SOFTWARE DESCRIPTION FOR THE O'HARE RUNWAY CONFIGURATION MANAGEMENT SYSTEM

VOLUME II: LOW-LEVEL PSEUDOCODE

SADEGH KAVOUSSI

The MITRE Corporation
McLean, Virginia 22102

OCTOBER 1982

Document is available to the U.S. public through the National Technical Information Service
Springfield, Virginia 22161

Prepared for
U.S. DEPARTMENT OF TRANSPORTATION
FEDERAL AVIATION ADMINISTRATION
OFFICE OF SYSTEMS ENGINEERING MANAGEMENT
Washington, D.C. 20591
NOTICE

This document is disseminated under the sponsorship of the Department of Transportation in the interest of information exchange. The United States Government assumes no liability for its contents or use thereof.
### Abstract

This document describes the software developed as part of the Chicago O'Hare Runway Configuration Management System (CMS). The software is designed as an interactive automated planning aid to assist the O'Hare assistant chief in the consistent selection of efficient runway configurations in order to lower aircraft delays. In addition, CMS serves as an information management system by consolidating various airport data and making them available for the O'Hare facility personnel. Volume I of this document contains the general description of the CMS software plus high level pseudocode describing its logic. Volume II is dedicated to detailed description of the software via low level pseudocode.
ACKNOWLEDGEMENT

The author is indebted to Lucille Perrin for the many hours spent in preparing this document.
EXECUTIVE SUMMARY

The O'Hare Runway Configuration Management System (CMS) is an interactive multi-user computer system designed to aid O'Hare management personnel in the consistent selection of runway configurations in order to reduce aircraft delays. CMS is also used for the purpose of communicating and disseminating information about the airport among the tower and Terminal Radar Control Facility (TRACON) personnel.

Although the CMS software was written for O'Hare International Airport, it can be adapted for other airports to serve as an automated planning aid for runway configuration management. This would require changing the associated site specific adaptation data. At some airports, however, the need might be to manage the surrounding airspace which is shared with other airports, or to manage the flow of aircraft on taxiways as opposed to runway configuration management. The basic concepts of CMS can be extended to include such applications as well but would require site specific model development to suit the needs of the individual airport.

The purpose of this report is to describe the CMS software in the time sharing environment of MITRE Washington's Computer Center. Currently, CMS is housed in an IBM 4341 computer with VM/SP operating system. CMS employs the IBM's Display Management System (DMS) software package that provides full screen menu type displays. The display terminals used by CMS are IBM's 3270 series or equivalent. The CMS software is written exclusively in PL/I and complies fully with top-down structured programming techniques.

CMS has been designed to facilitate manual data entry, since automated inputs are not yet readily available. CMS is available for interactive access by the tower and radar room personnel who normally monitor and report changes in the airport operational environment. These users are: the Assistant Chief (AC), who has the primary responsibility for configuration selection; the team supervisor of the tower cab (CAB), who provides operational information (wind, weather, runway conditions) to the system; and the Airways Facilities operations officer (AF), who is responsible for the runway equipment status. The interactions between these users and CMS are illustrated in Figure A.

Because of the limitations of the time-sharing system under which CMS is currently operating, these three different users can only be supported by three separate programs. These programs are compiled and stored separately and operate independently, but communicate through a common database which contains all information on O'Hare status over the planning horizon. When CMS is implemented at
Figure A
PHYSICAL CONFIGURATION OF CMS
O'Hare, it will operate on a dedicated mini-computer which permits multi-tasking (that is, multiple users interacting with a single program simultaneously). This will eliminate the need for three separate programs; certain changes to the program structure will be required to make best use of the multi-tasking environment, but the basic CMS logic will not be affected.

Each program within the CMS software package supports a set of data "screens", each containing a predetermined subset of information for input or display. An example of a CMS screen is given in Figure B. Table A contains a list of display screens within the CMS software. In some cases there is an overlap of information among several screens. Although the screens are not mutually independent (i.e., changes in one screen may affect the contents of the others), they are self-contained in that they serve a specific purpose and are acted upon separately.

The screens provide a convenient format for entering data on the current and future operating environment at O'Hare. This includes information on wind speed and direction, ceiling and visibility, runway surface conditions, status of runway landing aids, and the expected volume and distribution of traffic. This information is then used by CMS to determine the operational availability of individual runways. The operational suitability of the runway configurations is then determined, based upon runway availability and other operational factors; and the configurations are ranked according to their capacities, based upon projected demand for the next hour. The penalty of transitioning from current configuration, in terms of capacity during the transition period, is also calculated and displayed. This yields the primary output of the runway configurations management system -- an ordered list of transition strategies indicating which runways to use at what times during the planning period.

Volume I of this report defines the major subsystems within the CMS software package, discusses the overall control and architecture of the CMS software, and describes the software logic pertaining to each component. "High-level", English-like pseudocode is used to describe the CMS software. Pseudocode is used because it can provide a clear, English-like description which is believed superior to flowcharts for conveying complex logic to the reader, while still maintaining a formal structure.

Volume II contains the "low-level", variable specification pseudocode, in order to provide a detailed description of the software.
**RUNWAY EQUIPMENT PLANNING LOG**

<table>
<thead>
<tr>
<th>RWY</th>
<th>EQUIPMENT</th>
<th>OTS</th>
<th>RTS</th>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>4L</td>
<td>ALS</td>
<td>1500</td>
<td>1600</td>
<td>REPAIRS</td>
</tr>
</tbody>
</table>

**ASST. CHIEF—ADDITIONAL ENTRIES**

**DATA STORED AT 1447**

**FIGURE B**

**EXAMPLE OF CMS SCREEN**
TABLE A
LIST OF INPUT/OUTPUT DISPLAY SCREENS

1. Menu of Program Function Keys and Program Termination
2. Parameters
3. O'Hare Status Summary
4. Planning Log Selection
5. Wind and Weather Planning Log
6. Runway Conditions Planning Log
7. Equipment Planning Log
8. Demand Planning Log
9-10. Airport Status (Current/Forecast)
11-12. Runway Equipment Status (Current/Forecast)
13-14. Demand Profile (Current/Forecast)
15-16. Ordered List of Configurations (Current/Forecast)
17. Current Departure Queues
18. Ordered List of Transitions
19-20. Configuration Information (Current/Forecast)
TABLE OF CONTENTS

1. INTRODUCTION

2. CMS SOFTWARE LOGIC (LOW LEVEL PSEUDOCODES)

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 System Data Structures</td>
<td>2-1</td>
</tr>
<tr>
<td>2.2 High Level Processing</td>
<td>2-59</td>
</tr>
<tr>
<td>2.3 O'Hare Status Summary Screen</td>
<td>2-149</td>
</tr>
<tr>
<td>2.4 Planning Log Selection Screen</td>
<td>2-177</td>
</tr>
<tr>
<td>2.5 Weather and Wind Planning Log Screen</td>
<td>2-186</td>
</tr>
<tr>
<td>2.6 Airport Runway Surface Planning Log Screen</td>
<td>2-207</td>
</tr>
<tr>
<td>2.7 Equipment Planning Log Screen</td>
<td>2-227</td>
</tr>
<tr>
<td>2.8 Demand Planning Log Screen</td>
<td>2-245</td>
</tr>
<tr>
<td>2.9 Airport Status Screen</td>
<td>2-264</td>
</tr>
<tr>
<td>2.10 Runway Equipment Status Screen</td>
<td>2-279</td>
</tr>
<tr>
<td>2.11 Demand Profile Screen</td>
<td>2-290</td>
</tr>
<tr>
<td>2.12 Ordered List of Configurations Screen</td>
<td>2-310</td>
</tr>
<tr>
<td>2.13 Departure Queue Screen</td>
<td>2-332</td>
</tr>
<tr>
<td>2.14 Ordered List of Transitions Screen</td>
<td>2-340</td>
</tr>
<tr>
<td>2.15 Configuration Information Screen</td>
<td>2-397</td>
</tr>
<tr>
<td>2.16 Menu and Parameter Screens</td>
<td>2-421</td>
</tr>
</tbody>
</table>
1. INTRODUCTION

The O'Hare Runway Configuration Management System (CMS) is an interactive multi-user computer system designed to aid the O'Hare management personnel in the consistent selection of runway configurations in order to reduce aircraft delays. CMS is also used for the purpose of communicating and disseminating information about the airport among the various tower and Terminal Radar Control Facility (TRACON) personnel.

This set of reports in two volumes describes the software description of the existing CMS at the MITRE Washington's Computer Center, where it resides in a time share mode on an IBM 4341. Volume I contains the technical description and the high level pseudocode.

This document, Volume II, contains the detailed description of CMS software in the form of low-level pseudocode.
This appendix is dedicated to detailed description of the CMS software logic via the use of low level pseudocodes. There are sixteen separate modules given here, as follows:

2.1. System data structures  
2.2. High level processing  
2.3. O'Hare status summary screen  
2.4. Planning log selection screen  
2.5. Weather and wind planning log screen  
2.6. Airport runway surface planning log screen  
2.7. Equipment planning log screen  
2.8. Demand planning log screen  
2.9. Airport status screen  
2.10. Runway equipment status screen  
2.11. Demand profile screen  
2.12. Ordered list of configurations screen  
2.13. Departure queue screen  
2.14. Ordered list of transitions screen  
2.15. Configuration information screen  
2.16. Menu and parameter screens

The high level pseudocodes and a cross-reference table for both low and high level pseudocodes are located in Appendix A of Volume I.

2.1 System Data Structures

System Data Structures are described on pages 2-2 to 2-58.
[VARIABLES PERTAINING TO PROGRAM FUNCTION KEYS]

BITS PF1 [8 bit variable set to '11110001'B, invokes O'Hare status summary screen]

BITS PF2 [8 bit variable set to '11110010'B, invokes planning log selection screen]

BITS PF3 [8 bit variable set to '11110011'B, invokes current and forecast airport status screens]

BITS PF4 [8 bit variable set to '11110100'B, invokes current and forecast airport status screens]

BITS PF5 [8 bit variable set to '11110101'B, invokes current and forecast demand profile screens]

BITS PF6 [8 bit variable set to '11110110'B, invokes current and forecast ordered list of configurations screens]

BITS PF7 [8 bit variable set to '11110111'B, invokes current departure queue screen]

BITS PF8 [8 bit variable set to '11111000'B, invokes ordered list of transition screen]

BITS PF9 [8 bit variable set to '11111001'B, invokes current and forecast configuration information screen]

BITS PF10 [8 bit variable set to '01111010'B, is used for 'acknowledge' and/or 'screen update functions]

BITS PF11 [8 bit variable set to '01111011'B, invokes program function/menu program termination and parameters screens]

BITS PF12 [8 bit variable set to '01111100'B, is used for 'return to previously stored screen' function]

BITS PF13 [8 bit variable set to '11000001'B, invokes weather and wind planning log screen]

BITS PF14 [8 bit variable set to '10000010'B, invokes airport runway and surface planning log screen]

BITS PF15 [8 bit variable set to '11000011'B, invokes equipment planning log screen]

BITS PF16 [8 bit variable set to '11000100'B, invokes demand planning log screen]

BITS PA1 [8 bit variable set to '01101100'B, is used for stopping program under abnormal conditions]

BITS ENTER [8 bit variable set to '01111101'B, is used to signal screen inputs]
STRUCTURE CNFRQ (73)

BITES ID [24 bit configuration ID, where 1 indicates that a particular runway belongs to this configuration. First 12 bits used for arrival runways, next for departure runways starting with 4R to 32L]

CHR ARR_RNY (3) [character representation of arrival (length 3) runways in a configuration, e.g., 4R]

CHR DEP_RNY (4) [character representation (length 3) of departure runways in a configuration, e.g., 9L]

INT NORTH [integer index used for accessing a particular north complex capacity curve from capacity files]

INT SOUTH [integer index used for accessing a particular south complex capacity curve from capacity files]

INT ARR[6] [integers indicating fix to runway assignment for a particular configuration. Indices 1 through 6 signify fixes for 1-KUBBS, 2-PLANT, 3-CCT, 4-VAINS, 5-PABMM, 6-MILWAUKEE; contents of array holds numbers indicating arrival runways, 1-4R, 2-4L, ..., 12-32L]

INT DEP[5] [integers indicating fix to runway assignment for a particular configuration. Indices 1 through 5 signify fixes for 1-NORTH, 2-SOUTH, 3-EAST, 4-WEST, 5-MILWAUKEE; contents of array holds numbers indicating departure runways, 1-4R, 2-4L, ..., 12-32L]

