    if CHANNEL_MODE = [2|4] then
      ECSN := ECSN + 1;
    end if;
    COMPLETE TI;
  elsif CHANNEL_MODE = [2|4] then
    CHECK MODE TI IV_CSN;
  elsif CHANNEL_MODE = 5 then
    CHECK MODE V_CSN;
  end if;
end DECIDE_CHECK_SEQUENCE;

EOT..
procedure CHECK_MODE_II_IV_CSN is
begin
  if ICSN(1..3) not in digits then
    GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE, MESSAGE_ID);
    GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
      (INCORRECT_CSN, MESSAGE_ID));
    ECSN := ECSN + 1;
  elsif ICSN /= ECSN then
    GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE, MESSAGE_ID);
    GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
      (INCORRECT_CSN, MESSAGE_ID));
    ECSN := ICSN + 1;
  else
    if LAST_GOOD_CSN + 1 /= ICSN then
      GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
        (OPEN_CSN, MESSAGE_ID, LAST_GOOD_CSN, ICSN));
    end if;
    LAST_GOOD_CSN := ICSN;
    ECSN := ICSN + 1;
  end if;
end CHECK_MODE_II_IV_CSN;
procedure CHECK_MODE_V_CSN is
begin
  if ICSN(1..3) not in digits then
    GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE, MESSAGE_ID);
    GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
      (INCORRECT_CSN, MESSAGE_ID));
    LAST_CSN_REJ := FALSE;
  elsif ICSN /= ECSN and ((LAST_CSN_REJ and ICSN /= IRCSN) or
    not LAST_CSN_REJ) then
    GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE, MESSAGE_ID);
    GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
      (INCORRECT_CSN, MESSAGE_ID));
    LAST_CSN_REJ := TRUE;
    IRCSN := ICSN;
  else
    if LAST_GOOD_CSN +1 /= ICSN then
      GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
        (OPEN_CSN, MESSAGE_ID,, LAST_GOOD_CSN, ICSN));
    end if;
    LAST_CSN_REJ := FALSE;
    LAST_GOOD_CSN := ICSN;
  end if;
end CHECK_MODE_V_CSN;

EOT.
procedure CHECK_ICD is
begin
    if ICD /= CHNL DES then
        GENERATE_CONTROL_CHARACTER( REJECT MESSAGE,MESSAGE_ID);
        GENERATE_SERVICE_MESSAGE(SERVICE MESSAGE_INFO => INVALID_HEADER_REJ,MESSAGE_ID));
    end if;
end CHECK_ICD;

EOT..
procedure COMPLETE_TI is
begin
  --> TI LINE MUST NOT EXCEED 79 CHARACTERS
  --> STARTING WITH ICD AND ENDING WITH FIRST LF
  --> TI LINE MUST END IN LOWER CASE IF ITA
  --> MINIMUM END OF LINE IS ONE LF
  if any of ABOVE not MET then
    GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
      (INVALID_HEADER_REJ,MESAGE_ID) );
    GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESAGE_ID);
    --> TERMINATE PROCESSING ON THIS MESSAGE
  end if;
end COMPLETE_TI;

EOT..
procedure DETECT_CANTRAN is
begin
  if ACP 127 or (JANAP_128 and FIRST_LMF_CHARACTER /= [S;C;B;D ;I]) then
    START := INDEX of EOM SEQUENCE; -- LF NNNN
    FINI := START - 23;
    for I in reverse FINI..START loop
      exit when MESSAGE(I) = '#';
      if MESSAGE(I..I+3) = "E E AR" then
        -- > MAKE CANTRAN REC LOG ENTRY
        -- > DISCARD MESSAGE
        -- > NOTIFY OPERATOR
        -- > TERMINATE PROCESSING ON THIS MESSAGE
        exit;
      end if;
    end loop;
  end if;
end DETECT_CANTRAN;

EOT..
procedure GENERATE ASYNCH_CTL_CHARS is
begin
  if REPLY TIMER (2-4 SECONDS) = EXPIRED then
    if MESSAGE STATUS = CANCEL SEQ then
      CONTROL_CODE:=CANCEL_SENT;
    elsif MESSAGE STATUS = REPLY SEQ then
      CONTROL_CODE:=REPLY_SENT;
    else
      raise exception;
    end if;
  end if;
  if STOP TIMER (2-8 SECONDS) = EXPIRED then
    CONTROL_CODE:=SEND_STOP;
  end if;
  case CONTROL_CODE is
    when SEND_RT =>
      SEND_ACKNOWLEDGEMENT_RESPONSE;
      -- TRANSMIT STOP CODE
      START_STOP_TIMER;
      -- EXPECT REPLY SEQUENCE
      MESSAGE_STATUS:=RT;
    when SEND_STOP =>
      if CHAR_BUFFER < LOWER THRESHOLD then
        if MESSAGE_STATUS = RT then
          SEND_ACKNOWLEDGEMENT_RESPONSE;
          -- TRANSMIT RT_CODE
          MESSAGE_STATUS:=NOT_STOP_SEQ;
        elsif MESSAGE_STATUS = OK then
          SEND_ACKNOWLEDGEMENT_RESPONSE;
          -- TRANSMIT ACK_SEQ_REPLY (ACK_1 OR ACK_2)
          -- ALTERNATE ACK_SEQ_REPLY
        elsif MESSAGE_STATUS = INCOMPLETE then
          SEND_ACKNOWLEDGEMENT_RESPONSE;
          -- TRANSMIT STOP_CODE
          MESSAGE_STATUS:=STOP_SEQ;
          -- CHECK FOR THIRD STOP TRANSMITTED AND NOTIFY
          -- OPERATOR IF SO.
        else
          raise exception;
        end if;
      elsif MESSAGE_STATUS = STOP SEQ then
        if THIRD STOP TRANSMITTED then
          -- NOTIFY OPERATOR
        end if;
      else
        raise exception;
      end if;
    else
      SEND_ACKNOWLEDGEMENT_RESPONSE;
      -- TRANSMIT STOP_CODE
      MESSAGE_STATUS:=STOP_SEQ;
  end case;
end procedure
end if;
end if;

when ACK_SEND =>
SEND ACKNOWLEDGEMENT RESPONSE;
-- TRANSMIT ACK_SEQ_REPLY (TWO ACK_1 OR ACK_2 CHARACTERS
-- ALTERNATING
-- BETWEEN MESSAGE TRANSMISSIONS.
ACK_SEQ_REPLY:=ALTERNATE of ACK_1 or ACK_2;
when CANCEL RCVD =>
ACK_SEQ_XPTD:=ACK 2;
SEND ACKNOWLEDGEMENT RESPONSE;
-- STOP MESSAGE TRANSMISSION (IF ANY)
RESET STOP TIMER (IF SET);
--> AND CLEAN UP FOR NEXT MESSAGE.
when REPLY RCVD =>
if MESSAGE_STATUS = STOPPED then
SEND ACKNOWLEDGEMENT RESPONSE;
-- TRANSMIT STOP CODE
if THIRD STOP TRANSMITTED then
--> NOTIFY OPERATOR
elsif MESSAGE_STATUS = RT then
SEND ACKNOWLEDGEMENT RESPONSE;
-- TRANSMIT RT CODE
elsif MESSAGE_STATUS = OK then
SEND ACKNOWLEDGEMENT RESPONSE;
-- TRANSMIT ACK_SEQ_REPLY
ACK_SEQ_REPLY:=ALTERNATE of ACK_SEQ_REPLY;
end if;
end if;
when RT_RCVD =>
SEND ACKNOWLEDGEMENT RESPONSE;
-- STOP MESSAGE TRANSMISSION
ACK_SEQ_XPTD:=ACK 2;
--> PREPARE MESSAGE FOR RETRANSMISSION AND WAIT
MESSAGE_STATUS:=EOMS_NOT_OUTSTANDING and NOT_REPLY_SEQ;
when STOP_RCVD =>
if MESSAGE_STATUS = CANCEL_SEQ then
CONTROL_CODE:=CANCEL_SENT;
end if;
SEND ACKNOWLEDGEMENT RESPONSE;
-- STOP MESSAGE TRANSMISSION
START STOP TIMER;
when EOMS_CODE_SENT =>
START REPLY_TIMER;
REPEAT_COUNTER:=REPEAT_COUNTER+1;
MESSAGE_STATUS:=EOMS_OUTSTANDING;
when EOMS_CODE_RCVD =>
if CHARACTER_BUFFER > UPPER THRESHOLD then
SEND ACKNOWLEDGEMENT RESPONSE;
-- TRANSMIT STOP_CODE
STOP_SEQ:=TRUE;
else
if MESSAGE_STATUS = COMPLETE then
if MESSAGE_STATUS = RT then
SEND_ACKNOWLEDGEMENT_RESPONSE;
-- TRANSMIT RT_CODE
elsif MESSAGE_STATUS = OK then
SEND_ACKNOWLEDGEMENT_RESPONSE;
-- TRANSMIT ACK_SEQ_REPLY (ACK_1 OR ACK_2)
else
    raise exception;
end if;
MESSAGE_STATUS:=EOMS_RCV;
end if;
end if;
when REPLY_SENT =>
    START_REPLY_TIMER;
    REPEAT_COUNTER:=REPEAT_COUNTER+1;
    SEND_ACKNOWLEDGEMENT_RESPONSE;
-- TRANSMIT REPLY_CODE
    MESSAGE_STATUS:=REPLY_SEQ;
when ACK_1_RCVD =>
    REPEAT_COUNTER:=0;
    RESET_REPLY TIMER;
    if ACK_SEQ_XPTD:= ACK_1_RCVD then
        ---> CONTINUE MESSAGE TRANSMISSION
    else
        SEND_ACKNOWLEDGEMENT_RESPONSE;
        -- CANCEL MESSAGE
        ---> PREPARE TO RETRANSMIT
    end if;
    ACK_SEQ_XPTD:=ACK_2;
    MESSAGE_STATUS:=EOMS_NOT_OUTSTANDING and NOT_REPLY_SEQ;
when ACK_2_RCVD =>
    REPEAT_COUNTER:=0;
    RESET_REPLY_TIMER;
    if ACK_SEQ_XPTD = ACK_2_RCVD then
        ---> CONTINUE MESSAGE TRANSMISSION
    else
        SEND_ACKNOWLEDGEMENT_RESPONSE;
        -- CANCEL MESSAGE
        ---> PREPARE TO RETRANSMIT
    end if;
    ACK_SEQ_XPTD:=ACK_1;
    MESSAGE_STATUS:=EOMS_NOT_OUTSTANDING and NOT_CANCEL_SEQ
    and NOT_REPLY_SEQ;
when CANCEL_SENT =>
    if MESSAGE_STATUS = EOMS_NOT_OUTSTANDING then
        MESSAGE_STATUS:=CANCEL_SEQ;
        ACK_SEQ_REPLY:=ACK_2;
        SEND_ACKNOWLEDGEMENT_RESPONSE;
        -- TRANSMIT CANCEL_CODE
        if REPEAT_COUNTER = 3 then
            ---> SET_OPERATOR_ALARM
        end if;
    else
        MESSAGE_STATUS:=REPLY_SEQ;
        SEND_ACKNOWLEDGEMENT_RESPONSE;
        -- TRANSMIT REPLY_CODE
    end if;

50
end if;
START_REPLY_TIMER;
REPEAT_COUNTER:=REPEAT_COUNTER + 1;
end case;
end GENERATE_ASYNCH_CTL_CHARS;

EOT..
procedure ROUTE_TO_VALIDATION is
begin
  if JANAP_128 then
    if PRECEDENCE = [W;Y;Z] then
      VALIDATE_HIPREC_JANAP_128;
    else
      VALIDATE_JANAP_128;
    end if;
  else -- ACP-127
    if DOUBLE PRECEDENCE_PROSIGN(1) = [Y;Z] or
      DOUBLE PRECEDENCE_PROSIGN(2) = [Y;Z] or
      A (GARbled) BELL SIGNAL IS PRESENT then
      VALIDATE_HIPREC_ACP_127;
    else
      VALIDATE_ACP_127;
    end if;
  end if;
end ROUTE_TO_VALIDATION;

EOT...
procedure CHECK_128_HEADER is
begin
  if PRECEDENCE /= [O|P|R] then
    raise INVALID_HEADER_REJ;
  end if;
  if LMF PAIR not LEGAL then
    --LEGAL PAIRS ARE LISTED IN THE DATA DICTIONARY
    raise INVALID_HEADER_REJ;
  end if;
  if CLASS /= [M|A|T|S|C|R|E|U] then
    raise INVALID_SCTY_FIELD;
  end if;
  if CLASS >= LINE_DATA_SECURITY_PROSIGN then
    GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
    GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
      (SCTY_MISMATCH,MESSAGE_ID));
    --> TERMINATE PROCESSING ON THIS MESSAGE
  end if;
  if not SINGLE_CARD then
    if REDUNDANT_CLASS(I..4) /= CLASS then
      raise INVALID_SCTY_FIELD;
    end if;
  end if;
  if CIC_CAI(1..3) not in LETTER or CIC_CAI(4) not in [LETTER !DIGIT] then
    raise INVALID_HEADER_REJ;
  end if;
  if OSRI(1..7) not in LETTER then
    raise INVALID_HEADER_REJ;
  end if;
  if OSSN(1..4) not in DIGIT then
    raise INVALID_HEADER_REJ;
  end if;
  if DATE_TIME (1..7) not in DIGIT then
    raise INVALID_HEADER_REJ;
  end if;
  --> CHECK SENTINELS AND SIGNALS
  --> BLANKS IN POS 9 & 21
  --> '-' IN POS 28 (33 IF RECORD COUNT PRESENT)
  --> '--' IN POS 33 (38 IF RECORD COUNT PRESENT)
  --> IF ANY ARE BAD RAISE INVALID_HEADER_REJ;
exception
  when INVALID_HEADER_REJ =>
    CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
      (INVALID_HEADER_REJ,MESSAGE_ID));
    --> TERMINATE PROCESSING ON THIS MESSAGE
  if CHANNEL_MODE = [1|3] then
    CALL GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID );
  end if;
end if;
when INVALID_SCTY_FIELD =>
    CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
        (INVALID_SCTY_FIELD, MESSAGE_ID));
    ---> TERMINATE PROCESSING ON THIS MESSAGE
if CHANNEL_MODE = [1:3] then
    CALL GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE, MESSAGE_ID);
end if;
end CHECK_128_HEADER;

EOT..
procedure CHECK_RECORD_COUNT is
begin
  if RECORD_COUNT /= ["MTMS";"PLTS"] then
    if RECORD_COUNT /= VALID_RI then
      raise INVALID_HEADER_REJ;
    end if;
  elsif RECORD_COUNT(1) in digits then
    if RECORD_COUNT(2..4) not in digits then
      raise INVALID_HEADER_REJ;
    end if;
    if RECORD_COUNT < 3 or RECORD_COUNT > 500 then
      raise INVALID_HEADER_REJ;
    end if;
  else
    raise INVALID_HEADER_REJ;
  end if;
exception
  when INVALID_HEADER_REJ =>
    CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
      (INVALID_HEADER_REJ, MESSAGE_ID));
    -- > TERMINATE PROCESSING ON THIS MESSAGE
  if CHANNEL_MODE = [113] then
    CALL GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE, MESSAGE_ID);
  end if;
end CHECK_RECORD_COUNT;
EOT..
procedure CHECK_RIS is
begin
  loop
    --> FIND NEXT RI
    if A DELIMITER OTHER THAN [`B`|CR|LF] is FOUND then
      raise INVALID_RI_FIELD;
    end if;
    exit when END_OF_ROUTING is FOUND;
    -- END_OF_ROUTING IS '.' FOR JANAP-128
    -- AND CR CR LF Z OR CR CR LF D FOR ACP-127
    if RI(1..4) not in LETTER then
      raise INVALID_RI_FIELD;
    end if;
    if RI(1..4) = OUR RI then
      --> CHECK ENTIRE RI FOR VALIDITY
    elsif RI(3..4) = "CR" then
      --> CHECK RI FOR VALID COLLECTIVE RI
    else
      --> CHECK RI(1..) FOR VALID RELAY RI
    end if;
    if RI GOOD then
      --> ADD RI TO VALID_RI_LIST
    else
      --> ADD RI TO INVALID_RI_LIST
    end if;
  end loop;
  if VALID_RI_LIST is EMPTY then
    raise ALL_RI_INVALID;
  elsif INVALID_RI_LIST not EMPTY then
    raise INVALID_RI;
  end if;
  if ACP-127 and END_OF_ROUTING = CR CR LF Z then
    --> RETURN TO CHECK_127_HEADER (THIS LINE WAS A PILOT)
  end if;
exception
  when INVALID_RI_FIELD =>
    CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
                                 (INVALID_RI_FIELD,MESSAGE_ID));
    --> TERMINATE MESSAGE PROCESSING ON THIS MESSAGE
    if CHANNEL_MODE = [1:3] then
      GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
    end if;
  when ALL_RI_INVALID =>
    CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
                                 (ALL_RI_INVALID,MESSAGE_ID,INVALID_RI_LIST));
    --> TERMINATE MESSAGE PROCESSING ON THIS MESSAGE
    if CHANNEL_MODE = [1:3] then
      GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
    end if;
  when INVALID_RI =>
CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
(INVALID_RI, MESSAGE_ID, INVALID_RI_LIST));
if PRECEDENCE = ['W', 'Y', 'Z'] then
  CIC_CA1(4) := 'W';
end if;
end CHECK_RIS;
EOT..
procedure CHECK_EOM is
begin
  if OSSN /= EOM_VALIDATION(2.5) then
    -- USE ACP-SSN FOR ACP-127
    CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
      (SUSPECTED_STRAGGLER, MESSAGE_ID));
    if PRECEDENCE = [W,Y,Z] then
      CIC:CAI(4) := 'W';
    else
      --> TERMINATE PROCESSING ON THIS MESSAGE
      if CHANNEL_MODE = [1|3] then
        GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE, MESSAGE_ID);
      end if;
    end if;
  elsif EOM_SEQ DOES not CONTAIN LF "NNNN" then
    CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
      (INVALID_EOM_REJ, MESSAGE_ID));
    --> TERMINATE PROCESSING ON THIS MESSAGE
    if CHANNEL_MODE = [1|3] then
      GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE, MESSAGE_ID);
    end if;
  end if;
  if PRECEDENCE = [W,Y,Z] then
    CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
      (HIGH_PRECEDENCE_ACCEPTANCE, MESSAGE_ID));
  end if;
  if CHANNEL_MODE = 5 then
    ECSN := ECSN + 1;
  end if;
end CHECK_EOM;