ENDSTRUCTURE;
<table>
<thead>
<tr>
<th>Structure</th>
<th>OAGDEM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GROUP</strong></td>
<td>TABLE (0.23)</td>
</tr>
<tr>
<td><strong>CHR</strong></td>
<td><strong>GMT</strong> [character representation (length 4) of times associated with demand information given in Greenwich Mean Time]</td>
</tr>
<tr>
<td><strong>CHR</strong></td>
<td><strong>TTLARR</strong> [character representation (length 3) of total arrival demand]</td>
</tr>
<tr>
<td><strong>CHR</strong></td>
<td><strong>TTLDEP</strong> [character representation (length 3) of total departure demand]</td>
</tr>
<tr>
<td><strong>CHR</strong></td>
<td><strong>KUBBS</strong> [character representation (length 3) of arrival demand at fix KUBBS]</td>
</tr>
<tr>
<td><strong>CHR</strong></td>
<td><strong>CGT</strong> [character representation (length 3) of arrival demand at fix CGT]</td>
</tr>
<tr>
<td><strong>CHR</strong></td>
<td><strong>VAINS</strong> [character representation (length 3) of arrival demand at fix VAINS]</td>
</tr>
<tr>
<td><strong>CHR</strong></td>
<td><strong>NORTH</strong> [character representation (length 3) of arrival demand at fix NORTH]</td>
</tr>
<tr>
<td><strong>CHR</strong></td>
<td><strong>EAST</strong> [character representation (length 3) of departure demand at EAST fix]</td>
</tr>
<tr>
<td><strong>CHR</strong></td>
<td><strong>SOUTH</strong> [character representation (length 3) of departure demand at SOUTH fix]</td>
</tr>
<tr>
<td><strong>CHR</strong></td>
<td><strong>WEST</strong> [3 bit character representation (length 3) of departure demand at WEST fix]</td>
</tr>
<tr>
<td><strong>ENDSTRUCTURE:</strong></td>
<td></td>
</tr>
</tbody>
</table>
[***CAPACITY DATA***]

STRUCTURE CAPFILE (4) [four capacity files: VFR DRY, VFR WET, IFR DRY, IFR WET]
  GROUP KEY(80) [80 distinct capacity curves]
  INT PNUN [integer indicating number of points describing a particular curve]
  FLT CAP(14) [capacity points]
ENDSTRUCTURE;

[***DEPENDENCE MATRIX DATA***]

STRUCTURE DEPHAT(2) [2 dependence matrices: VFR, IFR]
  GROUP SECT(4) [4 partitions: ARR/ARR, ARR/DEP, DEP/ARR, DEP/DEP]
  FLT MATRIX (12,12) [individual dependence matrix entries]
ENDSTRUCTURE;

[***TRAVEL TIME DATA***]

FLT PIXTRAV (73,3,6) [73 configurations, up to 3 arrival runways, 6 arrival fixes, if a fix does not feed a particular runway entry is zero]
**RUNWAY MINIMA PARAMETERS**

**STRUCTURE**

<table>
<thead>
<tr>
<th>RUNWAY MINIMA (12)</th>
<th>[for each of 12 runways]</th>
</tr>
</thead>
</table>

**GROUP**

<table>
<thead>
<tr>
<th>CAT II</th>
<th>[pertaining to CAT II]</th>
</tr>
</thead>
</table>

**GROUP**

<table>
<thead>
<tr>
<th>NONE</th>
<th>FLY CEIL [ceiling minima with CATII operable]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FLY VIS [visibility minima with CATII operable]</td>
</tr>
</tbody>
</table>

**GROUP**

<table>
<thead>
<tr>
<th>ILS</th>
<th>[pertaining to ILS]</th>
</tr>
</thead>
</table>

**GROUP**

<table>
<thead>
<tr>
<th>NONE</th>
<th>FLY CEIL [ceiling minima with both localizer and glide slope operable]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FLY VIS [visibility minima with both localizer and glide slope operable]</td>
</tr>
</tbody>
</table>

**GROUP**

<table>
<thead>
<tr>
<th>MM</th>
<th>[pertaining to middle marker]</th>
</tr>
</thead>
</table>

**GROUP**

<table>
<thead>
<tr>
<th>RAIL, ALS</th>
<th>[pertaining to RAIL and ALS]</th>
</tr>
</thead>
</table>

**GROUP**

| FLY CEIL [ceiling minima with both localizer and glide slope operable, RAIL and ALS inoperable] |
|------|-----------------------------|
|      | FLY VIS [visibility minima with both localizer and glide slope operable, RAIL and ALS inoperable] |
GROUP TDZ [pertaining to TDZ]
  FLT CEIL [ceiling minima with both localizer and glide slope operable, TDZ inoperable]

GROUP CL [pertaining to CL]
  FLT CEIL [ceiling minima with both localizer and glide slope operable, CL inoperable]
  FLT VIS [visibility minima with both localizer and glide slope operable, CL inoperable]

GROUP LOC [pertaining to localizer]

GROUP NONE
  FLT CEIL [ceiling minima with localizer operable and glide slope inoperable]
  FLT VIS [visibility minima with localizer operable and glide slope inoperable]

GROUP MM [pertaining to middle marker]
  FLT CEIL [ceiling minima with localizer operable, glide slope and middle marker inoperable]
  FLT VIS [visibility minima with localizer operable, glide slope and middle marker inoperable]

GROUP RAIL [pertaining to RAIL]
  FLT CEIL [ceiling minima with localizer operable, glide slope and RAIL inoperable]
  FLT VIS [visibility minima with localizer operable, glide slope and RAIL inoperable]

GROUP ALS [pertaining to ALS]
  FLT CEIL [ceiling minima with localizer operable, glide slope and ALS inoperable]
  FLT VIS [visibility minima with localizer operable, glide slope and ALS inoperable]
GROUP  NDB_VOR [pertaining to NDB_VOR]

GROUP  NONE

FLT  CEIL  [ceiling minima with localizer inoperable and NDB_VOR operable]

FLT  VIS  [visibility minima with localizer inoperable and NDB_VOR operable]

GROUP  RAIL [pertaining to RAIL]

FLT  CEIL  [ceiling minima with NDB_VOR operable, localizer and RAIL inoperable]

FLT  VIS  [visibility minima with NDB_VOR operable, localizer and RAIL inoperable]

GROUP  ALS [pertaining to ALS]

FLT  CEIL  [ceiling minima with NDB_VOR operable, localizer and ALS inoperable]

FLT  VIS  [visibility minima with NDB_VOR operable, localizer and ALS inoperable]

ENDSTRUCTURE;
[AIRPORT STATUS INFORMATION FOR SCREEN USE]

STRUCTURE APTSTAT (2) [2 environments: current, forecast]

CHR TIME [character representation (length 8) of environment: 'CURRENT' or 'FORECAST']

GROUP WX [weather information]

CHR CEIL [character representation (length 4) of prevailing centerfield ceiling]

CHR VIS [character representation (length 4) of prevailing centerfield visibility]

GROUP WIND [wind information]

CHR DIR [character representation (length 3) of centerfield wind direction]

CHR VEL [character representation (length 3) of centerfield wind velocity]

GROUP RUNWAY (12) [12 runways]

GROUP TOWER [tower imposed runway conditions]

CHR ARR [character representation (length 2) of runway closures for arrivals]

CHR DEP [character representation (length 2) of runway closures for departures]

GROUP other_runway_information

CHR SURF [character representation (length 2) of runway surface conditions]

CHR BRK [character representation (length 2) of runway braking conditions]

CHR RVR [character representation (length 2) of runway RVR reading]

CHR DIR [character representation (length 3) of runway wind direction]
CUR VEL [character representation (length 2) of runway wind velocity]

CHR CRESS [character representation (length 2) of runway crosswind component]

CHR TAIL [character representation (length 2) of runway tailwind component]

CHR CEIL [character representation (length 4) of runway ceiling minima]

CHR VIS [character representation (length 4) of runway visibility minima]

GROUP CLOSED [overall runway closures]

CHR ARR [character representation (length 2) of arrival runway closures]

CHR DEP [character representation (length 2) of departure runway closures]

CHR MSG [character field (length 80) reserved for screen messages]

ENDSTRUCTURE;
[MIDWAY AIRPORT OPERATIONS INDICATOR FOR SCREEN USE]

CHR MIDFLAG (2) [character representation (length 2) of a flag indicating operation of runway 13R at MIDWAY airport for both current and forecast environments]

[AIRPORT STATUS DATA FOR SCREEN USE]

STRUCTURE CNVTAPT (2) [2 environments: current, forecast]

GROUP WX [weather data]
  FLT CEIL [prevailing airport ceiling]
  FLT VIS [prevailing airport visibility]

GROUP WIND [wind data]
  FLT DIR [airport's centerfield wind direction]
  FLT VEL [airport's centerfield wind velocity]

GROUP RUNWAY (12) [12 runways]
  FLT EVR [runway's EVR reading]
  FLT DIR [runway's wind direction]
  FLT VEL [runway's wind velocity]
  FLT CRSS [runway's crosswind component]
  FLT TAIL [runway's tailwind component]
  FLT CEIL [runway's ceiling minima]
  FLT VIS [runway's visibility minima]

ENDSTRUCTURE:
### Runway Equipment Status Information for Screen Use

**Structure**  
RWYEQP (2) [2 environments: current, forecast]

**Group Runway (12) [12 runways]**

<table>
<thead>
<tr>
<th>CHR</th>
<th>Representation</th>
<th>Length</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAT</td>
<td>Character indicator (length 2) for status of CAT II</td>
<td>2</td>
<td>CAT II status</td>
</tr>
<tr>
<td>LOC</td>
<td>Character indicator (length 2) for status of localizer</td>
<td>2</td>
<td>LOC status</td>
</tr>
<tr>
<td>GS</td>
<td>Character indicator (length 2) for status of glide slope</td>
<td>2</td>
<td>GS status</td>
</tr>
<tr>
<td>OM</td>
<td>Character indicator (length 2) for status of outer marker</td>
<td>2</td>
<td>OM status</td>
</tr>
<tr>
<td>MM</td>
<td>Character indicator (length 2) for status of middle marker</td>
<td>2</td>
<td>MM status</td>
</tr>
<tr>
<td>IM</td>
<td>Character indicator (length 2) for status of inner marker</td>
<td>2</td>
<td>IM status</td>
</tr>
<tr>
<td>RAIL</td>
<td>Character indicator (length 2) for status of runway alignment indicator lights</td>
<td>2</td>
<td>RAIL status</td>
</tr>
<tr>
<td>ALS</td>
<td>Character indicator (length 2) for status of approach lighting system</td>
<td>2</td>
<td>ALS status</td>
</tr>
<tr>
<td>RVR</td>
<td>Character indicator (length 2) for status of runway visual range</td>
<td>2</td>
<td>RVR status</td>
</tr>
<tr>
<td>HIRL</td>
<td>Character indicator (length 2) for status of high intensity runway lights</td>
<td>2</td>
<td>HIRL status</td>
</tr>
<tr>
<td>CL</td>
<td>Character indicator (length 2) for status of centerline lights</td>
<td>2</td>
<td>CL status</td>
</tr>
<tr>
<td>TDZ</td>
<td>Character indicator (length 2) for status of touchdown zone</td>
<td>2</td>
<td>TDZ status</td>
</tr>
<tr>
<td>NDB_VOR</td>
<td>Character indicator (length 2) for status of non-directional beacon/VHF omni-directional range</td>
<td>2</td>
<td>NDB_VOR status</td>
</tr>
</tbody>
</table>

**Message**  
[Character field (length 80) reserved for screen messages]
DEMAND PROFILE INFORMATION FOR SCREEN USE

STRUCTURE DEMAND (2) [2 environments: current, forecast]

CHR TIME [character representation (length 8) of environment: 'CURRENT', 'FORECAST']

GROUP ARR [arrival demand information]

CHR TOTAL [character representation (length 4) of total arrival demand]

CHR KUBBS [character representation (length 4) of arrival demand at fix KUBBS]

CHR PLANT [character representation (length 4) of arrival demand at fix PLANT]

CHR CGT [character representation (length 4) of arrival demand at fix CGT]

CHR VAINS [character representation (length 4) of arrival demand at fix VAINS]

CHR FARM [character representation (length 4) of arrival demand at fix FARM]

CHR MILWAUKEE [character representation (length 4) of arrival demand at fix MILWAUKEE]

GROUP DEP [departure demand information]

CHR TOTAL [character representation (length 4) of total departure demand TOTAL fix]

CHR NORTH [character representation (length 4) of departure demand at NORTH fix]

CHR EAST [character representation (length 4) of departure demand at EAST fix]

CHR SOUTH [character representation (length 4) of departure demand at SOUTH fix]

CHR WEST [character representation (length 4) of departure demand at WEST fix]

CHR MSG [character field (length 80) reserved for screen messages]

ENDSTRUCTURE;
STRUCTURE CMYTDEM (2) [2 environments: current, forecast]

GROUP ARR [arrival demand]
  FLT TOTAL [total arrival demand]
  FLT KUBBS [arrival demand at fix KUBBS]
  FLT PLANT [arrival demand at fix PLANT]
  FLT COT [arrival demand at fix COT]
  FLT VAINS [arrival demand at fix VAINS]
  FLT MARM [arrival demand at fix MARM]
  FLT MKE_A [arrival demand at fix MILWAUKEE]

GROUP DEP [departure demand]
  FLT TOTAL [total departure demand]
  FLT NORTH [departure demand at NORTH fix]
  FLT EAST [departure demand at EAST fix]
  FLT SOUTH [departure demand at SOUTH fix]
  FLT WEST [departure demand at WEST fix]
  FLT MKE_D [departure demand at MILWAUKEE fix]

ENDSTRUCTURE;
[CONFIGURATION INFORMATION SCREEN INFORMATION]

STRUCTURE CONFIG (2) [2 environments: current, forecast]

CHR TIME [character representation (length 8) of environment: 'CURRENT', 'FORECAST']

GROUP CONFIG

CHR ARR (12) [character representation (length 2) of configuration indicator for arrival runways]
CHR DEP (12) [character representation (length 2) of departure runways]

GROUP TOTAL [entire airport]

CHR PCT ARR [character representation (length 3) of airport's percentage of arrivals]
CHR SAT [character representation (length 4) of airport's saturation level]

GROUP ARR

CHR DEM [character representation (length 3) of airport's arrival demand]
CHR CAP [character representation (length 3) of airport's arrival capacity]

GROUP DEP

CHR DEM [character representation (length 3) of airport's departure demand]
CHR CAP [character representation (length 3) of airport's departure capacity]

GROUP NORTH

CHR PCT ARR [character representation (length 3) of percentage of arrivals for airport's north complex]
CHR SAT [character representation (length 4) of saturation level for airport's north complex]

GROUP ARR

CHR DEM [character representation (length 3) of arrival demand for airport's north complex]
CHR CAP [character representation (length 3) of arrival capacity for airport's north complex]
GROUP DEP

| CHRNEM | [character representation (length 3) of departure demand for airport's north complex] |
| CHRCAP | [character representation (length 3) of departure capacity for airport's north complex] |

GROUP SOUTH

| CHRPCT_ARR | [character representation (length 3) of percentage of arrivals for airport's south complex] |
| CHRSAT | [character representation (length 4) of saturation level for airport's south complex] |

GROUP ARR

| CHRDEN | [character representation (length 3) of arrival demand for airport's south complex] |
| CHRCAP | [character representation (length 3) of arrival capacity for airport's south complex] |

GROUP DEP

| CHRDEN | [character representation (length 3) of departure demand for airport's south complex] |
| CHRCAP | [character representation (length 3) of departure capacity for airport's south complex] |

GROUP BALANCING [demand balancing information]

| CHRMMOVE | [character representation (length 3) of number of arrivals moved from one complex to other for purpose of demand balancing] |
| CHRACOMPLEX | [character representation complex to which arrivals are moved] |
| CHRMMOVE | [character representation (length 3) of number of departures moved from one complex to other for purpose of demand balancing] |
| CHRDCOMPLEX | [character representation (length 5) of complex to which departures are moved] |

| CHRWMSCO | [character field (length 80) reserved for user warning messages] |
| CHRWMSC1 | [character field (length 80) reserved for user warning messages] |
| CHRWMSC2 | [character field (length 80) reserved for user warning messages] |
| CHRMSG | [character field (length 80) reserved for screen messages] |

ENDSTRUCTURE;
[CONFIGURATION INDEX FOR CMS USE]

INT CONFIND (2) [integer index indicating operating configuration in current and forecast environment]

[ORDERED LIST OF CONFIGURATIONS INFORMATION FOR SCREEN USE]

STRUCTURE COMFLST(2) [2 environments: current, forecast]

CHR TIME [character representation (length 8) of environment: 'CURRENT', 'FORECAST']

CHR TOT_ARR [character representation (length 3) of airport's percentage of arrivals]

CHR NUMBER [character representation (length 3) of number of eligible configurations]

CHR SCROLL [character representation (length 4) of screen scroll number]

GROUP COMFIC (73) [up to 73 eligible configurations]

CHR SELECT [character representation (length 1) used to indicate and select a new current configuration]

CHR RANK [character representation (length 2) of rank of a particular configuration in ordered list of configurations]

CHR ARR (3) [character representation (length 3) of arrival runways in a particular configuration]

CHR DEP (4) [character representation (length 3) of departure runways in a particular configuration]

CHR CAPACITY [character representation (length 5) of capacity of a particular configuration]

CHR REMARKS [character representation field (length 27) used for additional information on each configuration]

CHR MSG [character field (length 80) reserved for screen messages]

ENDSTRUCTURE;
[ORDERED LIST OF TRANSITIONS INFORMATION FOR SCREEN USE]

STRUCTURE

TRANSLST

CHR PCT_ARR [character representation (length 3) of airport's percentage of arrivals for forecast environment]

CHR NUM_ELIG [character representation (length 3) of number of eligible configuration in forecast environment]

CHR SCROLL [character representation (length 4) of screen scroll number]

CHR ARR (3) [character representation (length 3) of current configuration's arrival runways, e.g., 14R, 4L (up to 3 arrival runways)]

CHR DEP [character representation (length 3) of current configuration's departure runways, e.g., 14R, 32L, (up to 4 departure runways)]

CHR CTRANHR [character representation (length 5) of current configuration's transition hour capacity]

CHR CFINCAP [character representation (length 5) of current configuration's capacity in forecast environment]

GROUP CONFIG (73) [73 possible configurations]

CHR RANK [character representation (length 3) of rank of a particular configuration in ordered list of transitions]

CHR ARR(3) [character representation (length 3) of forecast configuration's arrival runways, e.g., 14R, 48, up to 3 arrival runways]

CHR DEP(4) [character representation (length 3) of forecast configuration's departure runways, e.g., 32R, 32L, runways, (up to 4 departure runways)]

CHR MINUTES [character representation (length 2) of transition duration]

CHR HOURLY [character representation (length 5) of transition hour capacity for a configuration eligible in forecast environment]

CHR FINCAP [character representation (length 5) of a forecast configuration's capacity in forecast environment]

CHR MSG [character field (length 80) reserved for screen messages]

ENDSTRUCTURE;
[RUNWAY EQUIPMENT PLANNING LOG INFORMATION FOR SCREEN USE]

STRUCTURE EQPLOG

GROUP TABLE (15) [up to 15 lines of log entries]

CHR RNY [character representation (length 3) of runway used in log entry, e.g., 14R]

CHR EQUIPMENT [character field (length 11) reserved for equipment name used in equipment log]

CHR OTS [character representation (length 4) of "OUT OF SERVICE" times used in equipment log]

CHR RTS [character representation (length 4) of "RETURN TO SERVICE" times used in equipment log]

CHR REMARKS [character field (length 39) reserved for free formatted comments used in equipment log]

CHR MSG [character field (length 80) reserved for screen messages]

ENDSTRUCTURE;

[RUNWAY EQUIPMENT PLANNING LOG DATA FOR SCREEN USE]

STRUCTURE CNVTEQP

GROUP TABLE (15) [up to 15 lines of log entries]

INT OTS [integer representing "OUT OF SERVICE" time used in equipment log]

INT RTS [integer representing "RETURN TO SERVICE" time used in equipment log]

ENDSTRUCTURE;
[WEATHER AND WIND PLANNING LOG INFORMATION FOR SCREEN USE]

STRUCTURE  WELOG

GROUP  TABLE (13)  [up to 13 lines of log entries]

<table>
<thead>
<tr>
<th>CHR</th>
<th>TIME</th>
<th>[character representation (length 4) of time used in weather and wind planning log]</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHR</td>
<td>CEIL</td>
<td>[character representation (length 5) of ceiling used in weather and wind log]</td>
</tr>
<tr>
<td>CHR</td>
<td>VIS</td>
<td>[character representation (length 5) of visibility used in weather and log]</td>
</tr>
<tr>
<td>CHR</td>
<td>DIR</td>
<td>[character representation (length 5) of wind direction used in weather and wind log]</td>
</tr>
<tr>
<td>CHR</td>
<td>VEL</td>
<td>[character representation (length 5) of wind velocity used in weather and wind log]</td>
</tr>
<tr>
<td>CHR</td>
<td>REMARKS</td>
<td>[character field (length 35) reserved for free formatted comments used in weather and wind log]</td>
</tr>
<tr>
<td>CHR</td>
<td>MSG</td>
<td>[character field (length 80) reserved for screen messages]</td>
</tr>
</tbody>
</table>

ENDSTRUCTURE;
STRUCTURE CNVTWX

GROUP TABLE (13) [up to 13 log entries]

INT TIME [integer representing time used in weather and wind planning log]

FLT CEIL [floating point variable used to represent value of ceiling in weather and wind planning log]

FLT VIS [floating point variable used to represent value of visibility in weather and wind planning log]

FLT DIR [floating point variable used to represent value of wind direction in weather and wind planning log]

FLT VEL [floating point variable used to represent value of wind velocity in weather and wind planning log]

ENDSTRUCTURE;
[RUNWAY SURFACE CONDITIONS PLANNING LOG INFORMATION FOR SCREEN USE]

STRUCTURE SURFLOG
 GROUP TABLE (13) [up to 13 log entries]
   CHR TIME [character representation (length 4) of time used in weather and wind planning log]
   CHR RUN [character representation (length 3) of a runway in an entry in runway surface conditions planning log]
   CHR SURF [character field (length 5) reserved for description of runway surface conditions in runway surface conditions planning log]
   CHR BRAK [character field (length 5) reserved for description of runway braking conditions in runway surface conditions planning log]
   CHR CLOSED [character field (length 6) reserved for information on runway closures on the runway surface conditions planning log]
   CHR OPEN [character field (length 6) reserved for information on runway openings on the runway surface conditions planning log]
   CHR REMARKS [character field (length 27) reserved for free formatted comments used in runway surface conditions planning log]
   CHR MSG [character field (length 80) reserved for screen messages]
 ENDSTRUCTURE;

[RUNWAY SURFACE CONDITIONS PLANNING LOG DATA FOR SCREEN USE]

STRUCTURE CWTSRF
 GROUP TABLE (13) [up to 13 log entries]
   INT TIME [integer representing time used in runway surface conditions planning log]
 ENDSTRUCTURE;
[DEMAND PLANNING LOG INFORMATION FOR SCREEN USE]

STRUCTURE OAGLOG

CHR INITIAL [character field (length 2) used to initialize demand planning log screen with nominal demand values from OAG demand file]

CHR SCROLL [character representation (length 4) of scroll number used on demand planning log screen]

GROUP TABLE (0:23) [24 hours of demand data available on demand planning screen]

CHR GMT [character representation (length 4) of time (hour) for demand information]

CHR TTLARR [character representation (length 3) of total hourly arrival demand]

CHR TTLDEP [character representation (length 3) of total hourly departure demand]

CHR KURBS [character representation (length 3) of hourly arrival demand at fix KURBS]

CHR CGT [character representation (length 3) of hourly arrival demand at fix CGT]

CHR VAINS [character representation (length 3) of hourly arrival demand at fix VAINS]

CHR FARN [character representation (length 3) of hourly arrival demand at fix FARN]

CHR NORTH [character representation (length 3) of hourly departure demand at NORTH fix]

CHR EAST [character representation (length 3) of hourly departure demand at EAST fix]

CHR SOUTH [character representation (length 3) of hourly departure demand at SOUTH fix]

CHR WEST [character representation (length 3) of hourly departure demand at WEST fix]

CHR MSG [character field (length 80) reserved for screen message]

ENDSTRUCTURE.
[DEMAND PLANNING LOG DATA FOR SCREEN USE]

```c
STRUCTURE CNTOAG

INT SCROLL [integer representing scroll number for demand planning log]

GROUP TABLE (0:23) [24 hours of demand data available on demand planning screen]

INT CNT [integer value of time (hour) for demand information]

FLT TTLARR [floating point value of total hourly arrival demand]

FLT TTLOEP [floating point value of total hourly departure demand]

FLT KUBBS [floating point value of hourly arrival demand at fix KUBBS]

FLT CCT [floating point value of hourly arrival demand at fix CCT]

FLT VAINS [floating point value of hourly arrival demand at fix VAINS]

FLT PARMM [floating point value of hourly arrival demand at fix PARMM]

FLT NORTH [floating point value of hourly departure demand at NORTH fix]

FLT EAST [floating point value of hourly departure demand at EAST fix]

FLT SOUTH [floating point value of hourly departure demand at SOUTH fix]

FLT WEST [floating point value of hourly departure demand at WEST fix]

ENDSTRUCTURE;
```
PARAMETERS INFORMATION FOR SCREEN USE

STRUCTURE PARAM

GROUP PARAMETER [wind thresholds]
  GROUP ARR [pertaining to arrivals]
    CHR CRSS [character representation (length 4) of crosswind component of wind thresholds]
    CHR TAIL [character representation (length 4) of tailwind component of wind thresholds]
  GROUP DEP [pertaining to departures]
    CHR CRSS [character representation (length 4) of crosswind component of wind thresholds]
    CHR TAIL [character representation (length 4) of tailwind component of wind thresholds]
GROUP MSG [character field (length 80) reserved for screen messages]
ENDSTRUCTURE;

PARAMETERS SCREEN DATA FOR SCREEN USE

STRUCTURE CNVTPRN

GROUP ARR [wind thresholds for arrivals]
  FLT CRSS [crosswind component of wind threshold]
  FLT TAIL [tailwind component of wind threshold]
GROUP DEP [wind thresholds for departures]
  FLT CRSS [crosswind component of wind threshold]
  FLT TAIL [tailwind component of wind threshold]
ENDSTRUCTURE;
[DEPARTURE QUEUES INFORMATION FOR SCREEN USE]

**STRUCTURE QUELEN**

CHR DEPRUN (4) [character representation (length 3) of current configuration's departure runways]

CHR QL (4) [character representation (length 2) of number of aircraft in departure queue]

CHR MSG [character field (length 80) reserved for screen messages]

ENDSTRUCTURE;

[DEPARTURE QUEUES DATA FOR SCREEN USE]

INT CNTQLN (4) [integer representing length of departure queues]
STRUCTURE O'HARE

GROUP WX [weather information]
  CHR CEIL [character representation (length 4) of prevailing ceiling]
  CHR VIS [character representation (length 4) of prevailing visibility]

GROUP WIND [wind information]
  CHR DIR [character representation (length 3) of wind direction]
  CHR VEL [character representation (length 2) of wind velocity]

CHR ARR (3) [character representation (length 3) of current configuration's arrival runways, e.g., 14R, 4R (up to 3 arrival runways)]

CHR DEP (4) [character representation (length 4) of current configuration's departure runways, e.g., 14R, 32L, (up to 4 departure runways)]