EOT..
procedure CHECK_EOT is
begin
  if not SINGLE_CARD then
    if EOT_CARD.OSSN /= ACP_SSN then
      CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
        (SUSPECTED_STRAGGLER, MESSAGE_ID));
      if PRECEDENCE = [X|Y|Z] then
        CIC_CAI(4) := 'W';
      else
        --> TERMINATE PROCESSING ON THIS MESSAGE
        if CHANNEL_MODE = [1:3] then
          GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE, MESSAGE_ID)
        end if;
      end if;
    end if;
  end if;
  if JANAP_FMT_LN_2.RECORD_COUNT = "PLTS" then
    if EOT_CARD.RECORD_COUNT < 3 or EOT_CARD.RECORD_COUNT >
      500 then
      CALL GENERATE_SERVICE_MESSAGE (SERVICE_MESSAGE_INFO =>
        (INVALID_EOM_REJ, MESSAGE_ID));
      --> TERMINATE PROCESSING ON THIS MESSAGE;
      if CHANNEL_MODE = [1:3] then
        GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE, MESSAGE_ID)
      end if;
    end if;
  else if JANAP_FMT_LN_2.RECORD_COUNT = "MTMS" then
    if EOT_CARD.RECORD_COUNT 7= "MTMS" and
      EOT_CARD.RECORD_COUNT /= ACTUAL_RECORD_COUNT then
      -- THIS IMPLIES YOU MUST COUNT THE ACTUAL NUMBER OF
      -- RECORDS
      CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
        (INVALID_EOM_REJ, MESSAGE_ID));
      --> TERMINATE PROCESSING ON THIS MESSAGE
      if CHANNEL_MODE = [1:3] then
        GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE, MESSAGE_ID)
      end if;
    end if;
  else
    if EOT_CARD.RECORD_COUNT /= ACTUAL_RECORD_COUNT then
      CALL GENERATE_SERVICE_MESSAGE (SERVICE_MESSAGE_INFO =>
        (INVALID_EOM_REJ, MESSAGE_ID));
      --> TERMINATE PROCESSING ON THIS MESSAGE
      if CHANNEL_MODE = [1:3] then
        GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE, MESSAGE_ID)
      end if;
    end if;
  end if;
end if;
end if;
else
  if SINGLE_CARD(80) /= 'N' then
    CALL GENERATE_SERVICE_MESSAGE (SERVICE_MESSAGE_INFO =>
      (INVALID_FROM_REJ, MESSAGE_ID));
    --> TERMINATE PROCESSING ON THIS MESSAGE
    if CHANNEL_MODE = [1;3] then
      GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE, MESSAGE_ID);
    end if;
    end if;
  end if;
  if PRECEDENCE = ['W'|'Y'|'Z'] then
    CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
      (HIGH_PRECEDENCE_ACCEPTANCE, MESSAGE_ID));
    end if;
  end if;
  if CHANNEL_MODE := 5 then
    ECSN := ECSN + 1;
  end if;
end CHECK_EOT;

EOT..
procedure CHECK_127_HEADER is
begin
  if DOUBLE_PRECEDENCE_PROSIGN /= ["00";"PP" ;"RR"] then
    DOUBLE_PRECEDENCE_PROSIGN := "00";
  end if;
  if ACP_FMT_LN 2(3) /= 'b' then
    CALL_GENERATED_SERVICE_MESSAGE (SERVICE_MESSAGE_INFO =>
      (INVALID_HEADER_REJ,MESSAGE_ID));
    -- > TERMINATE PROCESSING FOR THIS MESSAGE
    if CHANNEL_MODE = [1;3] then
      GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
    end if;
  end if;
end CHECK_127_HEADER;

EOT..
procedure CHECK_RIS is
  loop
    --> FIND NEXT RI
    if A DELIMITER OTHER THAN ["B", CR; LF] is FOUND then
      raise INVALID_RI_FIELD;
    end if;
    exit when END OF ROUTING is FOUND;
    -- END OF ROUTING IS ' ' FOR JANAP-128
    -- AND CR-CR LF Z OR CR LF D FOR ACP-127
    if RI(1..4) not in LETTER then
      raise INVALID_RI_FIELD;
    end if;
    if RI(1..4) = OUR RI then
      --> CHECK ENTIRE RI FOR VALIDITY
      elsif RI(3..4) = "CR" then
        --> CHECK RI FOR VALID COLLECTIVE RI
      else
        --> CHECK RI(1..) FOR VALID RELAY RI
      end if;
      if RI GOOD then
        --> ADD RI TO VALID_RI_LIST
      else
        --> ADD RI TO INVALID_RI_LIST
      end if;
    end loop;
    if VALID_RI_LIST is EMPTY then
      raise ALL_RI_INVALID;
    elsif INVALID_RI_LIST not EMPTY then
      raise INVALID_RI;
    end if;
    if ACP-127 and END OF ROUTING = CR CR LF Z then
      --> RETURN TO CHECK_127_HEADER (THIS LINE WAS A PILOT)
    end if;
  exception
    when INVALID_RI_FIELD =>
      CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
        (INVALID_RI_FIELD, MESSAGE_ID));
      --> TERMINATE MESSAGE PROCESSING ON THIS MESSAGE
      if CHANNEL_MODE = [1;3] then
        GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE, MESSAGE_ID);
      end if;
    when ALL_RI_INVALID =>
      CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
        (ALL_RI_INVALID, MESSAGE_ID, INVALID_RI_LIST));
      --> TERMINATE MESSAGE PROCESSING ON THIS MESSAGE
      if CHANNEL_MODE = [1;3] then
        GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE, MESSAGE_ID);
      end if;
    when INVALID_RI =>
      raise INVALID_RI
CALL GENERATE SERVICE MESSAGE(SERVICE_MESSAGE_INFO =>
(INVALID RI, MESSAGE_ID, INVALID RI LIST));
if PRECEDENCE = ['W' ! 'Y' ! 'Z'] then
  CIC_CA1(4) := 'W';
end if;
end CHECK_RIS;

EOT..
procedure COMPLETE_127_HEADER is
begin
  if ACP SSN PRECEDED BY '#' then
    --> RETAIN FOR EOM CHECK
  end if;
  --FMT_LN 4
  if OP_SIGNAL /= ['ZNY'..'ZNR'] or CLASS X5 /= ['MMMMM'..'AAAAA'
    [TTTTT'..'SSSSS'..'CCCCCC'..'RRRRR'..'EEEEE'..'UUUUU'] then
    call generate service message (service message info =>
      (INVALID SCTY FIELD,MESSAGE ID));
    --> TERMINATE PROCESSING FOR THIS MESSAGE
  end if;
  if CHANNEL MODE = [113] then
    generate_control_character(reject_message,MESSAGE_ID);
  end if;
  if CLASS >= LINE_DATA_SECURITY_PROSIGN then
    generate_control_character(REJECT_MESSAGE,MESSAGE_ID);
    generate_service_message(service_message_info =>
      (SCTY MISMATCH,MESSAGE_ID));
    --> TERMINATE PROCESSING ON THIS MESSAGE
  end if;
end COMPLETE_127_HEADER;
EOT.

procedure CHECK_EOM is
begin
  if OSSN /= EOM VALIDATION(2..5) then
    -- USE ACP_SSN FOR ACP-127
    CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
      (SUSPECTED_STRAGGLER, MESSAGE_ID));
    if PRECEDENCE = [W;Y;Z] then
      CIC_CAI(4) := 'W';
    else
      --> TERMINATE PROCESSING ON THIS MESSAGE
      if CHANNEL_MODE = [1;3] then
        GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
      end if;
      end if;
  elsif EOM_SEQ DOES not CONTAIN LF & "NNNN" then
    CALL GENERATE_SERVICE_MESSAGE (SERVICE_MESSAGE_INFO =>
      (INVALID_EOM_REJ,MESSAGE_ID));
    --> TERMINATE PROCESSING ON THIS MESSAGE
    if CHANNEL_MODE = [1;3] then
      GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
    end if;
    end if;
  if PRECEDENCE = [W;Y;Z] then
    CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
      (HIGH_PRECEDENCE_ACCEPTANCE,MESSAGE_ID));
  end if;
  if CHANNEL_MODE = 5 then
    ECSN := ECSN + 1;
  end if;
end CHECK_EOM;

EOT..
A3341
CHECK LMF

A3342
CHECK HP 128 HEADER

A3343
CHECK RECORD COUNT

A3344
Message Rejections

A3345
Service Message Info.
(A3324)
(A3334)
(A3354)

A3346
CHECK EOT

A334
VALIDATE HI PREC JANAP-128
procedure CHECK_LMF is
begin
  case FIRST_LMF_CHARACTER is
  when 'A' =>
    if SELECT_CHARACTER = ['H'|'S'] then
      LMF_OK := TRUE;
    else
      LMF_OK := FALSE;
    end if;
  when 'B'|'D'|'I' =>
    if SELECT_CHARACTER = ['B'|'C'] then
      LMF_OK := TRUE;
    else
      LMF_OK := FALSE;
    end if;
  when 'C'|'S' =>
    if SELECT_CHARACTER = ['D'|'F'] then
      LMF_OK := TRUE;
    else
      LMF_OK := FALSE;
    end if;
  when 'F'|'Q'|'R'|'V' =>
    if SELECT_CHARACTER = 'A' then
      LMF_OK := TRUE;
    else
      LMF_OK := FALSE;
    end if;
  when others =>
    LMF_OK := FALSE;
  end case;
  if not LMF_OK then
    case SELECT_CHARACTER is
    when A =>
      FIRST_LMF_CHARACTER := 'T';
    when 'B'|'C' =>
      FIRST_LMF_CHARACTER := 'B';
    when 'D'|'F' =>
      FIRST_LMF_CHARACTER := 'C';
    when 'H'|'S' =>
      FIRST_LMF_CHARACTER := 'A';
    end case;
    CIC_CA1(4) := 'W';
  end if;
end CHECK_LMF;

--> CHECK FOR LEGAL LMF PAIR (LEGAL PAIRS ARE LISTED IN THE
-- DATA DICTIONARY)
if not LEGAL_PAIR then
  if FIRST_LMF_CHARACTER = ['R'|'Q'|'F'] then
    SECOND_LMF_CHARACTER := 'T';
  elsif FIRST_LMF_CHARACTER = 'S' then
    SECOND_LMF_CHARACTER := 'C';
else
    SECOND_LMF_CHARACTER := FIRST_LMF_CHARACTER;
end if;
CIC_CA(4) := 'W';
end if;
end CHECK_LMF;

EOT.
procedure CHECK_HP_128_HEADER is
begin
  --> SECURITY FIELD 3 OUT OF 5 CHECK
  --> 3 OF THE 5 SECURITY CHARACTERS MUST BE CORRECT AND MATCH
  --> ELSE GENERATE INVALID SCTY FIELD SERVICE MESSAGE AND
  --> TERMINATE AND REJECT MESSAGE
  if CLASS (MATCHING 3) >= LINE_DATA.SECURITY_PROSIGN then
    GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
    GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
     (SCTY_MISMATCH,MESSAGE_ID));
    --> TERMINATE PROCESSING ON THIS MESSAGE
  end if;
  if OSRI(1..4) = OUR RI then
    --> CHECK ENTIRE RI FOR VALID TRIBUTARY
    if not VALID then
      CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
       (INVALID_HEADER_REJ,MESSAGE_ID));
      --> TERMINATE PROCESSING ON THIS MESSAGE
      if CHANNEL_MODE = [113] then
        GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
      end if;
    end if;
  end if;
  else
    --> CHECK OSRI(1..4) FOR VALID RELAY
    if not VALID then
      CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
       (INVALID_HEADER_REJ,MESSAGE_ID));
      --> TERMINATE PROCESSING ON THIS MESSAGE
      if CHANNEL_MODE = [113] then
        GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
      end if;
    end if;
  end if;
  --> CHECK OSSN FOR EMBEDDED SPACES OR HYPHENS
  if FOUND then
    CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
     (INVALID_HEADER_REJ,MESSAGE_ID));
    --> TERMINATE PROCESSING ON THIS MESSAGE
    if CHANNEL_MODE = [113] then
      GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
    end if;
  end if;
  --> CHECK SENTENALS & SIGNALS
  --> BLANKS IN POS 9 & 21
  --> '-' IN POS 28 (33 IF RECORD COUNT PRESENT)
  --> '--' IN POS 33 (38 IF RECORD COUNT PRESENT)
  --> GENERATE INVALID HEADER REJ IF ANY ARE BAD
  --> TERMINATE AND REJECT MESSAGE
end CHECK_HP_128_HEADER;
EOT..
procedure CHECK_RECORD_COUNT is
begin
  if RECORD_COUNT /= ["MTMS";"PLTS"] then
    if RECORD_COUNT(1) in LETTERS then
      if RECORD_COUNT(1) = VALID RI then
        raise INVALID_HEADER_REJ;
      end if;
    elsif RECORD_COUNT(1) in digits then
      if RECORD_COUNT(2..4) not in digits then
        raise INVALID_HEADER_REJ;
      end if;
      if RECORD_COUNT < 3 or RECORD_COUNT > 500 then
        raise INVALID_HEADER_REJ;
      end if;
    else
      raise INVALID_HEADER_REJ;
    end if;
  end if;
exception
when INVALID_HEADER_REJ =>
  CALL GENERATE_SERVICE_MESSAGE(SERVICE MESSAGE_INFO =>
    (INVALID HEADER REJ, MESSAGE ID));
  --> TERRY PROCESSING ON THIS MESSAGE
  if CHANNE. MODE = [1] then
    CALL GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE, MESSAGE_ID);
  end if;
end CHECK_RECORD_COUNT;
EOT..
procedure CHECK_RIS is
begin
  loop
    -> FIND NEXT RI
    if A DELIMITER OTHER THAN ['B'|CR|LF] is FOUND then
      raise INVALID_RI_FIELD;
    end if;
    exit when END_OF_ROUTING is FOUND;
    -- END_OF_ROUTING IS '.' FOR JANAP-128
    -- AND CR CR LF Z OR CR CR LF D FOR ACP-127
    if RI(1..4) not in LETTER then
      raise INVALID_RI_FIELD;
    end if;
    if RI(1..4) = OUR_RI then
      -> CHECK ENTIRE RI FOR VALIDITY
    elsif RI(3..4) = "CR" then
      -> CHECK RI FOR VALID COLLECTIVE RI
    else
      -> CHECK RI(1..) FOR VALID RELAY RI
    end if;
    if RI GOOD then
      -> ADD RI TO VALID_RI_LIST
    else
      -> ADD RI TO INVALID_RI_LIST
    end if;
  end loop;
  if VALID_RI_LIST is EMPTY then
    raise ALL_RI_INVALID;
  elsif INVALID_RI_LIST not EMPTY then
    raise INVALID_RI;
  end if;
  if ACP-127 and END_OF_ROUTING = CR CR LF Z then
    ---> RETURN TO CHECK_127_HEADER (THIS LINE WAS A PILOT)
  end if;
exception
  when INVALID_RI_FIELD =>
    CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
      (INVALID_RI_FIELD,MESSAGE_ID));
    ---> TERMINATE MESSAGE PROCESSING ON THIS MESSAGE
  if CHANNEL_MODE = [1|3] then
    GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
  end if;
  when ALL_RI_INVALID =>
    CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
      (ALL_RI_INVALID,MESSAGE_ID,INVALID_RI_LIST));
    ---> TERMINATE MESSAGE PROCESSING ON THIS MESSAGE
  if CHANNEL_MODE = [1|3] then
    GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
  end if;
  when INVALID_RI =>
CALL GENERATE SERVICE MESSAGE(SERVICE_MESSAGE_INFO =>
(INVALID_RI,MESSAGE_ID,INVALID_RI_LIST));
if PRECEDENCE = ['W'..'Y'..'Z'] then
  CIC_CA1(4) := 'W';
end if;
end CHECK_RIS;