CHR CAPACITY [character representation (length 5) of current configuration's capacity]

CHR PCT_HC [character representation (length 3) of relationship of capacity for the current runway configuration to maximum capacity achievable for current conditions]

CHR SCROLL [character representation (length 4) of screen scroll number]

CHR LOG_MSG(13) [character representation (length 80) of log messages appearing in O'Hare status screen, (up to 13 messages)]

CHR MSG [character field (length 80) reserved for screen messages]

ENDSTRUCTURE;
[TABLE STORING OLD LOG MESSAGES ON O'HARE STATUS SCREEN]

STRUCTURE OLDLOGS

GROUP TABLE (108) [up to 108 log messages are stored]

INT TIME [integer representing time associated with each message]

CHR MSG [character field (length 80) storing content of each log message]

ENDSTRUCTURE;

[CHARACTER REPRESENTATION OF TIME WHEN CONTENTS OF EACH SCREEN WAS LAST STORED IN DATA BASE]

STRUCTURE STORED

CHR OHSTATUS [character representation (length 4) of time when contents of O'Hare status screen was last stored in data base]

CHR PARNOPT [character representation (length 4) of time when contents of parameter screen was last stored in data base]

CHR APLOG1 [character representation (length 4) of time when contents of wind and weather planning log screen was last stored in data base]

CHR APLOG2 [character representation (length 4) of time when contents of airport surface conditions planning log screen was last stored in data base]

CHR AIRPORT(2) [character representation (length 4) of times when contents of airport status screen was last stored in data base (current and forecast)]
CHR RWYLOG character representation (length 4) of time when contents of runway equipment log screen was last stored in data base
CHR RUNWAY [character representation (length 4) of time when contents of runway equipment status screen was last stored in data base (current and forecast)]
CHR DMNDLOG [character representation (length 4) of time when contents of demand planning log screen was last stored in data base]
CHR DEMAND (2) [character representation (length 4) of times when contents of demand profile screen was last stored in data base (current and forecast)]
CHR OLIST (2) [character representation (length 4) of times when contents of ordered list of configurations screen was last stored in data base (current and forecast)]
CHR QLENGTH [character representation (length 4) of time when contents of current departure queue screen was last stored in data base]
CHR TRANSLIST [character representation (length 4) of time when contents of ordered list of transition screen was last stored in data base]
CHR CONF (2) [character representation (length 4) of times when contents of configuration information screen was last stored in data base (current and forecast)]
ENDSTRUCTURE;
CALCULATED VARIABLES

[AIRPORT DEMAND DATA]

STRUCTURE PREARR (2) [2 environments: current and forecast]
  FLT TOTAARR [total arrival demand]
  FLT TOTDEP [total departure demand]
GROUP COMP (73) [73 configurations]
  FLT NPROCNT [north complex percentage of arrivals]
  FLT SPROCNT [south complex percentage of arrivals]
  FLT NAREDEM [north complex arrival demand]
  FLT SAREDEM [south complex arrival demand]
  FLT NDEPDEM [north complex departure demand]
  FLT SDEPDEM [south complex departure demand]
  FLT BNPROCNT [north complex percentage of arrivals after demand balancing]
  FLT BSPROCNT [south complex percentage of arrivals after demand balancing]
  FLT BNAREDEM [north complex balanced arrival demand]
  FLT BSAREDEM [south complex balanced arrival demand]
  FLT BNDEPDEM [north complex balanced departure demand]
  FLT BSDEPDEM [south complex balanced departure demand]
ENDSTRUCTURE;
STRUCTURE [AIRPORT INFORMATION DATA]

GROUP CONF (73) [up to 73 configurations]

INT INDEX [index associated with each configuration for a table lookup]

FLT CAPACITY [total configuration capacity]

FLT NARRCAP [arrival capacity of north complex]

FLT NDRCAP [departure capacity of north complex]

FLT SARRCAP [arrival capacity of south complex]

FLT SDRCAP [departure capacity of south complex]

FLT SATURATION [airport saturation level]

FLT NSAT [north complex saturation level]

FLT SSAT [south complex saturation level]

INT CHANGENARR [change in north complex arrival demand due to demand balancing]

INT CHANGENDP [change in north complex departure demand due to demand balancing]

ENDSTRUCTURE;

[CAPACITY FILE CLASSIFICATION OF EACH CONFIGURATION]

STRUCTURE FILENUM (2) [2 environments: current, forecast]

INT CONF (73) [capacity file classification for each configuration, 1 - VFR DRY, 2 - VFR WET, 3 - IFR DRY, 4 - IFR WET]

ENDSTRUCTURE;
[ELIGIBILITY DATA]

STRUCTURE ELIGIBILITY (2) [2 environments: current, forecast]

BITS ID [7] bits string signifying eligibility, 0-eligible, 1-ineligible
INT NUM [number of eligible configurations]

ENDSTRUCTURE;

[CATEGORY DATA]

INT CNDRN (2) [1-up, 2-down]
[AIRPORT STATUS INFORMATION MOST CURRENT FROM DATA BASE]

STRUCTURE AIRW (2) (2 environments: current, forecast)

GROUP WX

CHR CEIL [character representation (length 4) of prevailing centerfield ceiling]

CHR VIS [character representation (length 4) of prevailing centerfield visibility]

GROUP WIND

CHR DIR [character representation (length 3) of centerfield wind direction]

CHR VEL [character representation (length 3) of wind centerfield velocity]

GROUP RUNWAY (12) (12 runways)

GROUP TOWER [tower imposed runway conditions]

CHR ARR [character representation (length 2) of runway closures for arrivals]

CHR DEP [character representation (length 2) of runway closures for departures]

GROUP OTHER_RUNWAY_INFORMATION

CHR SURF [character representation (length 2) of runway surface conditions]

CHR BRK [character representation (length 2) of runway braking conditions]

ENDSTRUCTURE;
MIDWAY AIRPORT OPERATIONS INDICATOR MOST CURRENT FROM DATA BASE

CHR KNOW (2) [character representation (length 2) of a flag indicating operation of runway 13R at MIDWAY airport for both current and forecast environments]

AIRPORT STATUS DATA MOST CURRENT FROM DATA BASE

STRUCTURE CVTANOW (2) [2 environments: current, forecast]

GROUP WX

  FLT CHIL [prevailing airport ceiling]

  FLT VIS [prevailing airport visibility]

GROUP WIND

  FLT DIR [airport's centerfield wind direction]

  FLT VEL [airport's centerfield wind velocity]

ENDSTRUCTURE;
[RUNWAY EQUIPMENT STATUS INFORMATION MOST CURRENT FROM DATA BASE]

STRUCTURE BNOW (2) [2 environments: current, forecast]

GROUP RUNWAY (12) [12 runways]

CHR CAT II [character indicator for status of CAT II]

CHR LOC [character indicator (length 2) for status of localizer]

CHR GS [character indicator (length 2) for status of glide slope]

CHR CM [character indicator (length 2) for status of outer marker]

CHR MN [character indicator (length 2) for status of middle marker]

CHR IN [character indicator (length 2) for status of inner marker]

CHR RAIL [character indicator (length 2) for status of runway alignment indicator lights]

CHR ALS [character indicator (length 2) for status of approach lighting system]

CHR RVR [character indicator (length 2) for status of runway visual range]

CHR MBL [character indicator (length 2) for status of high intensity runway lights]

CHR CL [character indicator (length 2) for status of centerline lights]

CHR TDZ [character indicator (length 2) for status of touchdown zone]

CHR NDB_VOR [character indicator (length 2) for status of non-directional beacon/VHF omni-directional range]

ENDSTRUCTURE;
DEMAND PROFILE INFORMATION MOST CURRENT FROM DATABASE

STRUCTURE DNOW (2) [2 environments: current, forecast]

GROUP ARR [arrival demand information]

CHR TOTAL [character representation (length 4) of total arrival demand]
CHR KUBBS [character representation (length 4) of arrival demand at fix KUBBS]
CHR PLANT [character representation (length 4) of arrival demand at fix PLANT]
CHR CGT [character representation (length 4) of arrival demand at fix CGT]
CHR VAINS [character representation (length 4) of arrival demand at fix VAINS]
CHR PARDN [character representation (length 4) of arrival demand at fix PARDN]
CHR MILK [character representation (length 4) of arrival demand at fix MILK]

GROUP DEP [departure demand information]

CHR TOTAL [character representation (length 4) of total departure demand]
CHR NORTH [character representation (length 4) of departure demand at NORTH fix]
CHR EAST [character representation (length 4) of departure demand at EAST fix]
CHR SOUTH [character representation (length 4) of departure demand at SOUTH fix]
CHR WEST [character representation (length 4) of departure demand at WEST fix]

ENDSTRUCTURE
[DEMAND PROFILE DATA MOST CURRENT FROM DATA BASE]

STRUCTURE CVTDNOW (2) [2 environments: current, forecast]

GROUP ARR [arrival demand]
  FLT TOTAL [total arrival demand]
  FLT KUBBS [arrival demand at fix KUBBS]
  FLT PLANT [arrival demand at fix PLANT]
  FLT CGT [arrival demand at fix CGT]
  FLT VAINS [arrival demand at fix VAINS]
  FLT FARM [arrival demand at fix FARM]
  FLT HILWAUEE [arrival demand at fix MILWAUKEE]

GROUP DEP [departure demand]
  FLT TOTAL [total departure demand]
  FLT NORTH [departure demand at NORTH fix]
  FLT EAST [departure demand at EAST fix]
  FLT SOUTH [departure demand at SOUTH fix]
  FLT WEST [departure demand at WEST fix]
  FLT MILWAUKEE [departure demand at MILWAUKEE fix]

ENDSTRUCTURE;
[CONFIGURATION INDEX MOST CURRENT FROM DATA BASE]

INT CMOW (2) [integer index indicating operating configuration in current and forecast environment]

[RUNWAY EQUIPMENT PLANNING LOG INFORMATION MOST CURRENT FROM DATA BASE]

STRUCTURE ENOW

GROUP TABLE (15) [up to 15 lines of log entries]

CHR RMY [character representation (length 3) of a runway used in log entry, e.g., 14R]

CHR EQUIPMENT [character field (length 11) reserved for equipment name used in equipment log]

CHR OTS [character representation (length 4) of "OUT OF SERVICE" times used in equipment log]

CHR RTS [character representation (length 4) of "RETURN TO SERVICE" times used in equipment log]

CHR REMARKS [character field (length 39) reserved for free-formatted comments used in equipment log]

ENDSTRUCTURE:

[RUNWAY EQUIPMENT PLANNING LOG DATA MOST CURRENT FROM DATA BASE]

STRUCTURE CVTENOW

GROUP TABLE (15) [up to 15 lines of log entries]

INT OTS [integer representing "OUT OF SERVICE" time used in equipment log]

INT RTS [integer representing "RETURN TO SERVICE" time used in equipment log]

ENDSTRUCTURE:
[WEATHER AND WIND PLANNING LOG INFORMATION MOST CURRENT FROM DATA BASE]

STRUCTURE

GROUP TABLE (13) [up to 13 lines of log entries]

CHR TIME [character representation (length 4) of time used in weather and wind planning log]

CHR CEIL [character representation (length 5) of ceiling used in weather and wind log]

CHR VIS [character representation (length 5) of visibility used in weather and log]

CHR DIR [character representation (length 5) of wind direction used in weather and wind log]

CHR VEL [character representation (length 5) of wind velocity used in weather and wind log]

CHR REMARKS [character field (length 35) reserved for free-formatted comments used in weather and wind log]

ENDSTRUCTURE;
STRUCTURE CVTWWN

GROUP TABLE (13) [up to 13 log entries]

INT TIME [integer representing time used in weather and wind planning log]

FLT CELL [floating point variable used to represent value of ceiling in weather and wind planning log]

FLT VIS [floating point variable used to represent value of visibility in weather and wind planning log]

FLT DIR [floating point variable used to represent value of wind direction in weather and wind planning log]

FLT VEL [floating point variable used to represent value of wind velocity in weather and wind planning log]

ENDSTRUCTURE.
RUNWAY SURFACE CONDITIONS PLANNING LOG INFORMATION MOST CURRENT FROM DATA BASE

STRUCTURE SNOW

GROUP TABLE (13) [up to 13 log entries]

CHR TIME [character representation (length 4) of time used in runway surface conditions planning log]

CHR RW Y [character representation (length 3) of runway in runway surface conditions planning log]

CHR SURF [character field (length 5) reserved for description of runway surface conditions in runway surface conditions planning log]

CHR BRAK [character field (length 5) reserved for description of runway braking conditions in runway surface conditions planning log]

CHR CLOSED [character field (length 6) reserved for information on runway closures on the runway surface conditions planning log]

CHR OPEN [character field (length 6) reserved for information on runway openings on the runway surface conditions planning log]

CHR REMARKS [character field (length 27) reserved for free-formatted comments used in runway surface conditions planning log]

ENDSTRUCTURE;

RUNWAY SURFACE CONDITIONS PLANNING LOG DATA MOST CURRENT FROM DATA BASE

STRUCTURE CVTSNOW

GROUP TABLE (13) [up to 13 log entries]

INT TIME [integer representing time used in runway surface conditions planning log]

ENDSTRUCTURE;
[DEMAND PLANNING LOG SCREEN INFORMATION MOST CURRENT FROM DATA BASE]

STRUCTURE GNOW

GROUP TABLE (0:23) [24 hours of demand data available on demand planning screen]
  