EOT...
procedure CHECK_EOM is
begin
  if OSSN /= EOM VALIDATION(2..5) then
    -- USE ACP_SSN FOR ACP-127
    CALL GENERATE SERVICE MESSAGE(SERVICE MESSAGE INFO =>
      (SUSPECTED STRAGGLER, MESSAGE_ID));
    if PRECEDENCE = [W;Y;Z] then
      CIC_CA(4) := 'W';
    else
      --> TERMINATE PROCESSING ON THIS MESSAGE
      if CHANNEL MODE = [1;3] then
        GENERATE_CONTROL_CHARACTER(REJECT MESSAGE, MESSAGE_ID);
      end if;
    end if;
  elsif EOM SEQ DOES not CONTAIN LF & "NNNN" then
    CALL GENERATE SERVICE MESSAGE (SERVICE MESSAGE INFO =>
      (INVALID EOM REJ, MESSAGE_ID));
    --> TERMINATE PROCESSING ON THIS MESSAGE
    if CHANNEL MODE = [1;3] then
      GENERATE_CONTROL_CHARACTER(REJECT MESSAGE, MESSAGE_ID);
    end if;
  end if;
  if PRECEDENCE = ['W';'Y';'Z'] then
    CALL GENERATE SERVICE MESSAGE(SERVICE MESSAGE INFO =>
      (HIGH PRECEDENCE ACCEPTANCE, MESSAGE ID));
  end if;
  if CHANNEL MODE = 5 then
    ECSN := ECSN + 1;
  end if;
end CHECK_EOM;

EOT..
procedure CHECK_EOT is
begin
  if not SINGLE_CARD then
    if EOT_CARD_OSSN /= ACP SSN then
      CALL GENERATE_SERVICE_MESSAGE (SERVICE_MESSAGE_INFO =>
        (SUSPECTED_STRAGGLER, MESSAGE_ID));
      if PRECEDENCE = [X|Y|Z] then
        CIC_CAI(4) := 'W';
      else
        -- TERMINATE PROCESSING ON THIS MESSAGE
        if CHANNEL MODE = [1|3] then
          GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE, MESSAGE_ID)
        end if;
      end if;
    end if;
  end if;

  if JANAP_FMT_LN 2.RECORD_COUNT = "PLTS" then
    if EOT_CARD.RECORD_COUNT < 3 or EOT_CARD.RECORD_COUNT >
      500 then
      CALL GENERATE_SERVICE_MESSAGE (SERVICE_MESSAGE_INFO =>
        (INVALID_YOM_REJ, MESSAGE_ID));
      -- TERMINATE PROCESSING ON THIS MESSAGE;
      if CHANNEL MODE = [1|3] then
        GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE, MESSAGE_ID)
      end if;
    end if;
  end if;

  elsif JANAP_FMT_LN 2.RECORD_COUNT = "MTMS" then
    if EOT_CARD.RECORD_COUNT /= ACTUAL_RECORD_COUNT then
      -- THIS IMPLIES YOU MUST COUNT THE ACTUAL NUMBER OF
      -- RECORDS
      CALL GENERATE_SERVICE_MESSAGE (SERVICE_MESSAGE_INFO =>
        (INVALID_EOM_REJ, MESSAGE_ID));
      -- TERMINATE PROCESSING ON THIS MESSAGE
      if CHANNEL MODE = [1|3] then
        GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE, MESSAGE_ID)
      end if;
    end if;
  end if;

  else
    if EOT_CARD.RECORD_COUNT /= ACTUAL_RECORD_COUNT then
      CALL GENERATE_SERVICE_MESSAGE (SERVICE_MESSAGE_INFO =>
        (INVALID_EOM_REJ, MESSAGE_ID));
      -- TERMINATE PROCESSING ON THIS MESSAGE
      if CHANNEL MODE = [1|3] then
        GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE, MESSAGE_ID)
      end if;
    end if;
  end if;
end if;
else
  if SINGLE_CARD(80) /= 'N' then
    CALL GENERATE_SERVICE_MESSAGE (SERVICE_MESSAGE_INFO =>
      (INVALID_EOM_REJ, MESSAGE_ID));
    -- TERMINATE PROCESSING ON THIS MESSAGE
    if CHANNEL_MODE = [1:3] then
      GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE, MESSAGE_ID);
    end if;
  end if;
end if;
if PRECEDENCE = ['W','Y','Z'] then
  CALL GENERATE_SERVICE_MESSAGE (SERVICE_MESSAGE_INFO =>
    (HIGH_PRECEDENCE_ACCEPTANCE, MESSAGE_ID));
end if;
if CHANNEL_MODE := 5 then
  ECSN := ECSN + 1;
end if;
ed CHECK_EOT;

EOT..
procedure CHECK_HP_127_HEADER is
begin
  if BELL SIGNAL (or GARBLED BELL SIGNAL) is PRESENT then
     --> SEARCH FIRST 18 CHARACTERS FOR 'b'
     if not FOUND then
       CALL GENERATE SERVICE MESSAGE(SERVICE MESSAGE_INFO =>
            (INVALID HEADER REJ, MESSAGE_ID));
       --> TERMINATE PROCESSING ON THIS MESSAGE
       if CHANNEL_MODE = [1;3] then
         GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
       end if;
     end if;
  elsif ACP_FMT_LN 2(3) /= 'b' then
    CALL GENERATE SERVICE MESSAGE(SERVICE MESSAGE_INFO =>
       (INVALID HEADER REJ, MESSAGE_ID));
    --> TERMINATE PROCESSING ON THIS MESSAGE
    if CHANNEL_MODE = [1;3] then
     GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
    end if;
  end if;
end CHECK_HP_127_HEADER;
EOT.
procedure CHECK_RIS is
begin
  loop
    --> FIND NEXT RI
    if A DELIMITER OTHER THAN ['B'|CR|LF] is FOUND then
      raise INVALID_RI_FIELD;
    end if;
    exit when END_OF_ROUTING is FOUND;
    -- END_OF_ROUTING IS '.' FOR JANAP-128
    -- AND CR CR LF Z OR CR CR LF D FOR ACP-127
    if RI(1..4) not in LETTER then
      raise INVALID_RI_FIELD;
    end if;
    if RI(1..4) = OUR RI then
      --> CHECK ENTRÉ RI FOR VALIDITY
      elsif RI(3..4) = "CR" then
        --> CHECK RI FOR VALID COLLECTIVE RI
      else
        --> CHECK RI(1..) FOR VALID RELAY RI
      end if;
      if RI GOOD then
        --> ADD RI TO VALID_RI_LIST
      else
        --> ADD RI TO INVALID_RI_LIST
      end if;
    end loop;
    if VALID_RI_LIST is EMPTY then
      raise ALL_RI_INVALID;
    elsif INVALID_RI_LIST not EMPTY then
      raise INVALID_RI;
    end if;
    if ACP-127 and END_OF_ROUTING = CR CR LF Z then
      --> RETURN TO CHECK_127_HEADER (THIS LINE WAS A PILOT)
    end if;
  exception
  when INVALID_RI_FIELD =>
    CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
      INVALID_RI_FIELD,MESSAGE_ID));
    --> TERMINATE MESSAGE PROCESSING ON THIS MESSAGE
    if CHANNEL_MODE = [1|3] then
      GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
    end if;
  when ALL_RI_INVALID =>
    CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
      ALL_RI_INVALID,MESSAGE_ID,INVALID_RI_LIST));
    --> TERMINATE MESSAGE PROCESSING ON THIS MESSAGE
    if CHANNEL_MODE = [1|3] then
      GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
    end if;
  when INVALID_RI =>
CALL GENERATE SERVICE MESSAGE(SERVICE_MESSAGE_INFO => (INVALID_RI, MESSAGE_ID, INVALID_RI_LIST));
if PRECEDENCE = ['W', 'Y', 'Z'] then
    CJC_CA(4) := 'W';
end if;
end CHECK_RIS;

EOT.
procedure COMPLETE_HP_127_HEADER is
begin
  if ACP_SSN PRECEDED BY '#' then
    -- > RETAIN FOR STRAGGLER CHECK
  end if;
  -- > MAKE 3 OF 5 CHECK ON CLASS X5
  -- > 3 OUT OF THE 5 CLASS CHARACTERS MUST BE CORRECT AND AGREE
  -- > TO ESTABLISH THE CLASSIFICATION OF THE MESSAGE
  -- > IF THIS CHECK FAILS, GENERATE AN INVALID SCTY_FIELD'SVC
  -- > MESSAGE, TERMINATE AND REJECT THE MESSAGE
  if (CLASS = 'U' and OP_SIGNAL /= "ZNR") or (CLASS /= 'U' and
    OP_SIGNAL /= "ZNY") then
    CALL_GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
      (INVALID SCTY FIELD, MESSAGE_ID));
    -- > TERMINATE PROCESSING ON THIS MESSAGE
  end if;
  if CHANNEL_MODE = [1|3] then
    GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE, MESSAGE_ID);
  end if;
  if CLASS >= LINE DATA SECURITY PROSIGN then
    GENERATE_CONTROL_CHARACTER(REJECT MESSAGE, MESSAGE_ID);
    GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
      (SCTY MISMATCH, MESSAGE_ID));
    -- > TERMINATE PROCESSING ON THIS MESSAGE
  end if;
end COMPLETE_HP_127_HEADER;

EOT..
procedure CHECK_EOM is
begin
  if OSSN /= EOM VALIDATION(2..5) then
    -- USE ACP_SSN FOR ACP-127
    CALL GENERATE SERVICE MESSAGE(SERVICE_MESSAGE_INFO =>
      (SUSPECTED_STRAGGLER, MESSAGE_ID));
    if PRECEDENCE = [W|Y|Z] then
      CIC_CAI(4) := 'W';
    else
      -- TERMINATE PROCESSING ON THIS MESSAGE
      if CHANNEL_MODE = [1|3] then
        GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
      end if;
    end if;
  end if;
elsif EOM_SEQ DOES not CONTAIN LF & "NNNN" then
  CALL GENERATE SERVICE MESSAGE (SERVICE_MESSAGE_INFO =>
    (INVALID_EOM_REJ,MESSAGE_ID));
  -- TERMINATE PROCESSING ON THIS MESSAGE
  if CHANNEL_MODE = [1|3] then
    GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
  end if;
  if PRECEDENCE = ['W'|'Y'|'Z'] then
    CALL GENERATE_SERVICE MESSAGE(SERVICE_MESSAGE_INFO =>
      (HIGH_PRECEDENCE_ACCEPTANCE,MESSAGE_ID));
  end if;
  if CHANNEL_MODE = 5 then
    ECSN := ECSN + 1;
  end if;
end CHECK_EOM;

EOT..
procedure CHECK_FOR_MCB is
begin
  if MCB_RECEIVED then
    MODIFY_MCB;
  elsif JANAP_128 then
    BUILD_JANAP_MCB;
  else
    BUILD_ACP_MCB;
  end if;
end CHECK_FOR_MCB;

EOT..
procedure BUILD_JANAP_MCB is
begin
  LMFPAIR := LMFPAIR (FROM HEADER);
  CICCAI := CICCAI (FROM HEADER);
  OSRI := OSRI (FROM HEADER);
  OSSN := OSSN (FROM HEADER);
  DATETIME := DATETIME (FROM HEADER);
  if MODE = 2 or MODE = 4 or MODE = 5 then
    --> ADD ICD
    --> ADD ICSN
  else
    ICD := 'bbb';
    ICSN := 'bbbb';
  end if;
end BUILD_JANAP_MCB;

EOT..
procedure BUILD_ACP_MCB is
begin
  LMF_PAIR := FT;
  CIC_CAIR := ZYUW;
  OSRI := HOME_RELAY (?) & ICD;
  OSSN := '0' & ICSN;
  DATE_TIME := SOM_IN_DATE_TIME;
  ICD := 'bbb';
  ICSN := 'bbbb';
end BUILD_ACP_MCB;

EOT..
procedure MODIFY_MCB is
  type CT is 0..2;
  COUNT : CT;
begin
  COUNT := 0;
  for all CHARACTERS in TDW loop
    if CHARACTER = SWITCH_ID then
      COUNT := COUNT + 1;
    end if;
  end loop;
case COUNT is
  when 0 =>
    -- ADD SWITCH_ID TO FIRST ZERO FIELD
  when 1 =>
    -- ZERO ALL FIELDS AFTER POSITION OF SWITCH_ID
    -- ADD SWITCH_ID SECOND TIME
  when 2 =>
    -- HOLD MESSAGE
    -- NOTIFY OPERATOR (PROVIDE ORBIT INFORMATION)
end case;
BP := MAC(1);
for I in 2..83 loop
  BP := BP xor MCB(I);
end loop;
-- ADD SOH_TIME
end MODIFY_MCB;
procedure COMPLETE_MCB is
  begin
    -- PICK SELECT CHARACTER BASED ON LMF
    PRECEDENCE := PRECEDENCE (FROM HEADER);
    CLASS := CLASS (FROM HEADER);
    NUMBER_BLOCKS := NUMBER of BLOCKS RECEIVED;
    -- ADD SOH TIME
    if COLLECTIVE RIS in HEADER then
      TRIP = 'C';
    else
      TRIP = 'N';
    end if;
    if SERVICE MESSAGE then
      SDC := 'S';
    elsif PRECEDENCE = W then
      -- CRITIC
      if COLLECTIVE RIS in HEADER then
        if MESSAGE COMPLETE then
          SDC := 'M';
        else
          SDC := 'I';
        end if;
      else
        -- NO COLLECTIVE RIS
        if MESSAGE COMPLETE then
          SDC := 'C';
        else
          SDC := 'P';
        end if;
      end if;
    else
      SDC := 'b';
    end if;
    TDW := (1..12 => 0);
    OASC := SWITCH_ID;
    if SERVICE MESSAGE then
      SMRI := RI(2..6) (FROM HEADER);
    else
      SMRI := 'bbbbbb';
    end if;
    SDO := 'b';
    TASC := 'b';
    SASC := 'b';
    BP := MCB(1);
    for I in 2..83 loop
      BP := BP xor MCB(I);
    end loop;
  end COMPLETE_MCB;
procedure GENERATE_SERVICE_MESSAGE is
begin
  OBTAIN SVC MESSAGE FORMAT BASED ON SERVICE_MESSAGE_TYPE AND MESSAGE FORMAT
  FILL IN OSRI, OSSN, DATE_TIME
  ADD RI FROM ORIG MESSAGE OSRI
  ADD ORIG MESSAGE OSRI, OSSN, DATE_TIME AS REFERENCE
  if SERVICE_MESSAGE_TYPE = [INVALID_RI|OUTPUT_SCTY_MISMATCH |ILLEGAL_EXCHANGE| ALL RI INVALID] then
    ADD RIs
  elsif SERVICE_MESSAGE_TYPE = OPEN_CSN or INCORRECT_CSN then
    ADD CSNS
  end if;
end GENERATE_SERVICE_MESSAGE;

EOT..
procedure ROUTE_TO_STORAGE is
begin
  -- SEND A COPY OF THE MESSAGE & MCB TO REFERENCE STORAGE
  if PRECEDENCE /= W and PRECEDENCE /= Y and PRECEDENCE /= Z
    and THRESHOLD_STATUS = UPPER then
    -- ROUTE MESSAGE AND MCB TO OVERFLOW
    -- MAKE AN OVERFLOW_OUT_ENTRY
  else
    -- ROUTE MESSAGE TO INTRANSIT STORAGE
  end if;
end ROUTE_TO_STORAGE;

EOT..
procedure DETERMINE_SATURATION_LEVEL is
begin
  if OCCUPANCY_LEVEL > UPPER_THRESHOLD then
    THRESHOLD_STATUS = UPPER;
  elsif OCCUPANCY_LEVEL > MIDDLE_THRESHOLD then
    THRESHOLD_STATUS = UPPER_MIDDLE;
  elsif OCCUPANCY_LEVEL > LOWER_THRESHOLD then
    THRESHOLD_STATUS = LOWER_MIDDLE;
  else
    THRESHOLD_STATUS = LOW;
  end if;
end DETERMINE_SATURATION_LEVEL;

EOT..
procedure REENTER_FROM_OVERFLOW is
begin
  loop
    if THRESHOLD_STATUS = LOW then
      for all MESSAGES in OVERFLOW_STORAGE loop
        --> ROUTE MESSAGE TO INTRANSIT_STORAGE
        --> MAKE OVERFLOW_IN_ENTRY
      end loop;
    end if;
  end loop;
end REENTER_FROM_OVERFLOW;

EOT..
procedure SELECT_NEXT_MESSAGE is
begin
    -- > WITHIN THE HIGHEST PRIORITY MESSAGES AVAILABLE
    -- > SELECT THE EARLIEST SERVICE MESSAGE
    if NO SERVICE MESSAGES then
        -- > SELECT THE EARLIEST MESSAGE
    end if;
    end SELECT_NEXT_MESSAGE;

EOT..
procedure EXPAND_COLLECTIVE_RIS is
begin
for all RIS in HEADER loop
  if RI(3..4) = "CR" then
    --> LOOK UP in COLLECTIVE RI TABLE
    --> ASSOCIATE RI WITH LINE ON LINES FOR OUTPUT
  end if;
end loop;
end EXPAND_COLLECTIVE_RIS;
EOT..
procedure LOOK_UP_INDIVIDUAL_RIS is
begin
  for all RIS ON ROUTING LINE loop
    if RI(3..4) /= "CR" then
      --> LOOK UP RI IN ROUTING TABLE
      --> ASSOCIATE LINE FOR OUTPUT WITH RI
    end if;
  end loop;
end LOOK_UP_INDIVIDUAL_RIS;
EOT..
procedure SEGREGATE is
begin
  for NEXT RI in ROUTE.LINE loop
    if NEXT RI = RI TERMINATOR (2EH) then
      copy RI TERMINATOR TO NEW_ROUTE_LINE;
      exit loop;
    end if;
    if NEXT RI = RI in GROUP RI then
      copy NEXT RI TO NEWROUTE_LINE;
    else
      repeat until all RI in GROUP RI have been compared to
        NEXT RI;
    end if;
    if NO RI = NEXT RI then
      if OUTPUT DEVICE is DTE then
        if LMF PAIR = "SC", "CC", "BB", "DD" or "II"then
          copy an equivalent number of spaces to
            NEW_ROUTE_LINE;
        end if;
        elsif LMF FIRST = "S", "C", "B", "D" or "I"then
          copy a single space to NEW ROUTE LINE;
        elsif LMF FIRST = "T", "R", "F", "Q" or "A"then
          copy "SI"(OFH) to NEW_ROUTE_LINE;
        else
          raise LMF_ERROR;
        end if;
      end if;
    end if;
  end loop;
end SEGREGATE;

EOT.
procedure CHECK_LMFPAIR is
begin
  if SECOND_LMF_CHARACTER not ACCEPTABLE TO LINE then
    TRANSLATION_PAIR = FIRST_LMF_CHARACTER & LINE_LMF_CHARACTER;
  else
    TRANSLATION_PAIR = LMF_PAIR;
  end if;
  if TRANSLATION_PAIR /= LEGAL_EXCHANGE then
    -- > REMOVE MESSAGE FROM SYSTEM
    -- > TREAT RI AS BAD RI
    -- > GENERATE SERVICE MESSAGE
    -- > INFORM OPERATOR
  end if;
end CHECK_LMFPAIR;

EOT..
procedure CHECK_FORMAT_TRANSLATION is
begin
  -- READ FORMAT_INFO_FILE (KEY => TRANSLATION_PAIR)
  if TO 127 then
    CONVERT_TO_ACP_127;
  elsif TO T28 then
    CONVERT_TO_JANAP_128;
  else
    null;
  end if;
end CHECK_FORMAT_TRANSLATION;

EOT.
procedure CONVERT_TO_JANAP_128 is
begin
   --> DELETE ACP_FMT_LN 2, SAVING INFO
   --> GENERATE JANAP_FMT_LN 2 AS FOLLOWS
   PRECEDENCE := PRECEDENCE FROM FMT_LN 2;
   LMF_PAIR := "FT";
   CLASS := CLASS FROM FMT_LN 4;
   CIC_CAI = ?????????;
   if ORIGINATING_STATION is RELAY or TRIBUTARY OFF S&F then
      OSRI(1..4) := RELAY_RI(1..4);
   else
      OSRI(1..4) := S&F_RI(1..4);
   end if;
   OSRI(5..7) := CHANNEL ID;
   OSSN(5..7) := '0' & INCOMING CSN;
   DATE_TIME := DATE_TIME-RECEIVED;
   RIS := ACP_RIS;
   --> ADD SPACES, DASHES AND END OF ROUTING INDICATOR AS REQUIRED
end CONVERT_TO_JANAP_128;

EOT..
procedure CONVERT_TO_ACP_127 is
begin
  if FMT_LINE 4 is CORRECT or MESSAGE is SUSP_DUP then
    --> DELETE JANAP_FMT_LN_2, SAVING INFO
    -- GENERATE ACP_FMT_LN_2 AS FOLLOWS
    DOUBLE_PRECEDENCE_PROSIGN := (1,2 => PRECEDENCE);
    --> COPY RIS_Form_JANAP_FMT_LN_2
    -- GENERATE ACP_FMT_LN_3 AS FOLLOWS
    PROSIGN := "DE";
    RI := OSRI;
    ACP_SSN := OSSN;
    DATE_TIME := DATE_TIME;
    if PRECEDENCE = 'Y' or PRECEDENCE = 'Y' then
      --> INSERT <FIGS JJJJJSSSSS LTRS> BEFORE FMT_LN 2
    end if;
  if FMT_LN 4 not CORRECT then
    -- GENERATE FMT_LN 4 (ONLY ON SUSP DUP)
    if CLASS = 'U' then
      OP_SIGNAL := "ZNR";
    else
      OP_SIGNAL := "ZNY";
    end if;
    CLASS_X5 := (1,2,3,4,5 => CLASS);
  end if;
  --> ADD 12 LTRS AFTER EOM SEQUENCE
else
  --> REMOVE MESSAGE FROM SYSTEM
  --> INFORM SUPERVISOR
  --> GENERATE INVALID_RI SERVICE MESSAGE
  end if;
end CONVERT_TO_ACP_127;

EOT..
procedure CHECK_FOR_DELETE_REMOVAL is
begin
  -- READ DELETE_INFO_FILE (KEY => TRANSLATION_PAIR)
  if YES then
    REMOVE_DELETES;
  end if;
end CHECK_FOR_DELETE_REMOVAL;

EOT..
procedure REMOVE_DELETES is
begin
for all CHARACTERS in BODY loop
  if CHARACTER = SI or CHARACTER = SO then
    --> PLACE ' ' IN NEW_BODY
  elsif CHARACTER /= DEL then
    --> PLACE CHARACTER IN NEW_BODY
  end if;
end loop;
BODY := NEW_BODY;
end REMOVE_DELETES;
EOT..
procedure CHECK_FOR_CHARACTER_SET_TRANSLATION is
begin
  --> READ CHAR_SET_INFO_FILE (KEY => TRANSLATION_PAIR)
  if ASCII then
    TRANSLATE TO ASCII;
  elsif ITA then
    TRANSLATE TO ITA_2;
  else
    null;
  end if;
end CHECK_FOR_CHARACTER_SET_TRANSLATION;

EOT..
procedure TRANSLATE_TO_ASCII is
begin
  CURRENT_CASE := LTRS;
  for all CHARACTERS in MESSAGE loop
    if CHARACTER = LTRS or CHARACTER = FIGS then
      CURRENT_CASE := CHARACTER;
    end if;
  end loop;
end TRANSLATE_TO_ASCII;

EOT..
procedure TRANSLATE_TO_ITA_2 is
begin
  --> INSERT LTRS IN NEW_MESSAGE
  CURRENT_SHIFT := LTRS;
  for all_CHARACTERS in MESSAGE loop
    --> LOOK UP ITA_EQUIV AND REQUIRED_SHIFT FOR
    -- CHARACTER
    if REQUIRED_SHIFT /= CURRENT_SHIFT then
      --> INSERT REQUIRED_SHIFT
      CURRENT_SHIFT := REQUIRED_SHIFT;
    end if;
    --> INSERT ITA_EQUIV IN NEW_MESSAGE;
  end loop;
  MESSAGE := NEW_MESSAGE;
end TRANSLATE_TO_ITA_2;

EOT.
procedure CHECK_FOR_CARD_CONVERSION is
begin
  --> READ CARD_INFO_FILE ( KEY => TRANSLATION_PAIR)
  if TO CARD then
    TRANSLATE TO CARD;
  elsif FROM CARD then
    TRANSLATE_FROM_CARD;
  else
    null;
  end if;
end CHECK_FOR_CARD_CONVERSION;

EOT..
procedure TRANSLATE_TO_CARD is
begin
  -- PLACE "MTMS" IN RECORD_COUNT OF HEADER
  for all LINES in BODY loop
    -- A LINE IS THE SEQUENCE OF CHARACTERS BETWEEN CR-CR-LF
    -- SEQUENCES
    -- DETERMINE LINE LENGTH
    if LINE_LENGTH = 80 then
      OUT_LINE := LINE;
    elsif LINE_LENGTH < 80 then
      OUTLINE := (1..LINE_LENGTH => LINE, others => ' ');
    else
      -- LINE_LENGTH > 80
      OUT_LINE := LINE(1..80);
      SECOND_OUT_LINE := (1..LINE_LENGTH-80 => LINE(81..
                                      LINE_LENGTH), others => ' ');
    end if;
  end loop;
  -- DELETE EOM SEQUENCE
  -- GENERATE EOT CARD WITH "MTMS" IN RECORD_COUNT
end TRANSLATE_TO_CARD;

EOT..
procedure TRANSLATE_FROM_CARD is
begin
  if JANAP 128 then
    for EACH LINE (CARD) in body loop
      --> DELETE TRAILING SPACES
      --> ADD CR CR LF
    end loop;
  else
    for EACH LINE in body loop
      --> COUNT PRINTABLE CHARACTERS IN LINE
      if > 69 then
        --> BREAK INTO TWO LINES OF MAX 69 CHARACTERS
        --> BREAK AT WORD AS CLOSE TO 69 CHARACTERS AS POSSIBLE
      end if;
      --> ADD CR CR LF TO EACH LINE PRODUCED
    end loop;
  end if;
  --> DELETE EOT CARD
  --> ADD EOM_SEQUENCE
end TRANSLATE_FROM_CARD;

EOT..
procedure CHECK_FOR_FILL is
begin
  --> READ FILL_INFO_FILE (KEY => TRANSLATION_PAIR)
  if YES then
    FILL_LAST_BLOCK;
  end if;
end CHECK_FOR_FILL;

EOT..
procedure FILL_LAST_BLOCK is
begin
  -- FILL LAST 80 CHARACTER BLOCK OF MESSAGE (AFTER LAST 'N')
  if ASCII then
    -- FILL WITH SI
  else
    -- FILL WITH LTRS
  end if;
end FILL_LAST_BLOCK;

EOT.
procedure CHECK_LINE_AVAILABILITY is
begin
    --> LOOK UP LINE AVAILABILITY IN LINE TABLE
    if LINE AVAILABLE then
        --> PLACE MESSAGE IS INTRANSIT_STORAGE
    else
        --> MAKE SOM ENTRY IN JOURNAL
        --> PLACE MESSAGE ON INTERCEPT_STORAGE
        --> MAKE EOM ENTRY IN JOURNAL
        if VOLUME FULL then
            --> REQUEST NEW VOLUME
            --> ENTER NEW VOLUME IN INTERCEPT_VOLUMES LIST
        end if;
    end if;
end CHECK_LINE_AVAILABILITY;
EOT..
procedure REINTRODUCE_INTERCEPTED_MESSAGES is
begin
  -- DONE ON SSF COMMAND
  for all VOLUMES ON INTERCEPT_VOLUMES_LIST loop
    if VOLUME NOT MOUNTED then
      --> REQUEST MOUNT BY OPERATOR
      end if;
  for all MESSAGES ON VOLUME loop
    if MESSAGE OUTPUT LINE = LINE(S) TO BE REINTRODUCED then
      --> MAKE SOM ENTRY ON JOURNAL
      --> PLACE MESSAGE IN INTRANSIT_STORAGE
      --> MAKE EOM ENTRY ON JOURNAL
      end if;
  end loop;
end loop;
end REINTRODUCE_INTERCEPTED_MESSAGES;
EOT..
procedure SELECT_MESSAGE_FOR_OUTPUT is
begin
  ~> SELECT NEXT MESSAGE FOR OUTPUT BASED ON HIGHEST PRIORITY
  ~> WITHIN PRIORITY, SELECT ON EARLIEST TIME RECEIVED
end SELECT_MESSAGE_FOR_OUTPUT;

EOT..
procedure VALIDATE MESSAGE_LENGTH is
begin
  COMPUTED_NUMBER_OF_BLOCKS := 0;
  while FILE_STATUS /= END_OF_MESSAGE loop
    -- BUMP TO NEXT BLOCK(FILE_STATUS);
    COMPUTED_NUMBER_OF_BLOCKS := COMPUTED_NUMBER_OF_BLOCKS + 1;
  end loop;
  if COMPUTED_NUMBER_OF_BLOCKS /= NUMBER_BLOCKS
    then
      ERROR_CODE := INVALID_LENGTH_CODE;
  end if;
end;

EOT..
procedure VALIDATE_OUTPUT_RI is
begin
  --> DETERMINE RI THIS LINE
  --> DETERMINE RI OF CURRENT TRANSMISSION
  if RI THIS OUTPUT LINE /= RI THIS MESSAGE then
    ERROR_CODE := INVALID_LINE_RI;
  end if;
end;
procedure VALIDATE_LMF is
begin
  case FIRST_LMF_CHARACTER is
    when B =>
      if SECOND_LMF_CHARACTER /= 'B' then
        ERROR_CODE := INVALID_LMF;
      end if;
    when J =>
      if SECOND_LMF_CHARACTER /= 'D' then
        ERROR_CODE := INVALID_LMF;
      end if;
    when I =>
      if SECOND_LMF_CHARACTER /= 'I' then
        ERROR_CODE := INVALID_LMF;
      end if;
    when S =>
      case SECOND_LMF_CHARACTER is
        when C:A:T => null;
        when others =>
          ERROR_CODE := INVALID_LMF;
      end case;
    when C:A:T:Q:F =>
      case SECOND_LMF_CHARACTER is
        when C:A:T:T_PAREN => null;
        when others =>
          ERROR_CODE := INVALID_LMF;
      end case;
    when R =>
      case SECOND_LMF_CHARACTER is
        when T:T_PAREN => null;
        when others =>
          ERROR_CODE := INVALID_LMF;
      end case;
      when others =>
        ERROR_CODE := INVALID_LMF;
      end case;
    end case;
  end case;
end;
procedure VALIDATE_HEADER_SECURITY is
begin
  NUMBER_OF_TRANSMISSIONS := 0;
  NUMBER_OF DESTINATIONS := NUMBER_RIS_THIS_MESSAGE;
  while NUMBER_OF DESTINATIONS > 0 loop
    --> DETERMINE SECURITY OF CURRENT RI
    if CLASS > SECURITY_OF_CURRENT_RI then
      --> SAVE BAD RI
      --> DO NOT SEND TO THIS RI
      ERROR_CODE := INVALID_RI_SECURITY;
    else
      NUMBER_OF_TRANSMISSIONS := NUMBER_OF_TRANSMISSIONS + 1;
    end if;
  end loop;
  end loop;
  --> DETERMINE IF ANY RI'S WERE MISSED
  if NUMBER_OF_TRANSMISSIONS /= NO OF DESTINATIONS then
    --> PRODUCE NON-TRANSMITTED RI'S ON SERVICE MESSAGE
    GENERATE_SERVICE_MESSAGE;
  end if;
end;

EOT..
procedure INSERTS_OUTPUT_CHANNEL_ID_AND_NUMBER is
begin
  case CHANNEL_MODE is
    when 2415 =>
      --> APPEND SOH ON HEADER
      --> INSERT CHANNEL ID IN TRANSMISSION IDENTIFIER
      --> INSERT CHANNEL SEQUENCE IN TRANSMISSION IDENTIFIER
      if PRECEDENCE = 'Y' or 'Z' and
      MESSAGE_MODE = JANAP_128 then
        --> INSERT BELL CODE AFTER TRANSMISSION IDENTIFIER
      end if;
    when others =>
      null;
  end case;
  if MESSAGE CANCELLED then
    if MESSAGE_IN_TRANSMISSION then
      --> TRANSMIT CANCEL_SEQUENCE
      --> INCREMENT_CSN
    end if;
  end if;
end;