  CHR GMT [character representation (length 4) of time (hour) for demand information]
  CHR TTLARR [character representation (length 3) of total hourly arrival demand]
  CHR TTLDEP [character representation (length 3) of total hourly departure demand]
  CHR KUBBS [character representation (length 3) of hourly arrival demand at fix KUBBS]
  CHR CGT [character representation (length 3) of hourly arrival demand at fix CGT]
  CHR VAINS [character representation (length 3) of hourly demand at fix VAINS]
  CHR PARDON [character representation (length 3) of hourly arrival demand at fix PARDON]
  CHR NORTH [character representation (length 3) of hourly departure demand at NORTH fix]
  CHR EAST [character representation (length 3) of hourly departure demand at EAST fix]
  CHR SOUTH [character representation (length 3) of hourly departure demand at SOUTH fix]
  CHR WEST [character representation (length 3) of hourly departure demand at WEST fix]

ENDSTRUCTURE;
STRUCTURE CVTNOW

GROUP TABLE (0:23) [24 hours of demand data available on demand planning screen]

INT GMT [integer value of time (hour) for demand information]
FLT TTLARR [floating point value of total hourly arrival demand]
FLT TTLDEP [floating point value of total hourly departure demand]
FLT CGT [floating point value of hourly arrival demand at fix CGT]
FLT VAIN [floating point value of hourly arrival demand at fix VAINS]
FLT FARM [floating point value of hourly arrival demand at fix FARM]
FLT NORTH [floating point value of hourly departure demand at NORTH fix]
FLT EAST [floating point value of hourly departure demand at EAST fix]
FLT SOUTH [floating point value of hourly departure demand at SOUTH fix]
FLT WEST [floating point value of hourly departure demand at WEST fix]

ENDSTRUCTURE;
[PARAMETERS INFORMATION MOST CURRENT FROM DATA BASE]

STRUCTURE PNOW

GROUP PARAMETER [wind thresholds]

GROUP ARR [pertaining to arrivals]

CHR CRSS [character representation (length 4) of crosswind component of wind thresholds]

CHR TAIL [character representation (length 4) of tailwind component of wind thresholds]

GROUP DEP [pertaining to departures]

CHR CRSS [character representation (length 4) of crosswind component of wind thresholds]

CHR TAIL [character representation (length 4) of tailwind component of wind thresholds]

ENDSTRUCTURE;

[PARAMETERS DATA MOST CURRENT FROM DATA BASE]

STRUCTURE CVTPNOW

GROUP ARR [wind thresholds for arrivals]

FLT CRSS [crosswind component of wind threshold]

FLT TAIL [tailwind component of wind threshold]

GROUP DEP [wind thresholds for departures]

FLT CRSS [crosswind component of wind threshold]

FLT TAIL [tailwind component of wind threshold]

ENDSTRUCTURE;
STRUCTURE QNOW

CHR QL (4) [character representation (length 2) of number of aircraft in departure queue]

ENDSTRUCTURE;

INT CVTQNOW (4) [integer representing length of departure queues]
[AIRPORT STATUS INFORMATION ORIGINAL FROM DATABASE]

STRUCTURE BEGIN (2) [2 environments: current, forecast]

GROUP WX

CHR CEIL [4 bit character representation of prevailing ceiling]

CHR VIS [4 bit character representation of prevailing visibility]

GROUP WIND

CHR DIR [3 bit character representation of wind direction]

CHR VEL [3 bit character representation of wind velocity]

GROUP RUNWAY (12) [12 runways]

GROUP TOWER [tower imposed runway conditions]

CHR ARR [2 bit character representation of runway closures for arrivals]

CHR DEP [2 bit character representation of runway closures for departures]

GROUP OTHER_RUNWAY_INFORMATION

CHR SURF [2 bit character representation of runway surface conditions]

CHR BRK [2 bit character representation of runway braking conditions]

ENDSTRUCTURE
MIDWAY AIRPORT OPERATIONS INDICATOR ORIGINAL FROM DATA BASE

CHR MBEGIN (2) [2 byte character representation of a flag indicating Midway airport is operational for current and forecast environment]

[AIRPORT STATUS DATA ORIGINAL FROM DATA BASE]

STRUCTURE CVTABGN (2) [2 environments: current, forecast]

GROUP WX

  FLT CEIL [prevailing airport ceiling]
  FLT VIS [prevailing airport visibility]

GROUP WIND

  FLT DIR [airport's centerfield wind direction]
  FLT VEL [airport's centerfield wind velocity]

ENDSTRUCTURE;
[RUNWAY EQUIPMENT STATUS INFORMATION ORIGINAL FROM DATA BASE]

STRUCTURE BEGIN (2) [2 environments: current, forecast]

GROUP RUNWAY (12) [12 runways]

- **CHR CAT II** [2 bit character indicator for status of CAT II]
- **CHR LOC** [2 bit character indicator for status of localizer]
- **CHR GS** [2 bit character indicator for status of glide slope]
- **CHR OM** [2 bit character indicator for status of outer marker]
- **CHR MM** [2 bit character indicator for status of middle marker]
- **CHR IM** [2 bit character indicator for status of inner marker]
- **CHR RAIL** [2 bit character indicator for status of runway alignment indicator lights]
- **CHR ALS** [2 bit character indicator for status of approach lighting system]
- **CHR RVR** [2 bit character indicator for status of runway visual range]
- **CHR HIRL** [2 bit character indicator for status of high intensity runway lights]
- **CHR CL** [2 bit character indicator for status of centerline lights]
- **CHR TD2** [2 bit character indicator for status of touchdown zone]
- **CHR NDB_VOR** [2 bit character indicator for status of non-directional beacon/VHF omni-directional range]

ENDSTRUCTURE;
[DEMAND PROFILE INFORMATION ORIGINAL FROM DATA BASE]

STRUCTURE BEGIN (2) [2 environments: current, forecast]

GROUP ARR [arrival demand information]
CHR TOTAL [4 bit character representation of total arrival demand]
CHR KUBBS [4 bit character representation of arrival demand at fix KUBBS]
CHR PLANT [4 bit character representation of arrival demand at fix PLANT]
CHR COT [4 bit character representation of arrival demand at fix COT]
CHR VAINS [4 bit character representation of arrival demand at fix VAINS]
CHR FARM [4 bit character representation of arrival demand at fix FARM]
CHR MILWAUIKEE [4 bit character representation of arrival demand at fix MILWAUIKEE]

GROUP DEP [departure demand information]
CHR TOTAL [4 bit character representation of total departure demand]
CHR NORTH [4 bit character representation of departure demand at NORTH fix]
CHR EAST [4 bit character representation of departure demand at EAST fix]
CHR SOUTH [4 bit character representation of departure demand at SOUTH fix]
CHR WEST [4 bit character representation of departure demand at WEST fix]

ENDSTRUCTURE;
[DEMAND PROFILE DATA ORIGINAL FROM DATA BASE]

STRUCTURE CVTDGBK (2) [2 environments: 'current', 'forecast']

GROUP ARR [arrival demand]
  FLT TOTAL [total arrival demand]
  FLT KUBBS [arrival demand at fix KUBBS]
  FLT PLANT [arrival demand at fix PLANT]
  FLT COT [arrival demand at fix COT]
  FLT VAINS [arrival demand at fix VAINS]
  FLT FARMN [arrival demand at fix FARMN]
  FLT MILK [arrival demand at fix MILK]

GROUP DEP [departure demand]
  FLT TOTAL [total departure demand]
  FLT NORTH [departure demand at NORTH fix]
  FLT EAST [departure demand at EAST fix]
  FLT SOUTH [departure demand at SOUTH fix]
  FLT WEST [departure demand at WEST fix]
  FLT MILK [departure demand at MILK fix]

ENDSTRUCTURE;
[CONFIGURATION INDEX ORIGINAL FROM DATA BASE]

INT CBEGIN (2) [integer index indicating current configuration in current and forecast environment]

[RUNWAY EQUIPMENT PLANNING LOG INFORMATION ORIGINAL FROM DATA BASE]

STRUCTURE EBEGIN
GROUP TABLE (15) [up to 15 lines of log entries]
  CHR rwy [3 bit character representation of a runway used in log entry, e.g., 1AR]
  CHR EQUIPMENT [11 bit character field reserved for equipment name used in equipment log]
  CHR OTS [4 bit character representation of "OUT OF SERVICE" times used in equipment log]
  CHR RTS [4 bit character representation of "RETURN TO SERVICE" times used in equipment log]
  CHR REMARKS [39 bit character field reserved for free formatted comments used in equipment log]
ENDSTRUCTURE;

[RUNWAY EQUIPMENT PLANNING LOG DATA ORIGINAL FROM DATA BASE]

STRUCTURE CBEGIN
GROUP TABLE (15) [up to 15 lines of log entries]
  INT OTS [integer representing "OUT OF SERVICE" time used in equipment log]
  INT RTS [integer representing "RETURN TO SERVICE" time used in equipment log]
ENDSTRUCTURE;
STRUCTURE BEGIN

GROUP TABLE (13) [up to 13 lines of log entries]

CHR TIME [4 bit character representation of time used in weather and wind planning log]

CHR CEIL [5 bit character representation of prevailing ceiling used in weather and wind log]

CHR VIS [5 bit character representation of prevailing visibility used in weather and log]

CHR DIR [5 bit character representation of wind direction used in weather and wind log]

CHR VEL [5 bit character representation of wind velocity used in weather and wind log]

CHR REMARKS [35 bit character field reserved for free-formatted comments used in weather and wind log]

ENDSTRUCTURE;
[WEATHER AND WIND PLANNING LOG DATA ORIGINAL FROM DATA BASE]

STRUCTURE CVTWBN

GROUP TABLE (13) [up to 13 log entries]

  INT TIME [integer representing time used in weather and wind planning log]
  FLT CEIL [floating point variable used to represent value of prevailing ceiling in weather and wind planning log]
  FLT VIS [floating point variable used to represent value of prevailing visibility in weather and wind planning log]
  FLT DIR [floating point variable used to represent value of wind direction in weather and wind planning log]
  FLT VEL [floating point variable used to represent value of wind velocity in weather and wind planning log]

ENDSTRUCTURE;

[RUNWAY SURFACE CONDITIONS PLANNING LOG INFORMATION ORIGINAL FROM DATA BASE]

STRUCTURE SREGIN

GROUP TABLE (13) [up to 13 log entries]

  CHR TIME [4 bit character representation of time used in weather and wind planning log]
  CHR RWY [3 bit character representation of a runway in an entry in runway surface conditions planning log]
  CHR SURF [5 bit character field reserved for description of runway surface conditions in runway surface conditions planning log]
  CHR BRAK [5 bit character field reserved for description of runway braking conditions in runway surface conditions planning log]
CHR CLOSED [6 bit character field reserved for information on runway closures on runway surface conditions planning log]

CHR OPEN [6 bit character field reserved for information on runway closures on runway surface conditions planning log]

CHR REMARKS [27 bit character field reserved for free formatted comments used in runway surface conditions planning log]

ENDSTRUCTURE:

[RUNWAY SURFACE CONDITIONS PLANNING LOG DATA ORIGINAL FROM DATA BASE]

STRUCTURE CVTSGW

GROUP TABLE (13) [up to 13 log entries]

INT TIME [integer representing time used in runway surface conditions planning log]

ENDSTRUCTURE:
[DEMAND PLANNING LOG SCREEN INFORMATION ORIGINAL FROM DATA BASE]

STRUCTURE BEGIN

GROUP TABLE (0:23) [24 hours of demand data available on demand planning screen]

CHR GMT [4 bit character representation of time (hour) for demand information]
CHR TTLARR [3 bit character representation of total hourly arrival demand]
CHR TTLDEP [3 bit character representation of total hourly departure demand]
CHR KUBBS [3 bit character representation of hourly arrival demand at fix KUBBS]
CHR CGT [3 bit character representation of hourly arrival demand at fix CGT]
CHR VAINS [3 bit character representation of hourly arrival demand at fix VAINS]
CHR FARKH [3 bit character representation of hourly arrival demand at fix FARKH]
CHR NORTH [3 bit character representation of hourly departure demand at NORTH fix]
CHR EAST [3 bit character representation of hourly departure demand at EAST fix]
CHR SOUTH [3 bit character representation of hourly departure demand at SOUTH fix]
CHR WEST [3 bit character representation of hourly departure demand at WEST fix]

ENDSTRUCTURE;
[DEMAND PLANNING LOG DATA ORIGINAL FROM DATA BASE]

STRUCTURE CVTGCBN

GROUP TABLE (0:23) [24 hours of demand data available on demand planning screen]

INT GMT [integer value of time (hour) for demand information]

FLT TTLARR [floating point value of total hourly arrival demand]

FLT TTLDEP [floating point value of total hourly departure demand]

FLT KUBBS [floating point value of hourly arrival demand at fix KUBBS]

FLT CCT [floating point value of hourly arrival demand at fix CCT]

FLT VAINS [floating point value of hourly arrival demand at fix VAINS]

FLT FARMN [floating point value of hourly arrival demand at fix FARMN]

FLT NORTH [floating point value of hourly departure demand at NORTH fix]

FLT EAST [floating point value of hourly departure demand at EAST fix]

FLT SOUTH [floating point value of hourly departure demand at SOUTH fix]