EOT..
THE PURPOSE OF THIS ROUTINE IS TO DESCRIBE THE REQUIREMENTS FOR THE FRAMING OF ASYNCHRONOUS MESSAGES AS DESCRIBED IN THE FOLLOWING:

MANUAL JANAP 128

01/28/82

procedure FRAMEASYNCHRONOUSMESSAGE is
begin
  if MESSAGE_MODE = ACP 127 ; ACP 127 MOD then
    --> PRECEDE TEXT WITH SIX "BLANKS" AND SIX "LTRS"
  elsif MESSAGE_MODE = JANAP 128 then
    --> PRECEDE TEXT WITH SIX "NULLS" AND SIX "DELETES"
  end if;
  --> FOLLOW TEXT WITH ?????????
end FRAMEASYNCHRONOUSMESSAGE;

EOT..
procedure FRAME_TRIBUTARY_MESSAGE_BLOCK_BY_BLOCK is
begin
  case SECOND LMF CHARACTER is
    when C|R|T|A|Q =>
      if BLOCK FRAMED = 1 then
        COPY MCB TO OUTPUT BUFFER
      elsif BLOCK_FRAMED /= NUMBER BLOCKS then
        PRECEDE CURRENT BLOCK WITH "STX" AND "DEL"
        AND APPEND IT WITH "ETB" AND "BP"
      end if;
    if BLOCK_FRAMED = NUMBER BLOCKS then
      PRECEDE BLOCK WITH "STX" AND "DEL"
      APPEND "ETX" AND "BP" TO BLOCK
    end if;
    when S =>
      PRECEDE BLOCK WITH "SOH" AND "D"
      APPEND "ETX" AND "BP" TO BLOCK
    when D =>
      THE DCAC 370-D175-1 DOCUMENT REFERENCES
      TO NON-EXISTENT
      FIGURE 8-1 IN DEFINING THE FORMAT FOR LMF "D"
    when B|I =>
      if BLOCK_FRAMED = 1 then
        PRECEDE THE MCB WITH "SOH" AND "D" CHARACTERS
        FOLLOW IT WITH "ETX" AND "BP" CHARACTERS
      elsif BLOCK_FRAMED = 2|NUMBER BLOCKS then
        BUILD MODE CHANGE BLOCK
        TRANSMIT MODE CHANGE BLOCK
      else
        PRECEDE MESSAGE BLOCK WITH "SOH" AND "D"
        CHARACTERS
        AND FOLLOW IT WITH ?????????????????
      end if;
    when G =>
      if BLOCK_FRAMED=1 then
        PRECEDE BLOCK OF MESSAGE WITH "SOH" AND SEL
        CHARACTERS
        AND FOLLOW IT WITH "EM", "ETB", AND "BP" CHARACTERS
      elsif BLOCK_FRAMED /= NUMBER BLOCKS then
        PRECEDE BLOCK WITH "STX" AND "DEL" CHARACTERS
        AND FOLLOW IT WITH "ETB" AND "BP" CHARACTERS
      elsif BLOCK_FRAMED = NUMBER BLOCKS then
        PRECEDE BLOCK WITH "STX" AND "DEL" CHARACTERS
        AND FOLLOW IT WITH "ETB" AND "BP" CHARACTERS
      end if;
end case;

when others =>
  ERROR_CODE := INVALID_LMF;
end case;
end FRAME_TRIBUTARY_MESSAGE_BLOCK_BY_BLOCK;
EOT..
procedure FRAME_TRUNK_MESSAGE_BLOCK_BY_BLOCK is
begin
  case SECOND_LMF_CHARACTER is
    when C|R|T|A|Q =>
      if BLOCK_COUNT = 1 then
        --> PRECEDE HEADER WITH "SOH" AND "BP"
        --> FOLLOW HEADER WITH "ETB" AND "BP"
      elsif BLOCK_COUNT /= NUMBER_BLOCKS then
        --> PRECEDE BLOCK WITH "STX" AND "SEC" CHARACTERS
        --> FOLLOW IT WITH "ETB" AND "BP" CHARACTERS
      elsif BLOCK_COUNT = NUMBER_BLOCKS then
        --> PRECEDE BLOCK WITH "STX"
        --> FOLLOW BLOCK WITH "ETX" AND "BP"
      end if;
    when S =>
      if BLOCK_COUNT = 1 then
        --> PRECEDE HEADER WITH "SOH" AND "D"
        --> FOLLOW HEADER WITH "ETB" AND "EB"
      else
        --> PRECEDE BLOCK WITH "STX" AND "U"
        --> FOLLOW BLOCK WITH "ETX" AND "BP"
      end if;
  end case;
end FRAME_TRUNK_MESSAGE_BLOCK_BY_BLOCK;

EOT..
THE PURPOSE OF THIS PROCEDURE IS TO DESCRIBE THE FRAMING OF SYNCHRONOUS MESSAGES WITH THE PROPER PROTOCOL AS DICTATED BY THE LMF OF THE RECEIVING STATION. THIS REQUIREMENT IS CONTAINED IN DCAC 370-D175-1, CHAPTER 8.

12/11/81 RCR

procedure FRAME_SYNCHRONOUS_MESSAGE_BLOCK_BY_BLOCK is begin
  case OUTPUT_LINE_TYPE is
    when TRIBUARY =>
      FRAME_TRIBUTARY_MESSAGE_BLOCK_BY_BLOCK;
    when TRUNK =>
      FRAME_TRUNK_MESSAGE_BLOCK_BY_BLOCK;
  end case;
  -- A BLOCK OF A SYNCHRONOUS BLOCK-BY-BLOCK MESSAGE IS READY TO TRANSMIT.
  -- SYNCHRONOUS CONTINUOUS MESSAGES CANNOT BE TRANSMITTED UNTIL ALL BLOCKS ARE FRAMED.
  end FRAME_SYNCHRONOUS_MESSAGE_BLOCK_BY_BLOCK;
EOT...
The purpose of this routine is to add protocol to a given message as a function of:

- Synchronous/Asynchronous
- LMF
- Trunk/Tributary
- Mode of transmission

This protocol is required by the following by the following manual references:

- Manual Paragraph
  - TT-B1-1101-0001A 3.2.1.2.15
  - 3.2.1.2.8.6
  - DCAC 370-D175-1 Chapter 5, 8, 11
- 12/15/81 RCR

procedure ADD_PROTOCOL is
begin
  case TYPE OF TRANSMISSION is
    when ASYNCHRONOUS =>
      -- The basic unit of transmission is the message.
      -- It is framed by start of messages and end of message
      -- sequences as defined in JANAP-128 and ACP-127.
      -- The asynchronous coordination is a derivative
      -- of mode I synchronous control.
      frame ASYNCHRONOUS MESSAGE;
    when SYNCHRONOUS =>
      -- The basic unit of transmission for synchronous
      -- messages is an 80 character (or less) block
      -- framed by special characters. The protocol
      -- defines unique framing for "first", "intermediate"
      -- and "last" blocks.
      BLOCK_FRAMED := 1;
      while BLOCKED_FRAMED <= NUMBER_BLOCKS loop
        frame SYNCHRONOUS MESSAGE BLOCK BY BLOCK;
        BLOCK_FRAMED := BLOCKED_FRAMED + 1;
      end loop;
      -- Since all blocks are framed, now
      -- ready to transmit synchronous continuous messages
  end case;
end;
TRANSMIT A BLOCK
THE PURPOSE OF THIS ROUTINE IS TO DESCRIBE THE REQUIREMENTS FOR THE TRANSMISSION OF A BLOCK OF A BLOCK-BY-BLOCK SYNCHRONOUS MESSAGE AS DESCRIBED BY THE FOLLOWING:

MANUAL PARAGRAPH

TT-B1-1101-0001A 3.2.1.2.15
DCAC 370-D175-1 CHAPTER 5

procedure TRANSMIT_SYNCHRONOUS_MESSAGE_BLOCK_BY_BLOCK is
begin
loop
loop
GENERATE_CONTROL_CHARACTERS;
if BLOCK_READY_TO_SEND then
exit;
else
--> SEND SYNC CHARACTERS
end if;
end loop;
TRANSMIT_START_OF_BLOCK;
TRANSMIT_BLOCK_TEXT;
if SYNC_DATA_MODE /= BLOCK_BY_BLOCK then
if ACK_RECEIVED_FOR_LAST_BLOCK /= TRUE then
TRANSMIT_CONTROL;
end if;
end if;
GENERATE_CONTROL_CHARACTERS;
TRANSMIT_END_OF_BLOCK;
if SYNC_DATA_MODE = BLOCK_BY_BLOCK then
TRANSMIT_CONTROL;
end if;
WAIT_FOR_BLOCK_ACKNOWLEDGEMENT;
-- START AND END OF MESSAGE JOURNAL ENTRIES MAY BE MADE AT THIS POINT
if BLOCK_COUNT = 1 then
OUTPUT_JOURNAL_ENTRY_TYPE := START_OF_MESSAGE_OUT_CODE;
elsif BLOCK_COUNT = NUMBER_BLOCKS then
OUTPUT_JOURNAL_ENTRY_TYPE := END_OF_MESSAGE_OUT_CODE;
end if;
--> GENERATE JOURNAL ENTRY
end loop;
end;
EOT..
procedure TRANSMIT_START_OF_MESSAGE is
  begin
    -- THE FIRST TWO FRAMING CHARACTERS WERE INSERTED
    -- IN THE TEXT ACCORDING TO MODULE A532
    --> TRANSMIT FRAMING CHARACTERS
  end;

EOT..
-- A5412
-- THE PURPOSE OF THIS PROCEDURE IS TO DESCRIBE THE
-- REQUIREMENTS FOR THE TRANSMISSION OF THE TEXT PORTION
-- OF A MESSAGE BLOCK AS CONTAINED BY THE FOLLOWING:
-- MANUAL PARAGRAPH
-- TT-B1-1101-0001A 3.2.1.2.15
-- DCAC 370-D175-1 CHAPTER 5-4(d)
-- 02/03/81 RCR

procedure TRANSMIT_BLOCK_TEXT is
begin
    CHARACTERS_THIS_BLOCK := 2;
    -- COUNTER INCLUDES TWO FRAMING CHARACTERS
    loop
        GENERATE_CONTROL_CHARACTERS;
        if CURRENT_CHARACTER = TEXT_CHARACTER!MODE_CONTROL;
            CHARACTERS_THIS_BLOCK := CHARACTERS_THIS_BLOCK + 1;
            -- UPDATE PARITY OF BLOCK
            -- SEND CURRENT CHARACTER
            if CHARACTERS_THIS_BLOCK = 82 or CURRENT_CHARACTER = EOM then
                exit;
            end if;
        end if;
    end loop;
end;
EOT.
THE PURPOSE OF THIS PROCEDURE IS TO DESCRIBE THE REQUIREMENTS FOR THE TRANSMISSION OF THE END-OF-BLOCK SEQUENCE AS CONTAINED IN THE FOLLOWING:

MANUAL PARAGRAPH
TT-B1-1101-0001A 3.2.1.2.15
DCAC 370-D175-1 CHAPTER 5-4(e)

procedure TRANSMIT_END_OF_BLOCK is
begin
  -- SEND END-OF-BLOCK SEQUENCE
  -- SEND BLOCK PARITY
  -- SET EOMS SENT CODE
  GENERATE_CONTROL_CHARACTERS;
end;
-- A5414
-- THE PURPOSE OF THIS PROCEDURE IS TO DESCRIBE THE
-- REQUIREMENTS FOR THE RECEIPT OF ACKNOWLEDGEMENTS
-- TECHNIQUE AS CONTAINED IN THE FOLLOWING:
-- MANUAL PARAGRAPH
-- TT-B1-1101-0001A 3.2.1.2.15
-- DCAC 378-D175-1 CHAPTER 5-4(e)
-- CHAPTER 3-4(a)
-- 02/03/81 RCR
procedure WAIT_FOR_BLOCK_ACKNOWLEDGEMENT is
begin
   GENERATE_CONTROL_CHARACTERS;
   -- SATISFIES THE FOLLOWING REQUIREMENTS:
   -- ACK RECEIVED := FALSE;
   -- TIMER_TRYS := 0;
   -- while TIMER_TRYS <= 3 loop
      -- RESET ACK_TIMER TO MAX VALUE
      -- (A DECREASING COUNTER ASSUMED)
      -- while ACK_TIMER > 0 loop
         -- if CONTROL_CHARACTERS_TO_BE_SENT then
            -- SEND_INDICATED_CONTROLS
         -- else
            -- SEND_SYNC_CHARACTERS
         -- end if;
      -- end loop;
      -- TIMER_TRYS := TIMER_TRYS +1
   -- SEND_REPLY_CHARACTER
end loop;
-- TIMER HAS EXPIRED THREE TIMES
-- ACTIVATE ALARM
end WAIT_FOR_BLOCK_ACKNOWLEDGEMENT;
EOT..
THE PURPOSE OF THIS ROUTINE IS TO DESCRIBE THE REQUIREMENTS FOR SENDING A RCC, RECEIVER CONTROL CHARACTER, SEQUENCE AS DESCRIBED IN THE FOLLOWING:

DCAC 370-D175-1 11-28

procedure SEND_RCC_SEQUENCE is
  begin
    GENERATE_CONTROL_CHARACTERS;

    THIS PROCEDURE SATISFIES THE FOLLOWING REQUIREMENTS:
    if SEND_GEN_CODE = FALSE then
      return;
    elsif STOP_CODE = TRUE then
      SEND_STOPSEQUENCE;
    elsif RT_CODE = TRUE then
      SEND_RTSEQUENCE;
    else if ACK_ALTERNATE = ACK_1 then
      SEND_ACK1_SEQUENCE;
    else
      SEND_ACK2_SEQUENCE;
    end if;
    end if;
    RCC_TO_SEND := FALSE;
    SEND_GEN_CODE := FALSE;
  end SEND_RCC_SEQUENCE;
EOT.
-- A5422
-- THE PURPOSE OF THIS ROUTINE IS TO DESCRIBE THE REQUIREMENTS
-- FOR REPLYING TO CANCEL COMMAND DESCRIBED IN THE FOLLOWING:
-- MANUAL
-- DCAC 370-D175-1
-- 01/14/82 RCR
procedure REPLY_TO_CAN is
begin
    GENERATE CONTROL CHARACTERS;
    THIS PROCEDURE SATISFIES THE FOLLOWING REQUIREMENTS:
    if STOP RECD CODE = TRUE then
        STOP RECD CODE := FALSE;
        THREE_RPT COUNTER := 0;
        elsif ACK RECD CODE = TRUE then
            CAN_CODE := FALSE;
            REPLY_TIMER := 0;
            THREE_RPT COUNTER := 0;
            if CAN-SENT WITHIN MESSAGE then
                --> RETURN MESSAGE TO QUEUE
                --> PREPARE TO SEND CANTRAN
                end if;
                end if;
                if REPLY TIMER EXPIRED then
                    --> RESET REPLY TIMER
                    if THIRD CAN SENT W/O RESPONSE then
                        TCC_TO_SEND := TRUE;
                    else
                        TCC_TO_SEND := TRUE;
                    end if;
                    else
                        STOP RECD CODE := FALSE;
                        THREE_RPT COUNTER := 0;
                    end if;
        end if;
        STOP RECD CODE := FALSE;
        THREE_RPT COUNTER := 0;
    end REPLY_TO_CAN;
EOT..
procedure REPLY_TO_REP_OR_STOP is
begin
    GENERATE_CONTROL_CHARACTERS;
    \text{THIS PROCEDURE SATISFIES THE FOLLOWING REQUIREMENTS:}
    if STOP_RECD_CODE = TRUE then
        \text{STOP\_RECD\_CODE} := \text{FALSE};
        \text{THREE\_RPT\_COUNTER} := 0;
    elsif ACK_RC'D_CODE = FALSE then
        \text{REP\_CODE} := \text{FALSE};
        \text{REPLY\_TIMER} := 0;
        \text{THREE\_RPT\_COUNTER} := 0;
        if RCVD_ACK = TRANS_ACK_ALT then
            if ACK_TO_COMPLETE_MSG = TRUE then
                \text{PREPARE\_TO\_COMPLETE\_MSG} := \text{TRUE};
            else
                \text{CAN\_CODE} := \text{TRUE};
            end if;
        end if;
    elseif RT_RECD_CODE = TRUE then
        \text{CAN\_CODE} := \text{TRUE};
        \text{REPLY\_TIMER} := 0;
        \text{THREE\_RPT\_COUNTER} := 0;
        exit;
    end if;
    if REPLY\_TIMER \neq 0 then
        exit;
    else
        \text{REPLY\_TIMER} := 0;
        \text{TCC\_TO\_SEND} := \text{TRUE};
        if THIRD\_REP\_SENT\_W/O\_RESPONSE then
            \text{THREE\_RPT\_ALARM} := \text{TRUE};
        end if;
    end if;
end REPLY_TO_REP_OR_STOP;
EOT..
THE PURPOSE OF THIS ROUTINE IS TO DESCRIBE THE REQUIREMENTS FOR REPLYING TO EOM COMMAND AS DESCRIBED IN THE FOLLOWING:

MANUAL PARAGRAPH(S)

DCAC 370-D175-1 11-31

01/19/82 RCR

procedure REPLY_TO_EOM is
    begin
        GENERATE_CONTROL_CHARACTERS;
        THIS PROCEDURE SATISFIES THE FOLLOWING REQUIREMENTS:
        if STOP RECD_CODE = TRUE then
            STOP RECD_CODE := FALSE;
            THRE_RPT_COUNTER := 0;
        elsif ACK RECD_CODE = TRUE then
            REPLY_TIMER := 0;
            if RECD_ACK = TRANS_ACK_ALT then
                PREPARE_TO_SEND_NEXT_MESSAGE := TRUE;
            else
                CAN_CODE := TRUE;
            end if;
        exit;
        elsif RT RECD_CODE = TRUE then
            REPLY_TIMER := 0;
            CAN_CODE := TRUE;
            if REPLY_TIMER_EXPIRED then
                REPLY_TIMER := 0;
                REP_CODE := TRUE;
                TCC_TO_SEND := TRUE;
            end if;
        end if;
        end REPLY_TO_EOM;
    EOT..
THE PURPOSE OF THIS ROUTINE IS TO DESCRIBE THE REQUIREMENTS FOR SENDING A TCC SEQUENCE AS DESCRIBED IN THE FOLLOWING:

MANUAL

DCAC 370-D175-1

01/15/82 RCR

procedure SEND_TCC_SEQUENCE is
begin
  GENERATE_CONTROL_CHARACTERS;
  this procedure satisfies the following requirements:
  if SEND GEN CODE = TRUE then
    SEND REP SEQUENCE;
  elsif CAN CODE = TRUE then
    SEND CAN SEQUENCE;
  end if;
  if REP CODE = TRUE and CAN CODE = TRUE then
    THREE_RPT_COUNTER := THREE_RPT_COUNTER + 1;
  if ACK1 = TRUE then
    ACK1 := FALSE;
  end if;
  if ACK2 = TRUE then
    ACK2 := FALSE;
  end if;
  if RT RECEIVED CODE = TRUE then
    RT RECEIVED CODE := FALSE;
  end if;
  if STOP RECEIVED CODE = TRUE then
    STOP RECEIVED CODE := FALSE;
  end if;
  START_REPLY_TIMER;
else
  SEND START SEQUENCE;
end if;
end SEND_TCC_SEQUENCE;
EOT..
THE PURPOSE OF THIS ROUTINE IS TO DESCRIBE THE REQUIREMENTS FOR THE TRANSMISSION OF THE TEXT PORTION OF A MESSAGE AS DESCRIBED IN THE FOLLOWING:

MANUAL PARAGRAPH(S)

DCAC 370-D175-1 11-24,25

procedure SELECT_ACKNOWLEDGEMENT_RESPONSE is
begin
    CAN_CODE := TRUE;
    loop
        loop
            if PAUSE_GENERATED = TRUE then
                SEND_GEN_CODE := TRUE;
                SEND_DET_CODE := FALSE;
            elsif SEND_DETECT_GENERATED = TRUE then
                SEND_DET_CODE := TRUE;
            end if;
            if RCC_TO_SEND = TRUE then
                SEND_RCC_SEQUENCE;
                exit;
            end if;
            if REPLY_TIMER_RUNNING = TRUE then
                if TIMING_REPLY_TO_CAN then
                    REPLY_TO_CAN;
                    exit;
                else if TIMING_REP_OR_STOP then
                    REPLY_TO_REP_OR_STOP;
                    exit;
                else
                    REPLY_TO_EOM;
                    exit;
                end if;
            end if;
            if TCC_TO_SEND = TRUE then
                SEND_TCC_SEQUENCE;
                exit;
            end if;
            if STOP_RECQ = TRUE then
                -- > START REPLY TIMER
                exit;
            elsif CAN_CODE = TRUE then
                exit;
            elsif MESSAGE_CHAR_TO_SEND \= TRUE then
                if CANTRAN_CHAR_TO_SEND \= TRUE then
                    exit;
                end if;
            end if;

            connector 2 of requirements starts here

            if SEND_GEN_CODE = TRUE then
                -- > SEND START SEQUENCE
                SEND_GEN_CODE := FALSE;
            end if;

            152
elsif SEND_DET_CODE = TRUE then
  exit;
end if;

--> SEND A CHARACTER
if LAST_CHAR_OF_EOMS then
  --> COMPLIMENT TRANSMIT ACK ALTERNATOR
  --> SET ALL RECEIVE CODES
  --> START REPLY TIMER
  REPLY_TO_EOM;
  exit;
end if;
end loop;
if PREPARE_TO_SEND_NEXT_MESSAGE = TRUE then
  exit;
end if;
end loop;
end SELECT_ACKNOWLEDGEMENT_RESPONSE;
EOT..
THE PURPOSE OF THIS ROUTINE IS TO TRANSMIT MESSAGES AS A FUNCTION OF THE MODE OF TRANSMISSION OF THE LINE IN USE. THIS REQUIREMENT IS CONTAINED IN THE FOLLOWING:

MANUAL PARAGRAPH(S)

TT-B1-1101-0001A 3.2.1.2.15
DCAC 370 D175-1 CHAPTER 5

12/16/81 RCR

procedure TRANSMIT_MESSAGE is
begin
  case LINE_TRANSMISSION_TYPE is
    when SYNONCHRONOUS =>
      if SYNCHRONOUS_MODE = SYNCHRONOUS_BLOCK_BY_BLOCK then
        BLOCK_COUNT := BLOCK_FRAMED;
        TRANSMIT_SYNCHRONOUS_MESSAGE_BLOCK_BY_BLOCK;
      elsif SYNCHRONOUS_MODE = SYNCHRONOUS_CONTINUOUS then
        BLOCK_COUNT := 1;
        while BLOCK_COUNT <= NUMBER_BLOCKS loop
          TRANSMIT_SYNCHRONOUS_MESSAGE_BLOCK_BY_BLOCK;
          BLOCK_COUNT := BLOCK_COUNT + 1;
        end loop;
      end if;
    when ASYNCHRONOUS =>
      TRANSMIT_SYNCHRONOUS_MESSAGE;
    when OTHERS => ERROR_CODE := INVALID_TRANSMISSION_CODE;
  end case;
end TRANSMIT_MESSAGE;

EOT.
Select type of log entry

Type of Log Entry

Start of Message Entry

End of Message Entry

Cancel Message Entry

Scrubbed Message Entry

CANTRAN Message

Selected Message

Log Entry

A555

UPDATE JOURNAL
procedure START_OF_MESSAGE_OUT_JOURNAL_ENTRY is
  begin
    OUTGOING_LOG_ENTRY := SOM_OUT_ENTRY;
    -- CREATE A JOURNAL ENTRY WHICH CONTAINS:
    -- OUTPUT CHANNEL ID
    -- CHANNEL SEQUENCE NUMBER, IF APPLICABLE
    -- CHANNEL MODE OF OPERATION
    -- ROUTING INDICATOR
    -- HEADER INFORMATION
    -- TIME SOM TRANSMITTED
    -- S&F ASSIGNED SERIAL NUMBER
  end START_OF_MESSAGE_OUT_JOURNAL_ENTRY;
EOT..
procedure END_OF_MESSAGE_OUT_JOURNAL_ENTRY is
begin
  OUTGOING_LOG_ENTRY := EOM_OUT_ENTRY;
  -- CREATE A JOURNAL ENTRY WHICH CONTAINS:
  -- ENTRY IDENTIFICATION
  -- OUTPUT CHANNEL ID
  -- TIME OF TRANSMISSION
  -- CHANNEL SEQUENCE NUMBER
  -- DESTINATIONS FOR THIS MESSAGE
  -- S&F ASSIGNED SERIAL NUMBER
  -- BLOCK COUNT
end END_OF_MESSAGE_OUT_JOURNAL_ENTRY;
EOT..
-- A553
-- THE PURPOSE OF THIS ROUTINE IS TO DESCRIBE THE REQUIREMENTS
-- FOR CANCEL MESSAGE JOURNAL ENTRIES AS DESCRIBED IN THE FOLLOWING:
-- MANUAL PARAGRAPH(S)
-- TT-B1-1101-0001A 3.2.1.2.14.1.2(c)
-- 01/19/82 RCR

procedure CANCEL_MESSAGE_OUT_JOURNAL_ENTRY is
begin
  OUTGOING_LOG_ENTRY := CANCEL_OUT_ENTRY;
  -- CREATE A JOURNAL ENTRY WHICH CONTAINS:
  -- CHANNEL ID AND CSN
  -- CHANNEL MODE OF OPERATION
  -- TIME OF CANCELLATION
  -- REASON FOR CANCELLING
  -- ROUTING INDICATORS OF MESSAGE
  -- S&F MODULE ASSIGNED SERIAL NUMBER
  -- BLOCK COUNT
end CANCEL_MESSAGE_OUT_JOURNAL_ENTRY;

EOT..
-- A554
-- THE PURPOSE OF THIS ROUTINE IS TO DESCRIBE THE REQUIREMENTS
-- FOR SCRUBBED MESSAGE JOURNAL ENTRIES AS DESCRIBED IN THE FOLLOWING:
-- MANUAL PARAGRAPH(S)
-- TT-B1-1101-0001A 3.2.1.2.14.1.2(d)
-- 01/19/82 RCR

procedure SCRUBBED_MESSAGE_OUT_JOURNAL_ENTRY is
    begin
        OUTGOING LOG ENTRY := SCRUB ENTRY;
        -- CREATE A JOURNAL ENTRY WHICH CONTAINS:
        -- CHANNEL ID AND CSN
        -- CHANNEL MODE OF OPERATION
        -- TIME SCRUBBED
        -- S&F MODULE ASSIGNED SERIAL NUMBER
        -- ROUTING INDICATORS OF MESSAGE
        -- REASON FOR SCRUB
        -- BLOCK COUNT
        if MESSAGE OUT SENT TO OVERFLOW then
            -- INCLUDE OVERFLOW MESSAGE SERIAL NUMBER
        end if;
        if MESSAGE SENT TO INTERCEPT then
            -- INCLUDE INTERCEPT MESSAGE SERIAL NUMBER
        end if;
    end SCRUBBED_MESSAGE_OUT_JOURNAL_ENTRY;

EOT..
--- THE PURPOSE OF THIS ROUTINE IS TO DESCRIBE THE
--- REQUIREMENTS FOR JOURNAL ENTRY CREATION AS A RESULT
--- OF MESSAGE OUTPUT PROCESSING AS DESCRIBED IN THE
--- FOLLOWING:
--- MANUAL PARAGRAPHS
--- TT-B1-1101-0001A 3.2.1.2.14.1.2
--- 12/15/81

procedure UPDATE_JOURNAL is
begin
  case JOURNAL_ENTRY_TYPE is
  when START_OF_MESSAGE_CODE =>
    START_OF_MESSAGE_OUT_JOURNAL_ENTRY;
  when END_OF_MESSAGE_CODE =>
    END_OF_MESSAGE_OUT_JOURNAL_ENTRY;
  when CANCEL_MESSAGE_CODE =>
    CANCEL_MESSAGE_OUT_JOURNAL_ENTRY;
  when SCRUBBED_MESSAGE_CODE =>
    SCRUBBED_MESSAGE_OUT_JOURNAL_ENTRY;
  when CANTTRAN_MESSAGE_CODE =>
    CANTTRAN_MESSAGE_OUT_JOURNAL_ENTRY;
  end case;
end UPDATE_JOURNAL;

EOT..
DATA DICTIONARY
FEBRUARY 8, 1982

O1A3
ACKNOWLEDGMENT
= [ACK REQUEST|ACK RECEIVED]
* USED TO PASS REQUEST TO
SEND ACK OR TO PASS ACKS
WHICH HAVE BEEN RECEIVED TO
THE TRANSMIT FUNCTION *

ACP_FMT_LN_1
= 'VZCZC' + ICD + FIGS
+ ICSN + LTRS + <5>'b'
+ ([UU|HH]) + LN END
* TRANSMISSION IDENT (TI) *

ACP_FMT_LN_2
= (BELL SIGNAL) +
DOUBLE PRECEDENCE PROSIGN +
<..>'b' + RI + LN END

ACP_FMT_LN_3
= 'DEb' + RI + 'b' + ('#' +
ACP_SSN + [DATE_TIME]DATE +
'/'+ TIME + 'Z') +
LN_END

ACP_FMT_LN_15
= (CORRECTION) + (EOM_SEQ)

ACP_HEADER
= ACP_FMT_LN_1 + ACP_FMT_LN_2
+ ACP_FMT_LN_3 + (FMT_LN 4)
+ (FMT_LN-5) + (FMT_LN 6) +
(FMT_LN_7) + (FMT_LN_8) +
(FMT_LN_9) + (FMT_LN_10)

ACP_MESSAGE
= ACP_HEADER + SEPARATOR +
BODY + SEPARATOR +
ACP_TRAILER

ACP_PILOT
= ACP_FMT_LN_2 + FMT_LN 4 +
ICD + ICSN + <5>'b' +
([UU|HH]) + LN END

ACP_SSN
= <3>(DIGIT) + (LETTER)
* ORIGINATING STATION
SERIAL NUMBER *

ARRI
= <7>(['b']LETTER)

ACP_TRAILER
= (FMT_LN_14) +
ACP_FMT_LN_15
DATA DICTIONARY
FEBRUARY 8, 1982

ASN
   = TBD
   * ASSIGNED SER NO
   USED TO DIFFERENTIATE A
   PARTICULAR MESSAGE FROM ALL
   OTHER SYSTEM MESSAGES *

ASYNCHRONOUS_MODE
   = [ NORMAL ; STEPPED ]
   *TRANSMISSION METHOD*

ASYNCH.Receive_CONTROL_CHAR
   = [ACK 1; ACK 2; STOP; RT]
   * CHARACTER *

BELL_SIGNAL
   = FIGS + <5>('J') +
   <5>('S') + LTRS

BODY
   = <0..>(CHARACTER)
   * TEXT OF MESSAGE *

BP
   = CHARACTER
   * BLOCK PARITY OF MESSAGE
   FORMED BY THE BINARY
   ADDITION WITHOUT CARRY
   (XOR) OF THE BITS IN EACH
   ROW OF A BLOCK AND PLACED
   IN THE LAST FRAMING
   CHARACTER OF A
   SYNCHRONOUS MESSAGE. *

BP_CALC
   = * CUMULATIVE BLOCK PARITY
   CALCULATION DURING THE
   SYNCHRONOUS RECEIVE
   PROCESS STARTING WITH THE
   2ND FRAMING CHAR AND
   ENGING WITH "ETX" OR
   "ETB" *

CAN_CODEC
   = [TRUE ; FALSE ]

CANCEL_OUT_ENTRY
   = OCD + OCSN + DATE_TIME +
   + ASN
   + CHANNEL MODE
   + CANCEL_REASON
   + <1..50>(RI)
   + NUMBER_BLOCKS
   * MADE WHEN A CANCELLATION
   IS TRANSMITTED *

CANCEL_REASON
   = TBD

163
DATA DICTIONARY
FEBRUARY 8, 1982

CANCEL_REC_ENTRY = ICD + ICSN + DATE TIME + CHANNEL_MODE + NUMBER_BLOCKS + ASN
* MADE WHEN A CANCEL IS RECEIVED *

CANTRAN_CHAR_TO_SEND = [ TRUE ; FALSE ]

CANTRAN_REC_ENTRY = ICD + ICSN + DATE TIME + CHANNEL_MODE + NUMBER_BLOCKS + ASN
* MADE WHEN A CANCEL IS RECEIVED *

CANTRAN_OUT_ENTRY = OCD + OCSN + DATE TIME + CHANNEL_MODE + NUMBER_BLOCKS + ASN
* MADE WHEN A CANTRAN IS TRANSMITTED *

0xA4234 . CARD_CHECKED_MESSAGES
I1A4235
I1A4236

CARD_FMT_LN_2 = PRECEDENCE + LMF_PAIR + CLASS + CIC_CAI + 'b' + OSRI + OSSN + 'b' + DATE_TIME + 'b' + RECORD_COUNT + '-' + '-' + RI + <0..49>(RI) + 'b' +

0x1A4234 . CARD_INFO

CARD_MESSAGE = <28>({FROM_CARD|TO_CARD|NO})
* TABLE KEtED BY TRANSLATION_PAIR.
(CA;CT;CT' => FROM_CARD,
TC;QC;AC;FC => TO_CARD,
OTHERS => NO) *

01A4235 . CARD_TRANSLATED_MESSAGES
01A4236
IxA4237

CHANNEL_MODE = [1;2;3;4;5]
<table>
<thead>
<tr>
<th>Description</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR</td>
<td>= SEE CHARACTER</td>
</tr>
<tr>
<td>OxA4231 CHAR_CHECKED_MESSAGE</td>
<td>= MESSAGE + MCB + TRANSLATION_PAIR</td>
</tr>
<tr>
<td>I1A4232</td>
<td></td>
</tr>
<tr>
<td>I1A4233</td>
<td></td>
</tr>
<tr>
<td>01A4232 CHAR_TRANSLATED_MESSAGE</td>
<td>= MESSAGE + MCB + TRANSLATION_PAIR</td>
</tr>
<tr>
<td>01A4233</td>
<td></td>
</tr>
<tr>
<td>IxA4234</td>
<td></td>
</tr>
<tr>
<td>C1A4231 CHAR_SET_INFO</td>
<td>= &lt;28&gt;([ITA;ASCII;NO])</td>
</tr>
<tr>
<td></td>
<td>* TABLE KEYED BY TRANSFORMATION PAIR.</td>
</tr>
<tr>
<td></td>
<td>(CT</td>
</tr>
<tr>
<td></td>
<td>TC</td>
</tr>
<tr>
<td></td>
<td>OTHERS =&gt; NO) *</td>
</tr>
<tr>
<td>CHARACTER</td>
<td>= LEVEL + [TEXT_CHARACTER</td>
</tr>
<tr>
<td>CHARACTER_PARITY</td>
<td>= ASCII + [EVEN</td>
</tr>
<tr>
<td>CHNL_DES</td>
<td>= &lt;3&gt;(LETTER)</td>
</tr>
<tr>
<td></td>
<td>* CHANNEL DESIGNATOR *</td>
</tr>
<tr>
<td>CIC_CAI</td>
<td>= &lt;3&gt;(LETTER) + [LETTER</td>
</tr>
<tr>
<td></td>
<td>* CONTENT INDICATOR CODE -- CONTENT ACTION INDICATOR *</td>
</tr>
<tr>
<td>CLASS</td>
<td>= [M</td>
</tr>
<tr>
<td></td>
<td>* CLASSES ARE:</td>
</tr>
<tr>
<td></td>
<td>M -- DSSCS</td>
</tr>
<tr>
<td></td>
<td>A -- SPECAT</td>
</tr>
<tr>
<td></td>
<td>T -- TOP SECRET</td>
</tr>
<tr>
<td></td>
<td>S -- SECRET</td>
</tr>
<tr>
<td></td>
<td>C -- CONFIDENTIAL</td>
</tr>
<tr>
<td></td>
<td>R -- RESTRICTED</td>
</tr>
<tr>
<td></td>
<td>E -- EFTO</td>
</tr>
<tr>
<td></td>
<td>(ENCRYPT FOR TRANSMISSION ONLY)</td>
</tr>
<tr>
<td></td>
<td>U -- UNCLASSIFIED</td>
</tr>
<tr>
<td>CLASS_X5</td>
<td>= &lt;5&gt;(CLASS)</td>
</tr>
<tr>
<td>C1A411 COLLECTIVE_RI_TABLE</td>
<td>= &lt;0..200&gt;(RI +&lt;1..50&gt;(LINE_NO))</td>
</tr>
<tr>
<td>O1A411 COLLECTIVE_ROUTED_MESSAGES</td>
<td>= MESSAGE + MCB +&lt;1..50&gt;(OUTPUT_LINE_NUMBER)</td>
</tr>
<tr>
<td>I1A412</td>
<td></td>
</tr>
</tbody>
</table>
DATA DICTIONARY
FEBRUARY 8, 1982

COMMUNITIES_SERVED = (R) + (U) + (Y)

CONTROL_CHARACTER = [ASCII.NUL..ASCII.US!
ASCII.DEL|ITA.LTRS|
ITA.FIGS|ITA.CR|ITA.CR|
ITA.LF|ITA.BLANK]

CONTROL_CODE = [SEND_STOP|RT_RCVD|SEND_RT|
EOMS_RCVD|STOP_RCVD|
ACK_1_RCVD|ACK_2_RCVD|
CANCEL_RCVD|REPLY_RCVD|
SEND_RM|SEND_ACK|SEND_NAK|
WBT_RCVD|RM_RCVD|NAK_RCVD|
INVALID_RCVD|
INVALID_CHAR |
ENQUIRY_RCVD]

*USED TO SIGNAL CONDITIONS TO GENERATE_CONTROL
CHARACTERS -*

CORRECTION = 'C' + <..<>(CHARACTER) + LN_END

CURRENT_LIST = TBD

DATA_MODE = FORMAT + LEVEL

DATA_STATE = [TRUE|FALSE]

* TRUE, SYNCHRONOUS RECEIVER WILL ACCEPT TEXT CHARACTERS
FALSE, WILL ACCEPT CONTROL AND SYNC CHARACTERS *

DATE = <2>(DIGIT)

* DAY OF MONTH *

DATE_TIME = <7>DIGIT

* JULIAN DATE PLUS ZULU TIME *

DELS_CHECKED_MESSAGE = MESSAGE + MCB +
TRANSFORMATION_PAIR

DELS_INFO = <28>([YES|NO])

* TABLE KEYED BY TRANSFORMATION_PAIR
(AC|AT|AT' => YES,
OTHERS => NO ) *

166
DATA DICTIONARY
FEBRUARY 8, 1982

DIRECTION = (INPUT) + (OUTPUT)
* BOTH SET IMPLIES 2-WAY *

DOUBLE_PRECEDENCE_PROSIGN = <2>(PRECEDENCE)
* USED IN ACP_FMT_LN_2 *

ECSN = <3>(DIGIT)
* EXPECTED CHANNEL SEQUENCE NUMBER *

EOM_IN_ENTRY
= ICD + ICSN + DATE TIME +
  ASN + NUMBER_BLOCKS +
  CHANNEL MODE
* MADE WHEN EOM IS RECEIVED *

EOM_OUT_ENTRY
= OCD + OCSN + DATE TIME +
  ASN + <1..50>(RI) +
  NUMBER_BLOCKS
* MADE WHEN EOM IS TRANSMITTED *

EOM_SEQ
= <2>(CR) + <8>(LF) +
  <4>('N') + <8>(LTRS)

EOM_VALIDATION
= ' #' + <4>(DIGIT)

EOT_CARD
= PRECEDENCE + LMF PAIR +
  CLASS + CIC_CAI + 'b' + OSRI +
  OSSN + 'b' + DATE TIME +
  'b' + RECORD_COUNT + '-' +
  REDUNDANT_CLASS +
  <44>('b') + <4>('N')
* N'S START IN COLUMN 72 *

ERROR_CODE
= [ NO ERROR ;
  INVALID_LENGTH ;
  INVALID_LINE RI ;
  INVALID_LMF ;
  INVALID_RI_SECURITY ;
  BAD BLOCK COUNT ;
  INVALID_LINE_TRANSMISSION_TYPE ]
* CODES TO INDICATE ERROR CONDITION *

0xA4237 FILL_CHECKED_MESSAGE
I1A4238 = MESSAGE + MCB +
  TRANSLATION_PAIR

167
DATA DICTIONARY
FEBRUARY 8, 1982

C1A4237 FILL_INFO = <28>((YES; NO))
* TABLE KEYED BY
TRANSLATION PAIR
(CA|CT|CT'|AT|AT'|QA|FA =>
YES, OTHERS => NO) *

FIRST_LMF_CHARACTER = [B|D|I|S;C|A|T|R;Q|F]

FMT_LN_4 = OP_SIGNAL + CLASS_X5 +
<0..>(CHARACTER) + LN_END

FMT_LN_5 = PRECEDENCE +
(|'b' + PRECEDENCE)
+ 'b' + DATE + TIME + 'b' +
MONTH + 'b' + YEAR
+ <0..>(CHARACTER) + LN_END

FMT_LN_6 = 'FMb' + <..>(CHARACTER) +
LN_END

FMT_LN_7 = 'TOb' + <..>(CHARACTER) +
LN_END

FMT_LN_8 = 'INFOb' + <..>(CHARACTER) +
LN_END

FMT_LN_9 = 'XMTb' + <..>(CHARACTER) +
LN_END

FMT_LN_10 = <..>(CHARACTER) + LN_END

FMT_LN_14 = <..>(CHARACTER) + LN_END

FORMAT = [JANAP_128 ; ACP_127]
* MESSAGE FORMAT-*

0xA4221 FORMAT_CHECKED_MESSAGE = MESSAGE + MCB +
TRANSLATION_PAIR

IA14222
I1A4223

01A4222 FORMAT_CONVERTED_MESSAGE = MESSAGE + MCB +
TRANSLATION_PAIR

01A4223
IxA4224

C1A4221 FORMAT_INFO = <28>((TO_127; TO_128; NO))
* TABLE KEYED BY
TRANSLATION PAIR.
(CT|AT'|TT|RT|QT =>
TO_127, FC|FA|FT =>
TO_128, OTHERS => NO) *
DATA DICTIONARY
FEBRUARY 8, 1982

HEADER = [JANAP_HEADER; CARD_FMT_LN_2; SINGLE_CARD; ACP_HEADER]

ICD = <3>(CHARACTER)
* INCOMING CHANNEL ID *

ICSN = <3>(DIGIT)
* INCOMING CHANNEL SER NO *

INCOMING_LOG_ENTRY = [SOM_IN_ENTRY; EOM_IN_ENTRY; REJECT_ENTRY; CANCEL_REC_ENTRY; CANTRAN_REC_ENTRY]

O1A412 INDIVIDUAL_ROUTED_MESSAGES = MESSAGE + MCB +
<1..50>(OUTPUT_LINE_NUMBER)

I1A413 INPUT_STREAM = <..>(BIT)

I3A3 INCEPT_IN_ENTRY = HEADER + ASN + DATE_TIME +
MCB + NUMBER_BLOCKS
* MADE WHEN MESSAGE RECEIVED FROM INTERCEPT *

O1A1331 INTERCEPT_LIST = TBD

I1A1332 INTERCEPT_LOG_ENTRY = [INTERCEPT_IN_ENTRY;
INTERCEPT_OUT_ENTRY]

O2A43 INTERCEPT_OUT_ENTRY = ASN + HEADER + DATE_TIME +
<1..50>(RI) + NUMBER_BLOCKS
* MADE WHEN MESSAGE SENT TO INTERCEPT *

INVALID_CHAR = ASCII.BEL(16#07#)

JANAP_FMT_LN_1 = 'VZCZC' + ICD + FIGS
+ ICSN + LTRS + LN_END
* TRANSMISSION IDENT (TI) *

JANAP_FMT_LN_2 = PRECEDENCE + LMF_PAIR +
CLASS + CIC_CAI + 'b' + OSRI
+ OSSN + 'b' + DATE_TIME +
'- ' + REDUNDANT_CLASS +
'- ' + RI + <0..49>(RI) +
'.' + LN_END
DATA DICTIONARY
FEBRUARY 8, 1982

JANAP_FMT_LN_3 = 'DEb' + RI + OSSN + DATE_TIME + LN_END

JANAP_FMT_LN_15 = (CORRECTION) + (EOM_VALIDATION)

JANAP_FMT_LN_16 = EOM_SEQ

JANAP_HEADER = (JANAP_FMT_LN_1) + (JANAP_PILOT) + (JANAP_FMT_LN_3) + (JANAP_FMT_LN_3) + (FMT_LN_4) + (FMT_LN_5) + (FMT_LN_6) + (FMT_LN_7) + (FMT_LN_8) + (FMT_LN_9) + (FMT_LN_10)

JANAP_MESSAGE = [CARD_MESSAGE;TTY_MESSAGE]

JANAP_PILOT = PRECEDENCE + LMF_PAIR + CLASS + CIC_CAI + 'b' + OSRI + OSSN + 't' + DATE_TIME + '-' + REDUNDANT_CLASS + RECORD_COUNT + '---' + RI + <0..49>(RI) + '...' + LN_END
* RECORD_COUNT MUST BE RI *

JANAP_TRAILER = (FMT_LN_14) + (JANAP_FMT_LN_15) + JANAP_FMT_LN_16

LEVEL = [5;8]
* 5 LEVEL = ITA #2
8 LEVEL = ASCII *

LINE_AVAILABLE = TBD
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1A42</td>
<td>LINE_DATA</td>
<td>CHANNEL MODE + [SYNCHRONOUS MODE; ASYNCHRONOUS MODE] + DATA MODE + LOGICAL LINE NO + PHYSICAL LINE NO + LOOP SPEED + COMMUNITIES SERVED + (FIRST_LINK) + MAX NO RIS PER DELIV + (SOM_SEQ) + DIRECTION + ECSN + OCSN + CHNL_DES + (RCSN) + (NO_STOP_BITS) + (SECURITY_PROSIGN) + (SPEC_TERM) + LINE_AVAILABLE</td>
</tr>
<tr>
<td>C1A43</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C1A421</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C1A431</td>
<td></td>
<td></td>
</tr>
<tr>
<td>O1A421</td>
<td>LMF_MESSAGES</td>
<td>MESSAGE + MCB + TRANSLATION_PAIR</td>
</tr>
<tr>
<td>I1A422</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I1A4221</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LMF_PAIR</td>
<td></td>
<td>FIRST LMF_CHARACTER + SECOND LMF_CHARACTER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* LEGAL PAIRS ARE: TT TA TC AT AA AC CC CA CT RT SC BB CC II QT FT *</td>
</tr>
<tr>
<td>LN_END</td>
<td></td>
<td>&lt;2&gt;(CR) + LF</td>
</tr>
<tr>
<td>I2A1</td>
<td>LOG_DATA</td>
<td>&lt;0..&gt;(LOG_ENTRY)</td>
</tr>
<tr>
<td>I2A2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I2A12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I2A13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I2A121</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I2A132</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I2A133</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I1A1321</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I1A1331</td>
<td></td>
<td></td>
</tr>
<tr>
<td>O3A3</td>
<td>LOG_ENTRY</td>
<td>[INCOMING LOG_ENTRY; OUTGOING_LOG_ENTRY; OVERFLOW_LOG_ENTRY; INTERCEPT_LOG_ENTRY; SVC_GEN_ENTRY]</td>
</tr>
<tr>
<td>O3A4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>O3A5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOG_ENTRY_INFO</td>
<td></td>
<td>INCOMING_LOG_ENTRY * EXCEPT_CANTRAN *</td>
</tr>
<tr>
<td>MAX_NO_RIS_PER_DELIV</td>
<td></td>
<td>[1;6;14;50;500]</td>
</tr>
</tbody>
</table>
DATA DICTIONARY
FEBRUARY 8, 1982

MCB = 'SOH' + SELECT_CHARACTER + PRECEDENCE + LMFPAIR + CLASS + CIC_CAI + 'b' + OSRI + OSSN + 'b' + DATE_TIME + NUMBER_BLOCKS + SOH_TIME + TRIP + SDC + TDW + SDO + TASC + SASC + ARRI + SMRI + ICD + TMID + ICSN + 'ETB' + BP

MESSAGE = [JANAP_MESSAGE ; ACP_MESSAGE]

MESSAGE_CHAR_TO_SEND = [TRUE ; FALSE ]
* CHARACTER AVAILABILITY

MESSAGE_DATA = MESSAGE

MESSAGE_DATA_A = MESSAGE
* NOTE: ON MESSAGE_DATA_A THROUGH MESSAGE_DATA_D, LEXICALLY GREATER DISTINGUISHING LETTERS INDICATE PROGRESSIVELY MORE COMPLETE VALIDATION *

MESSAGE_DATA_B = MESSAGE

MESSAGE_DATA_C = MESSAGE

MESSAGE_DATA_D = MESSAGE
DATA DICTIONARY
FEBRUARY 8, 1982

MESSAGE_DATA_C = MESSAGE

MESSAGE_DATA_D = MESSAGE

MESSAGE_CANCELLED = [ TRUE ; FALSE ]

MESSAGE_IN_TRANSMISSION = [ TRUE ; FALSE ]