FLT WEST [floating point value of hourly departure demand at WEST fix]
STRUCTURE PBEGIN
  GROUP PARAMETER [wind thresholds]
    GROUP ARR [pertaining to arrivals]
      CHR CRSS [4 bit character representation of cross wind component of wind thresholds]
      CHR TAIL [4 bit character representation of cross wind component of wind thresholds]
    GROUP DEF [pertaining to departures]
      CHR CRSS [4 bit character representation of cross wind component of wind thresholds]
      CHR TAIL [4 bit character representation of tail wind component of wind thresholds]
  ENDSTRUCTURE;

[PARAMETERS DATA ORIGINAL FROM DATA BASE]

STRUCTURE CVTPBGN
  GROUP ARR [wind thresholds for arrivals]
    FLT CRSS [crosswind component of wind thresholds]
    FLT TAIL [tailwind component of wind thresholds]
  GROUP DEF [wind thresholds for departures]
    FLT CRSS [crosswind component of wind thresholds]
    FLT TAIL [tailwind component of wind thresholds]
  ENDSTRUCTURE;
[DEPARTURE QUEUES INFORMATION ORIGINAL FROM DATA BASE]

STRUCTURE QBEGIN

CHR QL (4) [2 bit character representation of number of aircraft in departure queue]

ENDSTRUCTURE

[DEPARTURE QUEUES DATA ORIGINAL FROM DATA BASE]

INT CVTQNC (4) [integer representing length of departure queues]
2.2 High Level Processing

High Level Processing is described on pages 2-60 to 2-148.
TASK ASSISTANT_CHIEF_MAIN_PROGRAM

[This is assistant chief program's main procedure referred to as ASSISTANT_CHIEF_MAIN_PROGRAM, it controls entire program by calling several routines that take user into CNS]

BITS ESTATUS [8 bit string containing current program status initialized to '0111011'B or equivalent for PF1 program function key]

BITS OLD_STATUS [8 bit string containing the previous program status]

CHR TERM [character string (length 1) indicating program termination, ' ' - do not terminate, 'X' - terminate, initialized to blank]

INT I [integer indicating environment of operation, 1_current, 2_forecast, it is initialized to 1]

CALL GETFILE; [read permanent data files containing program's global parameters]

CALL INREAD; [initial access to database, read database]

REPEAT UNTIL (TERM EQ 'X'); [begin main loop, repeat until termination is signaled]

CALL CHOOSE;
  INOUT (RSTATUS, OLD_STATUS,1); [choose screen or function]

CALL TOLINK; [access central data base]

CALL READER; [read central data base]

CALL MERGE;
  INOUT (OLD STATUS); [merge all versions of central database]

CALL WRITER; [write new central database]

CALL TOTACH; [release central database]

CALL ASSIGN; [assign new values to some program variables]

CALL UPDATE; [compute rest of program variables]

ENDREPEAT;

END ASSISTANT_CHIEF_MAIN_PROGRAM;
ROUTINE GETFILE
[This routine reads permanent data files containing CMS global parameters]
Read RWYMIN from RWYMIN file;
CALL MILES;
    INOUT(RWYMIN); [convert RVR readings to miles]
Read CNFREQ from CNFREQ file;
Read CAPFILE from CAPACITY file;
Read FIXTRAV from TRAVEL file;
Read DEPMAT from DEPEND file;
Read OACDEM from OACDMND file;
END GETFILE;

ROUTINE MILES
    INOUT(RWYMIN); [this routine converts RVR readings to miles]
    LOOP; [J = 1 to 12] [for each runway]
        PERFORM RVR_TO_MILES_CONVERSION;
    ENDCLOOP;
END MILES;

PROCESS RVR_TO_MILES_CONVERSION
[This process converts data items from RVR to miles]
    RWYMIN(J).CATII.NONE.VIS = M(RWYMIN(J).CATII.NONE.VIS);
    RWYMIN(J).ILS.NONE.VIS = M(RWYMIN(J).ILS.NONE.VIS);
    RWYMIN(J).ILS.MN.VIS = M(RWYMIN(J).ILS.MN.VIS);
END RVR_TO_MILES_CONVERSION;
ROUTINE INREAD

[This routine accesses base and reads data base for first time]

CALL TOLINK; [access central data base]

PERFORM READ_DATA_BASE;

CALL TODTACH; [release central data base]

PERFORM SET.ORIGINAL_PROGRAM_VARIABLES;

PERFORM SET.CMS_PROGRAM_VARIABLES;

END INREAD;

PROCESS READ_DATA_BASE

[This process reads data base]

Open file STARTUP;

Read STORED;
Read MNOW, CNOW, QNOW, CVTQNOW;
Read PNOW, CVTNOW;
Read ANOW;
Read CVTANOW;
Read RNOW;
Read DNOW;
Read CVTCDNOW;
Read KNOW;
Read CVTGNOW;
Read NOW;
Read CVTWNOW;
Read SNOW;
Read CVTSNOW;
Read GNOW;
Read CVTGNOW;

Close file STARTUP; [each read corresponds to a read statement in program]

END READ_DATA_BASE;
PROCESS SET ORIGINAL PROGRAM VARIABLES

[This process sets original program variables from current program variables]

MBEGIN = WNOW;
PBEGIN = PNOW;
CUTPRCN = CVTPNOW;
ABEGIN = ANOW;
CUTARGN = CVTAMOW;
EBEGIN = ENOW;
DBEGIN = DNOW;
CUTORGN = CVTOWNOW;
CBEGIN = CNOW;
CBEGIN = CNOW;
CUTQCN = CVTQNOW;
QBEGIN = QNOW;
CUTQCN = CVTQNOW;
WBEGIN = WNOW;
CUTWGEN = CVTWNOW;
SBEGIN = SNOW;
CUTSBGN = CVTSNOW;
GBEGIN = GNOW;
CUTSBGN = CVTSNOW;

END SET ORIGINAL PROGRAM VARIABLES;
PROCESS SET_CMS_PROGRAM_VARIABLES

[This process sets CMS program variables from current and original program variables]

MIDFLG = MBEGIN;
PARAM = PBEGIN, BY NAME;
CNVTTERM = CVTPRON, BY NAME;  [BY NAME OPTION chooses only those variables each structure that have similar names]

APTSTAT = ABEGIN, BY NAME;
CNVTAPT = CVTPRON, BY NAME;
RWTQP = RBEGIN, BY NAME;
DEMAND = DBEGIN, BY NAME;
CNVTDEM = CVTPRON, BY NAME;
CONFIND = CBEGIN;
EQPLOC = EBEGIN, BY NAME;
CNVTEQP = CVTPRON;
QUELEN = QBEGIN, BY NAME;
CNVTQLN = CVTPRON;
WXLOG = WBEGIN, BY NAME;
CNVTIWX = CVTPRON;
SJEFLOG = SBEGIN, BY NAME;
CNVTSRF = CVTPRON;
OAGLOG = GBEGIN, BY NAME;
CNVTOAG = CVTPRON, BY NAME;

END SET_CMS_PROGRAM_VARIABLES;
ROUTINE CHOOSE
(This routine checks value of current program status variable and chooses function or screen desired by user)

INOUT (RSTATUS, OLD_STATUS, I);

IF RSTATUS EQ PF10
THEN RSTATUS = OLD_STATUS;
ELSE OLD_STATUS = RSTATUS;
PERFORM SCREEN SELECTION;
END CHOOSE;

PROCESS SCREEN SELECTION
(This process selects function or screen)

IF RSTATUS EQ PF1
THEN CALL NSTAT;
IN (QSTAT, APSTAT(1), INFORM(1), CHFORM(QCONFIND(1)), QCONFIND(1), EQPLOG, EQUTERP, WXLOG, CRVTX, SURLOG, OUTERF);

INOUT (OLDNUM, RSTATUS); [O'Hare status screen]
ELSEIF RSTATUS EQ PF2;
THEN CALL LOGS;
INOUT (RSTATUS); [log selection screen]
ELSEIF RSTATUS EQ PF3;
THEN CALL WLOG;
INOUT (WXLOG, CRVTX, RSTATUS); [wind & weather planning log screen]
ELSEIF RSTATUS EQ PF14;
    THEN CALL SLOG;
    INOUT (SURFLOG, CNVTSP, RSTATUS);  [airport planning log screen]
ELSEIF RSTATUS EQ PF15
    THEN CALL ELOG;
    INOUT (EQPLOG, CNVTSP, RSTATUS);  [equipment planning log screen]
ELSEIF RSTATUS EQ PF16;
    THEN CALL GLOG;
    IN (OAGDEM)
    INOUT (OAGLOG, CNVTAG, RSTATUS);  [demand planning log screen]
ELSEIF RSTATUS EQ PF3;
    THEN CALL ARPT;
    IN (CNVTPEM)
    INOUT (ARTSTAT, MIDFLAG, CNVTAPT, RSTATUS, I);  [airport status screen (current, forecast)]
ELSEIF RSTATUS EQ PF4
    THEN CALL EWT;
    INOUT (KWTSEQ, RSTATUS, I);  [equipment status screen (current, forecast)]
ELSEIF RSTATUS EQ PF5;
    THEN CALL DWT;
IN (CNYTLMG)

INPUT (DEMAND, CNYTLMG, RSTATUS, 1); [demand profile screen]

ELSEIF RSTATUS EQ PP6
  THEN CALL ORDER;
    IN (PRCARR, CNYQMS, KNTKRP, MIDFLAG, COMPLST, COMPLIED);
    INPUT (RSTATUS, 1); [ordered list of configurations screen (current, forecast)]

ELSEIF RSTATUS EQ PP7
  THEN CALL QUEUE;
    INPUT (QUELEM, CNYTLMG, RSTATUS); [current queue length screen]

ELSEIF RSTATUS EQ PP8
  THEN CALL TSTUT;
    IN (PRCARR, CYNQMRG, CNYTLMG, DEPWRT, PRTTRNP, TRANLIST, INFORM, COMPLST, CNYTH, ELIGBST, CNYTLMG, (QUELEM));
    INPUT (RSTATUS); [ordered list of transitions screen]

ELSEIF RSTATUS EQ PP9
  THEN CALL CHNG;
IN (CONFIG, CNPGIQ, CONFIND, PRECAR, INFORM, MIDFLAG, PNTUPQ); INOUT (RSTATUS, I); [configuration information screen (current, forecast)]

ELSEIF RSTATUS EQ FF10

THEN OLD_STATUS = PF11;

ELSE CALL MENUW~m;

INOUT (PARAM, CNTPRM, RSTATUS);

OUT (TERM); [menu screen or parameter screen]

END SCREEN SELECTION;
ROUTINE TOLINK
[This routine establishes a link to data base]

INT IRETCD [contains return code]
CALL CONOD; [try to link to data base]
IN (' CP LINK K115768 191 197 M CNS #');
OUT (IRETCD); [this system routine issues a CP or CNS message]
IF (IROTCD GT 106) AND (IRETCD LT 120):
THEN STOP; [stop program if linkage error has occurred]
[Wait if another user is linked]
REPEAT WHILE (IRETCD NE 0);
   CALL CONOD;
   IN (' CP SLEEP 5 SEC#');
   OUT (IRETCD);
   CALL CONOD;
   IN (' CP LINK K115768 191 197 M CNS #');
   OUT (IROTCD);
ENDREPEAT;
CALL CONOD; [once linked, access]
IN (' ACC 197 N #');
OUT (IRETCD);
END TOLINK;
ROUTINE TODTACH

[This routine detaches user from data base]

INT IRETCO [contains return code]
CALL COMMD; [detach and release data base]
IN (' CP DET 197 #');
OUT (IRETCO);

CALL COMMD;
IN (' REL 197 #');
OUT (IRETCO);

END TODTACH;

ROUTINE READER

[This routine reads data base into current global variables]

PERFORM READ_DATA_BASE;

END READER;
ROUTINE WRITER
[This routine writes most current version of data on to data base]
Open STARTUP file;
Write STORED;
Write (MNOW, CNOW, QNOW, CVTQNOW);
Write (PNOW, CVTNOW);
Write ANOW;
Write CVTANOW;
Write ENOW;
Write DNOW;
Write CVTDNOW;
Write ENOW;
Write CVTENOW;
Write WNOW;
Write CVTWNOW;
Write SNOW;
Write CVTSNOW;
Write CHOW;
Write CMVTNOW;
Close STARTUP file;
END WRITER;
ROUTINE MERGE

[This routine merges and reconciles all different versions of data base and prepares most current version for data base. A number of routines that perform global updates are called from this routine]

DATA_STORED = 'DATA STORED AT ';

IF (OLD_STATUS EQ 'FPI') AND (SUBSTR(ONSTAT.MSG, 1, 15) NE DATA_STORED)
    THEN
        STORED.ONSTATUS = GMT;  [time update]
    ELSEIF (OLD_STATUS EQ 'PF13') AND (SUBSTR(WXLOG.MSG, 1, 15) NE DATA_STORED)
        THENCALL WGLOBAL;
            INOUT (WXLOG, CNVTWX, WREGIN, CVTWHM, WNOW, CVTRNOW);  [This routine reconciles different versions of wind and weather planning log information]
            STORED.APLOGI = GMT;  [time updates]
            STORED.ONSTATUS = STORED.APLOGI;
    ELSEIF (OLD_STATUS EQ 'PF14') AND (SUBSTR(SRFLOG.MSG, 1, 15) NE DATA_STORED)
        THENCALL SGLOBAL;
            INOUT (SRFLOG, CNVTSRF, SREGIN, CVTBSGN, SNOW, CVTSNOW);  [This routine reconciles different versions of airport planning log information]
            STORED.APLOG2 = GMT;  [time updates]
            STORED.ONSTATUS = STORED.APLOG2;
    ELSEIF (OLD_STATUS EQ 'PF15') AND (SUBSTR(RQPLG.MSG, 1, 15) NE DATA_STORED)
        THENCALL EGLLOBAL;
            INOUT (RQPLG, CNVTQRP, RREGIN, CVTBSGN, ENOW, CVTERNOW);  [This routine reconciles different versions of equipment planning log information]
STORED.ENYLOG = GMT;
STORED.OBSTATUS = STORED.ENYLOG;  [time updates]
ELSEIF (OLD_STATUS EQ PF16) AND (SUBSTR(OALOGMSG, 1, 15) NE DATA_STORED)
THENCALL OGLOBAL;
INOUT (OALOG, CNVTOAL, GBEGIM, CVTOLG, CVTEN, CVTEN);
[This routine reconciles different versions of demand planning log information]
STORED.DMDLOG = GMT;  [time update]
ELSEIF OLD_STATUS EQ PF3
THEN
LOOP;  [J = 1 to 2]
IF SUBSTR(APTSTAT(J).MSG, 1, 15) NE DATA_STORED
THENCALL AGLOBAL;
INOUT (APTSTAT(J), CNVTAPP(J), MIDFLAG(J),
APRENEW(J), CVTANOM(J), MBRENEW(J), ANW(J),
APANOM(J), MNOW(J));
[This routine reconciles different versions of airport status information]
STORED.AIRPORT(J) = GMT;  [time updates]
STORED.OLIST(J) = STORED.AIRPORT(J);
STORED.OCONF(J) = STORED.AIRPORT(J);
STORED.TRANLIST = STORED.AIRPORT(J);
ENDIFLOOP;
ELSEIF OLD_STATUS EQ PF4
THEN
LOOP: \( j = 1 \) to 2

IF SUBSTR(ENTRY(j).MSG, 1, 15) NE DATA_STORED

THEN CALL DGLOBAL;

INOUT (ENTRY(j), RBEGIN(j), RNOW(j));

[This routine reconciles different versions of equipment status equipment]

STORED-RUNWAY(j) = GNT; [time updates]
STORED-ALFORT(j) = STORED-RUNWAY(j);
STORED-GLIST(j) = STORED-RUNWAY(j);
STORED-CMP(j) = STORED-RUNWAY(j);
STORED-TRAMLIST = STORED-RUNWAY(j);

ENDLOOP;

ELSEIF OLD_STATUS EQ PF5

THEN

LOOP: \( j = 1 \) to 2

IF SUBSTR(DEMAND(j).MSG, 1, 15) NE DATA_STORED

THEN CALL DGLOBAL;

INOUT (DEMAND(j), CWTHER(j),
        BASE(j), CYTHER(j), DNS(j),
        CYTHER(j));

[This routine reconciles different versions of demand profile information]

STORED-DEMAND(j) = GNT; [time updates]
STORED-GLIST(j) = STORED-DEMAND(j);
STORED-DEMAND(j) = STORED-DEMAND(j);
STORED-TRAMLIST = STORED-DEMAND(j);

ENDLOOP;
ELSEIF OLD_STATUS EQ PF6 THEN

IF SUBST(CONFLST(1).MSG, 1, 15) NE DATA_STORED THENCALL CGLOBAL;
    INPUT (CONFIND(1), CBEGIN(1), CWON(1),
            CWON, CVTONOM, $ONE);
    [This routine reconciles different versions of operating configuration information]  
    STORED.OLIST(1) = GMT; [Time updates]
    STORED.CNS(1) = STORED.OLIST(1);
    STORED.QLENGTH = STORED.OLIST(1);
    STORED.TRANLIST = STORED.OLIST(1);

IF SUBST(CONFLST(2).MSG, 1, 15) NE DATA_STORED THENCALL CGLOBAL;
    INPUT (CONFIND(2), CBEGIN(2), CWON(2),
            CWON, CVTONOM, $TWO);
    STORED.OLIST(2) = GMT; [Time updates]
    STORED.CNS(2) = STORED.OLIST(2);
    STORED.QLENGTH = STORED.OLIST(2);

ELSEIF (OLD_STATUS EQ PF7) AND SUBST(QULEN.MSG, 1, 15) NE DATA_STORED THENCALL CGLOBAL;
    INPUT (QULEN, CVTONOM, QBEGIN, QWON, CVTONOM);
    [This routine reconciles different versions of current departure queue information]  
    STORED.QLENGTH = GMT; [Time updates]
    STORED.TRANLIST = STORED.QLENGTH;
ELSE IF OLD_STATUS EQ PPS
    THEN STORED.TRANLIST = GMT; [Time update]
ELSE IF OLD_STATUS EQ PPF
    THEN
        IF SUBSTR(CONFIG(1).MSG, 1, 15) NE DATA STORED
            THENCALL GGLOBAL;
            INPUT (CONFIG(1), CBEGIN(1),
                    CROW(1), QNOW, CVTQNOW, $ONE);
            STORED.CONF(1) = GMT; [time updates]
            STORED.OLIST(1) = STORED.CONF(1);
            STORED.QLength = STORED.CONF(1);
            STORED.TRANLIST = STORED.CONF(1);
            IF SUBSTR(CONFIG(2).MSG, 1, 15) NE DATA STORED
                THENCALL GGLOBAL;
                INPUT (CONFIG(2),
                        CBEGIN(2), CROW(2),
                        QNOW, CVTQNOW, $TWO);
                STORED.CONF(2) = GMT [time updates]
                STORED.OLIST(2) = STORED.CONF(2);
    ELSE IF (OLD_STATUS EQ PPL) AND (SUBSTR (PARAM.MSG, 1, 15) NE DATA STORED
            THEN
THENCALL PGLOBAL;
THENCALL PGLOBAL;
THEN CALL (PARAM, CNT2PN, PREDIN, CNT2CH, PREDON, CNT2NOW);
THENCALL PGLOBAL;
THEN CALL (PARAM, CNT2PN, PREDIN, CNT2CH, PREDON, CNT2NOW);
[This routine reconciles different versions of parameters information]

STORED.PARNOPT = GMT; [time updates]
STORED.AIRPORT(1) = STORED.
PAIRNOPT;
STORED.AIRPORT(2) = STORED.
PAIRNOPT;
STORED.OLIST(1) = STORED.
PAIRNOPT;
STORED.OLIST(2) = STORED.
PAIRNOPT;
STORED.CLIST(1) = STORED.
PAIRNOPT;
STORED.CLIST(2) = STORED.
PAIRNOPT;
STORED.TLIST = STORED.
PAIRNOPT;
STORED.T2LIST = STORED.
PAIRNOPT;
STORED.T3LIST = STORED.
PAIRNOPT;

END MERGE;
ROUTINE ASSIGN

[This routine produces two copies of global variables. One to be used in lower level programs and other to serve as original version until next update cycle]

STORED_DATA = 'DATA STORED AT';
PERFORM SET ORIGINAL_PROGRAM_VARIABLES;
PERFORM SET CMS_PROGRAM_VARIABLES;
PERFORM STORED_TIME_SET_UP;
END ASSIGN;

PROCESS STORED_TIME_SET_UP

[This process sets up message portion of global variables with stored times]

APPTAT(1).MSG = DATA_STORED CONCATENATE STORED-AIRPORT(1);
APPTAT(2).MSG = DATA_STORED CONCATENATE STORED-AIRPORT(2);
BENEFIP(1).MSG = DATA_STORED CONCATENATE STORED-RUNWAY(1);
BENEFIP(2).MSG = DATA_STORED CONCATENATE STORED-RUNWAY(2);
DEMAND(1).MSG = DATA_STORED CONCATENATE STORED-Demand(1);
DEMAND(2).MSG = DATA_STORED CONCATENATE STORED-Demand(2);
COMPLIST(1).MSG = DATA_STORED CONCATENATE STORED-OList(1);
COMPLIST(2).MSG = DATA_STORED CONCATENATE STORED-OList(2);
EQPLOG.MSG = DATA_STORED CONCATENATE STORED-EQPLIST;
QUELEN.MSG = DATA_STORED CONCATENATE STORED-QUELEN;
ONSTAT.MSG = DATA_STORED CONCATENATE STORED-ONSTATUS;
RUNLOG.MSG = DATA_STORED CONCATENATE STORED-RUNLOG;
SURFLOG.MSG = DATA_STORED CONCATENATE STORED-SURFLOG;
OAGLOG.MSG = DATA_STORED CONCATENATE STORED-OAGLOG;

END STORED_TIME_SET_UP;
ROUTINE UPDATE

(This routine performs a number of inner model computations needed during each update cycle, e.g., weather minima, crosswind and tailwind components of wind, runway closures, configuration eligibility, etc.)

LOOP: \[ \text{J = 1 to 2} \]

CALL MINIMA; \([\text{Compute minima based on equipment status}]\)
\[ \text{IN (ENTRYP(J), ENTRYN)}; \]
\[ \text{INOUT (APTSTAT(J), CVTAPT(J))}; \]

CALL WIND; \([\text{compute crosswind & tailwind components of wind for each runway}]\)
\[ \text{INOUT (APTSTAT(J), CVTAPT(J))}; \]

CALL CLOSING; \([\text{determine runway closures}]\)
\[ \text{IN (CVTMIN)}; \]
\[ \text{INOUT (APTSTAT(J), CVTAPT(J))}; \]

CALL FILES; \([\text{determine capacity file number for each configuration and set CVTMIN variable to indicate VFR(1) or IFR(2)}]\)
\[ \text{IN (APTSTAT(J), CVTAPT(J))}; \]
\[ \text{INOUT (FILEMIN(J), CVTHY(J))}; \]

CALL ELG; \([\text{determine eligibility of configurations}]\)
\[ \text{IN (CONFUG, APTSTAT(J), CVTAPT(J), ENTRYP(J))}; \]
\[ \text{INOUT (ELGIBILITY(J))}; \]

CALL PERCENT; \([\text{compute north and south demands based on fix-to-runway assignments}]\)
\[ \text{IN (CVTHDEH(J), CONFUG)}; \]
\[ \text{INOUT (PCTHARR(J))}; \]
CALL CAPSAT; [compute capacity and balance demand for each configuration]

IN (PRGARR(J), CNFRQ, CAPFLE, FILENUM(J), ELGRLTY(J));

INOUT (INFORM(J));

ENDLOOP;

CALL QFIX; [update departure runways for current configuration]

IN (CNFRQ(CONFIND(J)));

INOUT (QUELEN);

END UPDATE;
ROUTINE AGLOBAL

INPUT (APTSTAT(1), CVTAOFF(1), MIDFLAG(1), AREGIN(1), CVTABOM(1), MBEGIN(1), AMON(1), CVTAMON(1), NMON(1));

[This routine reconciles different versions of airport status information]

LOOP: [K = 1 to 12] [for each runway]

IF APTSTAT(I).RUNWAY(K).TOWER.ARR NE AREGIN(1).RUNWAY(K).TOWER.ARR
THEN AMON(I).RUNWAY(K).TOWER.ARR = APTSTAT(I).RUNWAY(K).TOWER.ARR;
ELSE APTSTAT(I).RUNWAY(K).TOWER.ARR = AMON(I).RUNWAY(K).TOWER.ARR;

IF APTSTAT(I).RUNWAY(K).TOWER.DEP NE AREGIN(1).RUNWAY(K).TOWER.DEP
THEN AMON(I).RUNWAY(K).TOWER.DEP = APTSTAT(I).RUNWAY(K).TOWER.DEP;
ELSE APTSTAT(I).RUNWAY(K).TOWER.DEP = AMON(I).RUNWAY(K).TOWER.DEP;

IF APTSTAT(I).RUNWAY(K).SURF NE AREGIN(1).RUNWAY(K).SURF;
THEN AMON(I).RUNWAY(K).SURF = APTSTAT(I).RUNWAY(K).SURF;
ELSE APTSTAT(I).RUNWAY(K).SURF = AMON(I).RUNWAY(K).SURF;

IF APTSTAT(I).RUNWAY(K).NRK NE AREGIN(1).RUNWAY(K).NRK
THEN AMON(I).RUNWAY(K).NRK = APTSTAT(I).RUNWAY(K).NRK;
ELSE APTSTAT(I).RUNWAY(K).NRK = AMON(I).RUNWAY(K).NRK;

ENDLOOP;

IF CVTAOFF(I).WX.CEIL NE CVTAMON(1).WX.CEIL
THEN CVTAMON(1).WX.CEIL = CVTAOFF(I).WX.CEIL;
AMON(I).WX.CEIL = APTSTAT(I).WX.CEIL;
ELSE CVTAOFF(I).WX.CEIL = CVTAMON(1).WX.CEIL;
APTSTAT(I).WX.CEIL = AMON(I).WX.CEIL;
IF CNYTAPT(I).WX.VIS NE CVTANW(I).WX.VIS THEN
    CVTANW(I).WX.VIS = CNYTAPT(I).WX.VIS;
    ANOW(I).WX.VIS = APTSTAT(I).WX.VIS;
ELSE
    CNYTAPT(I).WX.VIS = CVTANW(I).WX.VIS;
    APTSTAT(I).WX.VIS = ANOW(I).WX.VIS;
IF CNYTAPT(I).WIND.DIR NE CVTANW(I).WIND.DIR THEN
    CVTANW(I).WIND.DIR = CNYTAPT(I).WIND.DIR;
    ANOW(I).WIND.DIR = APTSTAT(I).WIND.DIR;
ELSE
    CNYTAPT(I).WIND.DIR = CVTANW(I).WIND.DIR;
    APTSTAT(I).WIND.DIR = ANOW(I).WIND.DIR;
IF CNYTAPT(I).WIND.VEL NE CVTANW(I).WIND.VEL THEN
    CVTANW(I).WIND.VEL = CNYTAPT(I).WIND.VEL;
    ANOW(I).WIND.VEL = APTSTAT(I).WIND.VEL;
ELSE
    CNYTAPT(I).WIND.VEL = CVTANW(I).WIND.VEL;
    APTSTAT(I).WIND.VEL = ANOW(I).WIND.VEL;
IF MIDFLAG(I) NE MRBEGIN(I) THEN
    MNOW(I) = MIDFLAG(I);
ELSE
    MINFLAG(I) = MNOW(I);
END AGLOBAL;
ROUTINE RGLOBAL