MESSAGE_MODE = [ JANAP 128 ; ACP_127 ; ACP_127 MOD ]

MESSAGE_REJECTION = SEND RM

* USED TO SIGNAL
MESSAGE REJECTION

TO GENERATE CONTROL
CHARACTER FUNCTION *

MONTH = [JAN:FEB ... DEC]

* 3 LETTER MONTH NAMES *
DATA DICTIONARY
FEBRUARY 8, 1982

O3A1 NEW_INTERCEPT_FILE = <O..>(MESSAGE + MCB)
O2A13
O1A133
O1A1332
I1A4

O4A1 NEW_OVERFLOW_FILE = <O..>(MESSAGE + MCB)
O2A13
O1A132
O1A1322
I2A3

NEW_ROUTE_LINE = ROUTE_LINE

NO_ERROR

NUMBER_BLOCKS = <3>(DIGIT)

OASC = ['b'!LETTER]

OCCUPANCY_LEVEL = [0..'00]

OCD = <3>(CHARACTER)

OCSN = <3>(DIGIT)

OP_SIGNAL = ['ZNY';'ZNR']

01A2 OPERATOR_COMMAND = [REINTRODUCE_COMMAND;
TRACE_COMMAND;
RETRIEVE_COMMAND;
RECOVERY_COMMAND;
THROTTLE_COMMAND;
DRY_UP_COMMAND;
CHANNEL_COMMAND;
TAPE_COMMAND;
STATISTICS_COMMAND;
MESSAGE_CONTROL_COMMAND;
DESTRUCT_COMMAND;
ROUTING_COMMAND;
EQUIPMENT_COMMAND;
TABLE_COMMAND]

01A2 OPERATOR_PRINTOUTS = TBD
DATA DICTIONARY
FEBRUARY 8, 1982

OSRI = RI
* ORIGINATING STATION ROUTING INDICATOR *

OSSN = <4>(DIGIT)
* ORIGINATING STATION SERIAL NUMBER *

OUTGOING_LOG_ENTRY = [SOM_OUT_ENTRY;
EOM_OUT_ENTRY;
CANCEL_OUT_ENTRY;
SCRUB_ENTRY;
CANTRAN_OUT_ENTRY]

OUTPUT_LINE_TYPE = [ TRIBUTARY | TRUNK ]

OUTPUT_LINE_NUMBER = [1..50]

OUTPUT_SELECTED_MESSAGE = MESSAGE + MCB

OUTPUT_STREAM = <..>(BIT)

OVERFLOW_IN_ENTRY = HEADER + ASN + DATE_TIME +
MCB + NUMBER_BLOCKS
* MADE WHEN MESSAGE RECEIVED FROM OVERFLOW *

OVERFLOW_LIST = TBD

OVERFLOW_LOG_ENTRY = [OVERFLOW_IN_ENTRY;
OVERFLOW_OUT_ENTRY]

OVERFLOW_OUT_ENTRY = ASN + HEADER + DATE_TIME +
NUMBER_BLOCKS
* MADE WHEN MESSAGE SENT TO OVERFLOW *

PAUSE_GENERATED = [ TRUE | FALSE ]

175
DATA DICTIONARY
FEBRUARY 8, 1982

PRECEDENCE
= [W;Y;Z;O;P;R]
  * PRECEDENCES ARE:
  W -- CRITIC
  Y -- ECP
  Z -- FLASH
  O -- IMMEDIATE
  P -- PRIORITY
  R -- ROUTINE *

PREPARE_TO_SEND_MESSAGE
= [ TRUE ; FALSE ]
  * INDICATES COMPLETION
  OF MESSAGE *

I3A1 PROGRAM_LOAD_FILE
= TBD

O4A2 PROGRAM_LOAD_INSTRUCTIONS
= TBD

IxA5426 RCC_TO_SEND
= [ TRUE ; FALSE ]

RCSN
= <3>DIGIT
  * REJECTED CHANNEL SEQUENCE
  NUMBER *

RECORD_COUNT
= [<4>(DIGIT);'MTMS';'PLTS';
RI]
  * RI--MS RI (4 LETTERS) *

O3A1 RECOVERED_MESSAGES
= <0..>(MESSAGE + MCB)

O1A12

O1A121

I1A3

O2A11 RECOVERY_COMMAND
= TBD

C1A12

C1A121

REDUNDANT_CLASS
= <4>(CLASS)

O4A3 REFERENCE_COPY
= MESSAGE + MCB

I1A1 REFERENCE_DATA
= <0..>(MESSAGE + MCB)
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>01A422</td>
<td>Reformatted_Message</td>
<td>Message + MCB + Translation_Pair</td>
</tr>
<tr>
<td>01A4225</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I1A423</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I1A4231</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C2A43</td>
<td>Reintroduce_Command</td>
<td>TBD</td>
</tr>
<tr>
<td>C2A432</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reject_Entry</td>
<td>Date Time + Reject_Reason + ASN + Channel_Mode + ICD + ICSN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* Made When A Message Is Rejected By The Switch *</td>
</tr>
<tr>
<td></td>
<td>Reject_Reason</td>
<td>TBD</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reply</td>
<td>Protocol Command</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* Sent By The Transmitting Station To Direct The Receiver To Send Its Current Status *</td>
</tr>
<tr>
<td></td>
<td>Reply_Character</td>
<td>ASCII. DC1(16#11#)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Request_for_answer_char</td>
<td>[Transmit_Control_Char ; BP_Char ]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RI</td>
<td>[R</td>
</tr>
<tr>
<td>C1A412</td>
<td>RI_Line_Table</td>
<td>&lt;0..1500&gt;(RI+LINE_NO)</td>
</tr>
<tr>
<td>C1A41</td>
<td>Route_Data</td>
<td>TBD</td>
</tr>
<tr>
<td>C1A332</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C1A333</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C1A334</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C1A335</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C1A3323</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C1A3332</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C1A3344</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C1A3352</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Route_Line</td>
<td>[JANAP_FMT_LN_2 ; ACP_FMT_LN_2 ; CARD_FMT_LN_2]</td>
</tr>
<tr>
<td>C2A4</td>
<td>Route_Line_Data</td>
<td>TBD</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
DATA DICTIONARY
FEBRUARY 8, 1982

O1A41 ROUTED_MESSAGE = MESSAGE + MCB
O1A413
I1A42
I1A421

O1A331 ROUTED_MESSAGE_DATA = MESSAGE
O2A331
O3A331
O4A331
I1A332
I1A333
I1A334
I1A335
I1A3321
I1A3331
I1A3341
I1A3351

SASC = ['b'|LETTER]
IxA5426 SEND_DET_CODE = [ TRUE ; FALSE ]
IxA5426 SEND_DET_GENERATED = [ TRUE ; FALSE ]

SCRUB_ENTRY = OCD + OCSN + CHANNEL_MODE + DATE_TIME + ASN + SCRUB_REASON + NUMBER_BLOCKS
<1..50>(RI)
* MADE WHEN A MESSAGE IS SCRUBBED *

SEND_GEN_CODE = [ TRUE ; FALSE ]
SCRUB_REASON = TBD
SDC = ['b'|S|C|P|M|T|A]
SDO = ['b'|LETTER]
SECOND_LMF_CHARACTER = [B|D|I|C|A|T]
SECURITY_PROSIGN = [A|T|S|C|R|E|U|M]
SELECT_CHARACTER = [A|B|C|D|E|F|G|H|J|K|M|P|S]
SEPARATOR = 'BT' + LN_END
DATA DICTIONARY
FEBRUARY 8, 1982

01A4 SERVICE_MESSAGE_INFO = SVC MESSAGE_TYPE + MESSAGE + \(0^{.50}\)(RI) + \(<2\)\)(ICSN)

01A42
02A422
02A4223
01A5
01A332
01A333
01A334
01A335
01A3321
02A3322
02A3323
01A3324
01A3325
01A3331
01A3332
01A3333
01A3334
01A3342
02A3343
02A3344
01A3345
01A3346
01A3351
01A3352
01A3353
01A3354
C2A3

SINGLE_CARD = PRECEDENCE + LMF_PAIR + CLASS + CIC_CAI + 'b' + OSRI + OSSN + 'b' + DATE_TIME + '-' + RI + '. ' + \(<41\)\)(CHARACTER) + 'N'
* FINAL N MUST BE IN COLUMN 80 *

SMRI = \(<5\)\)(LETTER)

SOH_TIME = DATE_TIME
* TIME OF RECEPTION OF START OF HEADER *

SOM_IN_ENTRY = CHANNEL_MODE + HEADER + DATE_TIME + ASN + ICD + ICSN + MCB
* MADE WHEN START OF MESSAGE RECEIVED *

179
DATA DICTIONARY
FEBRUARY 8, 1982

SOM_OUT_ENTRY = OCD + OCSN + CHANNEL_MODE + <1..50>(RI) + HEADER + DATE_TIME + ASN
* MADE WHEN START OF MESSAGE TRANSMITTED *

SOM_SEQ = [ FULL ; ABBREVIATED ]

SORTED_MESSAGE = MESSAGE + MCB

SPEC_TERM = [ ACCESS | INTERSWITCH | TECH_CONTROL ]

START_COMMAND = TBD

STOP_RECD_CODE = [ TRUE | FALSE ]

SVC_GEN_ENTRY = HEADER + ASN + DATE_TIME + MCB + NUMBER BLOCKS
* MADE WHEN SERVICE MESSAGE IS GENERATED *

SVC_MESSAGE_TYPE = [ INVALID RI;
OUTPUT SCTY MISMATCH;
EXCESSIVE ROUTING REJ;
ILLEGAL EXCHANGE;
SUSPENDED TRANSMISSION;
SUSPECTED STRAGGLER;
OPEN CSN;
INPUT SCTY MISMATCH;
INVALID SCTY FIELD;
INVALID HEADER REJ;
INVALID HEADER ACC;
HI PREC ACC;
INVALID RI FIELD;
ALL RI INVALID;
INVALID TI REJ;
INVALID TI ACC;
TWO CONSEC.SOM;
INVALID EOM REJ;
INVALID EOM ACC;
INCORRECT CSN;
INVALID BLOCK_COUNT;
TRAFFIC_CHECK;
NO_EOM ]
SYNCH_RECEIVE_CONTROL_CHAR = [ACK 1 | ACK 2 | NAK | RM | WBT] * CHARACTER *
SYNCHRONOUS_MODE = [BLOCK_BY_BLOCK | CONTINUOUS]

O3A2 TABLE_CHANGES = TBD

O1A1
O1A11

TASC = ['b';LETTER]

IxA5426 TCC_TO_SEND = [ TRUE ; FALSE ]

TDW = <12>(CHARACTER)

TEXT_CHARACTER = [ASCII.BLANK..ASCII.TILDE ; * PRINTABLE ITA CHARACTERS]

THRESHOLD_STATUS = [UPPER;UPPER MIDDLE; LOWER MIDDLE;LOW]

TIME = <4>(DIGIT) + 'Z'

T MID = <8>(CHARACTER)

IxA5426 TIMING_REPLY_TO_CAN = [TRUE ; FALSE ]

IxA5426 TIMING_REP_OR_STOP = [ TRUE ; FALSE ]

O1A42
O1A423
O1A4238
O1A431
O2A431
O2A432
I1A43
I1A431
I1A432
I1A433

TRANSLATION_PAIR = LMF_PAIR

TRANSMIT_CONTROL_CHAR = [REPLY_CHAR;CANCEL_CHAR; ENQUIRY_CHAR]

TRIP = [N;C;O;P;R;D]
TTY MESSAGE = JANAP_HEADER + SEPARATOR + BODY + SEPARATOR + JANAP_TRAILER

* NOTE: THE MAXIMUM LENGTH FOR A PHYSICAL LINE IN A TTY MESSAGE IS 69 PRINTABLE CHARACTERS. SINCE A FORMAT LINE MAY EXCEED THIS NUMBER, LN END'S MUST BE ADDED AS REQUIRED TO KEEP EACH PHYSICAL LINE AT OR BELOW THIS LENGTH *

03A332  VALIDATED_MESSAGE = MESSAGE

03A333
03A334
03A335
03A3324
03A3325
03A3334
03A3345
03A3346
03A3354

YEAR = <2>(DIGIT)
Message Switch Non-Functional Requirements

I. Messages shall be accepted without regard to the immediate availability of an outgoing line or trunk. The receive module will have the ability of throttling incoming traffic during overload periods or during periods of high priority messages. Two types of message codes will be handled.

A. ASCII Code

(1) Odd parity
(2) Seven (7) bits plus parity
(3) One (1) start unit interval
(4) One (1) or two (2) stop unit interval(s)

B. ITA #2 Code

(1) 5 bits data
(2) One (1) start unit interval
(3) One (1) or two (2) stop unit interval(s)

II. Synchronous Operation - Messages Received From an Autodin Switch

A. Block-by-Block

(1) 75-16000 b/s
(2) ASCII character set
(3) Parity Criteria - (7 bits for information and the 8th bit for parity)
(4) Bits received serially - (low order bit first and the parity bit last)
(5) Message characters will have odd parity
(6) Control characters will have even parity
(7) Modes I or III
(8) Acquire frame sync. before receive
(9) A message control block (MCB) will be employed if the message is via an interswitch trunk
(10) Bit stream - bit synchronous where each bit is accompanied with a clock pulse
(11) Recognize accepted idle pattern for channel sync.
(12) SOH/STX characters of a block stream will be separated by a syn. period from the previous block.
b. Continuous (block-by-block)

(1) SOH/STX characters of a block stream will be contiguous to the block parity character of the previous block.

(2) Reception of continuous (block-by-block) is the same as block-by-block above except for item 12.

III. Asynchronous Operations Modes II, IV (without Automatic Channel and Error Control)

A. ASCII and ITA#2 Codes (Mode II)

(1) Free running characters

(2) Character by character release

(3) Independent of simultaneous two-way operation

B. ASCII and ITA#2 Code (Mode IV)

(1) Unidirectional operation (send only or receive only)

(2) Equivalent to half duplex operation of Mode II

(3) Free running characters

IV. Asynchronous Operation Mode V (with automatic channel and error control)

A. ASCII Code

(1) 75-300 baud

(2) Continuous two-way transmission

B. ITA #2 Code

(1) 45.45 to 75 baud

(2) Continuous two-way transmission

V. Message statistics. The maximum length message is 44,000 characters (or 550 blocks). The maximum length teletype (TTY) message is 6900 characters. The average message length is 2400 characters. There will be an average of 1.75 addresses per message. The average collective RI received will have 3.9 destinations. The messages will be distributed among the various precedences as follows:

- ECP(Y)/CRITIC(W) 0.1%
- FLASH(Z) 2.9%
- IMMEDIATE(O) 30.0%
- PRIORITY(P) 33.0%
- ROUTINE(R) 34.0%
VI. Maximum character handling - characters per time period

<table>
<thead>
<tr>
<th></th>
<th>25 line switch</th>
<th>50 line switch</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 second</td>
<td>1 second</td>
</tr>
<tr>
<td>Input</td>
<td>3600</td>
<td>5400</td>
</tr>
<tr>
<td>Output</td>
<td>2400</td>
<td>3600</td>
</tr>
</tbody>
</table>

VII. Errors. Bit errors within the switch or delivered to output terminals shall not exceed 1 bit in 10 billion consecutive bits. No more than 1 message in 10 million messages shall be misrouted. No more than 1 in 10 million messages shall be lost due to nondetection or errors in the following: SOM, EOM, cancel transmission (CANTTRAN), or header. Loss of receipted messages shall not exceed 1 in 1 billion.

VIII. Processing time. Processing time will be the sum of the time for reception of the EOM until the message is placed on the output queue and the time from the output line becoming available until the first bit of the SOM is transmitted. Mean processing time will be 2 seconds per message. No more than 1 in 1,000 CRITIC, ECP, or FLASH messages will have in processing time greater than 6 seconds. No more than 1 in 1,000 messages of lower precedence will have a processing time in excess of 8 seconds.

IX. Routing table size. The routing table shall contain room for:

- 1500 single address RIs
- 200 collective RIs with an average 20 RIs each
- 600 RIs for other centers

X. Intransit storage size. For a 25 line switch, intransit storage shall be capable of holding a minimum of 2000 average length messages. For 50 lines this number shall be increased to 2500.

XI. Recovery and retrieval. Reentry of messages from intercept storage shall be such that the first message is ready for reentry with 10 minutes of the reentry directive. Retrieval of CRITIC, ECP, and FLASH messages less than 24 hours old shall be accomplished in 7 minutes or less. All other messages less than 24 hours old shall be retrieved within 15 minutes. Messages older than 24 hours must be capable of being retrieved within 30 minutes.
Recovery of messages after switch failure must take no more than 30 minutes.