INPUT (RWTEQP(I), RBEGIN(I), RNOW(I));
[This routine reconciles different versions of equipment status information]

LOOP; [K = 1 to 12] [for each runway]

IF RWTEQP(I).RUNWAY(K).CATII NE RBEGIN(I).RUNWAY(K).CATII
    THEN RNOW(I).RUNWAY(K).CATII = RWTEQP(I).RUNWAY(K).CATII;
    ELSE RWTEQP(I).RUNWAY(K).CATII = RNOW(I).RUNWAY(K).CATII;

IF RWTEQP(I).RUNWAY(K).LOC NE RBEGIN(I).RUNWAY(K).LOC
    THEN RNOW(I).RUNWAY(K).LOC = RWTEQP(I).RUNWAY(K).LOC;
    ELSE RWTEQP(I).RUNWAY(K).LOC = RNOW(I).RUNWAY(K).LOC;

IF RWTEQP(I).RUNWAY(K).GS NE RBEGIN(I).RUNWAY(K).GS
    THEN RNOW(I).RUNWAY(K).GS = RWTEQP(I).RUNWAY(K).GS;
    ELSE RWTEQP(I).RUNWAY(K).GS = RNOW(I).RUNWAY(K).GS;

IF RWTEQP(I).RUNWAY(K).ON NE RBEGIN(I).RUNWAY(K).ON
    THEN RNOW(I).RUNWAY(K).ON = RWTEQP(I).RUNWAY(K).ON;
    ELSE RWTEQP(I).RUNWAY(K).ON = RNOW(I).RUNWAY(K).ON;

IF RWTEQP(I).RUNWAY(K).IN NE RBEGIN(I).RUNWAY(K).IN
    THEN RNOW(I).RUNWAY(K).IN = RWTEQP(I).RUNWAY(K).IN;
    ELSE RWTEQP(I).RUNWAY(K).IN = RNOW(I).RUNWAY(K).IN;
IF RWYEQP(I).RUNWAY(K).RAIL NE RBEGIN(I).RUNWAY(K).RAIL
THEN RNOW(I).RUNWAY(K).RAIL = RWYEQP(I).RUNWAY(K).RAIL;
ELSE RWYEQP(I).RUNWAY(K).RAIL = RNOW(I).RUNWAY(K).RAIL;

IF RWYEQP(I).RUNWAY(K).ALS NE RBEGIN(I).RUNWAY(K).ALS
THEN RNOW(I).RUNWAY(K).ALS = RWYEQP(I).RUNWAY(K).ALS;
ELSE RWYEQP(I).RUNWAY(K).ALS = RNOW(I).RUNWAY(K).ALS;

IF RWYEQP(I).RUNWAY(K).EVR NE RBEGIN(I).RUNWAY(K).EVR
THEN RNOW(I).RUNWAY(K).EVR = RWYEQP(I).RUNWAY(K).EVR;
ELSE RWYEQP(I).RUNWAY(K).EVR = RNOW(I).RUNWAY(K).EVR;

IF RWYEQP(I).RUNWAY(K).HIEL NE RBEGIN(I).RUNWAY(K).HIEL
THEN RNOW(I).RUNWAY(K).HIEL = RWYEQP(I).RUNWAY(K).HIEL;
ELSE RWYEQP(I).RUNWAY(K).HIEL = RNOW(I).RUNWAY(K).HIEL;

IF RWYEQP(I).RUNWAY(K).CL NE RBEGIN(I).RUNWAY(K).CL
THEN RNOW(I).RUNWAY(K).CL = RWYEQP(I).RUNWAY(K).CL;
ELSE RWYEQP(I).RUNWAY(K).CL = RNOW(I).RUNWAY(K).CL;

IF RWYEQP(I).RUNWAY(K).TDZ NE RBEGIN(I).RUNWAY(K).TDZ
THEN RNOW(I).RUNWAY(K).TDZ = RWYEQP(I).RUNWAY(K).TDZ;
ELSE RWYEQP(I).RUNWAY(K).TDZ = RNOW(I).RUNWAY(K).TDZ;

IF RWYEQP(I).RUNWAY(K).NDB_VOR = RBEGIN(I).RUNWAY(K).NDB_VOR
THEN RNOW(I).RUNWAY(K).NDB_VOR = RWYEQP(I).RUNWAY(K).NDB_VOR;
ELSE RWYEQP(I).RUNWAY(K).NDB_VOR = RNOW(I).RUNWAY(K).NDB_VOR;

ENDLOOP;

END GLOBAL;
ROUTINE EGLOBAL

INPUT (EQPLOG, CNTEQP, ESBEGIN, CVTEQM, ENOW, CVTEQM);

[This routine reconciles different versions of equipment planning log]

LOOP: [J = 13 to 15] [from 13th to 15th message for AC]

  ENOW.TABLE(J).BUY = EQPLOG.TABLE(J).BUY;
  ENOW.TABLE(J).EQUIPMENT = EQPLOG.TABLE(J).EQUIPMENT;
  ENOW.TABLE(J).OTS = EQPLOG.TABLE(J).OTS;
  ENOW.TABLE(J).KTS = EQPLOG.TABLE(J).KTS;
  ENOW.TABLE(J).REMARKS = EQPLOG.TABLE(J).REMARKS;
  CVTEQM.TABLE(J).OTS = CNTEQM.TABLE(J).OTS;
  CVTEQM.TABLE(J).KTS = CNTEQM.TABLE(J).KTS;

ENLOOP;

END EGLOBAL;
ROUTINE DGLOBAL

INPUT (DEMAND(I), CVTDDM(I), DBEGIN(I), CVTDGNI(I), DNOW(I), CVTDMO(I));
[This routine reconciles different versions of demand profile information]

IF CVTDDM(I).ARR.TOTAL NE CVTDGNI(I).ARR.TOTAL
    THEN
        DNOW(I).ARR.TOTAL = DEMAND(I).ARR.TOTAL;
        CVTDMO(I).ARR.TOTAL = CVTDDM(I).ARR.TOTAL;
    ELSE
        DEMAND(I).ARR.TOTAL = DNOW(I).ARR.TOTAL;
        CVTDDM(I).ARR.TOTAL = CVTDGNI(I).ARR.TOTAL;

IF CVTDDM(I).ARR.KUSS NE CVTDGNI(I).ARR.KUSS
    THEN
        DNOW(I).ARR.KUSS = DEMAND(I).ARR.KUSS;
        CVTDGNI(I).ARR.KUSS = CVTDDM(I).ARR.KUSS;
    ELSE
        DEMAND(I).ARR.KUSS = DNOW(I).ARR.KUSS;
        CVTDGNI(I).ARR.KUSS = CVTDGNI(I).ARR.KUSS

IF CVTDDM(I).ARR.PLANT NE CVTDGNI(I).ARR.PLANT
    THEN
        DNOW(I).ARR.PLANT = DEMAND(I).ARR.PLANT;
        CVTDGNI(I).ARR.PLANT = CVTDDM(I).ARR.PLANT;
    ELSE
        DEMAND(I).ARR.PLANT = DNOW(I).ARR.PLANT;
        CVTDGNI(I).ARR.PLANT = CVTDGNI(I).ARR.PLANT;

IF CVTDDM(I).ARR.CGT NE CVTDGNI(I).ARR.CGT
    THEN
        DNOW(I).ARR.CGT = DEMAND(I).ARR.CGT;
        CVTDGNI(I).ARR.CGT = CVTDDM(I).ARR.CGT;
    ELSE
        DEMAND(I).ARR.CGT = DNOW(I).ARR.CGT;
        CVTDGNI(I).ARR.CGT = CVTDGNI(I).ARR.CGT;
ELSE
  DEMAND(1).ARR.CPT = DNOW(1).ARR.CPT
  CVTDDEM(1).ARR.CPT = CVTDNOW(1).ARR.CPT
IF CVTDDEM(1).ARR.VAINS NE CVTDGDM(1).ARR.VAINS
  THEN
    DNOW(1).ARR.VAINS = DEMAND(1).ARR.VAINS;
    CVTDNOW(1).ARR.VAINS = CVTDDEM(1).ARR.VAINS;
ELSE
  DEMAND(1).ARR.VAINS = DNOW(1).ARR.VAINS;
  CVTDDEM(1).ARR.VAINS = CVTDNOW(1).ARR.VAINS;
IF CVTDDEM(1).ARR.FARM NE CVTDGDM(1).ARR.FARM
  THEN
    DNOW(1).ARR.FARM = DEMAND(1).ARR.FARM;
    CVTDNOW(1).ARR.FARM = CVTDDEM(1).ARR.FARM;
ELSE
  DEMAND(1).ARR.FARM = DNOW(1).ARR.FARM;
  CVTDDEM(1).ARR.FARM = CVTDNOW(1).ARR.FARM;
IF CVTDDEM(1).ARR.MKE_A NE CVTDGDM(1).ARR.MKE_A
  THEN
    DNOW(1).ARR.MKE_A = DEMAND(1).ARR.MKE_A;
    CVTDNOW(1).ARR.MKE_A = CVTDDEM(1).ARR.MKE_A;
ELSE
  DEMAND(1).ARR.MKE_A = DNOW(1).ARR.MKE_A;
  CVTDDEM(1).ARR.MKE_A = CVTDNOW(1).ARR.MKE_A;
IF CVTDDEM(1).DEF.TOTAL NE CVTDGDM(1).DEF.TOTAL
  THEN
    DNOW(1).DEF.TOTAL = DEMAND(1).DEF.TOTAL;
    CVTDNOW(1).DEF.TOTAL = CVTDDEM(1).DEF.TOTAL;
ELSE

DEMAND(1).DEP.TOTAL = DNW(1).DEP.TOTAL;
CVTDEN(1).DEP.TOTAL = CVTDEN(1).DEP.TOTAL;

IF CVTDEN(1).DEP.NORTH NE CVTDGN(1).DEP.NORTH
THEN

DNW(1).DEP.NORTH = DEMAND(1).DEP.NORTH;
CVTDEN(1).DEP.NORTH = CVTDEN(1).DEP.NORTH;
ELSE

DEMAND(1).DEP.NORTH = DNW(1).DEP.NORTH;
CVTDEN(1).DEP.NORTH = CVTDEN(1).DEP.NORTH;

IF CVTDEN(1).DEP.EAST NE CVTDGN(1).DEP.EAST
THEN

DNW(1).DEP.EAST = DEMAND(1).DEP.EAST;
CVTDEN(1).DEP.EAST = CVTDEN(1).DEP.EAST;
ELSE

DEMAND(1).DEP.EAST = DNW(1).DEP.EAST;
CVTDEN(1).DEP.EAST = CVTDEN(1).DEP.EAST;

IF CVTDEN(1).DEP.SOUTH NE CVTDGN(1).DEP.SOUTH
THEN

DNW(1).DEP.SOUTH = DEMAND(1).DEP.SOUTH;
CVTDEN(1).DEP.SOUTH = CVTDEN(1).DEP.SOUTH;
ELSE

DEMAND(1).DEP.SOUTH = DNW(1).DEP.SOUTH;
CVTDEN(1).DEP.SOUTH = CVTDEN(1).DEP.SOUTH;

IF CVTDEN(1).DEP.WEST NE CVTDGN(1).DEP.WEST
THEN

DNW(1).DEP.WEST = DEMAND(1).DEP.WEST;
CVTDEN(1).DEP.WEST = CVTDEN(1).DEP.WEST;
ELSE
  \text{DEMAND}(i).\text{DEP.WEST} = \text{DMO}(i).\text{DEP.WEST};
  \text{CNYTD}(i).\text{DEP.WEST} = \text{CYTDMO}(i).\text{DEP.WEST};

IF
  \text{CNYTD}(i).\text{DEP.WKE.D} \neq \text{CYTDMO}(i).\text{DEP.WKE.D}
THEN
  \text{DMO}(i).\text{DEP.WKE.D} = \text{DEMAND}(i).\text{DEP.WKE.D};
  \text{CYTDMO}(i).\text{DEP.WKE.D} = \text{CNYTD}(i).\text{DEP.WKE.D};
ELSE
  \text{DEMAND}(i).\text{DEP.WKE.D} = \text{DMO}(i).\text{DEP.WKE.D};
  \text{CNYTD}(i).\text{DEP.WKE.D} = \text{CYTDMO}(i).\text{DEP.WKE.D};

END\ DGLOBAL;
ROUTINE COGLOBAL

INOUT (CONFIND(I), CBEGIN(I), CHOW(I), QNOW, CVTQNOW, K);
[This routine reconciles different versions of operating configuration information]

IF CONFIND(I) NE CBEGIN(I)
THEN
CHOW(I) = CONFIND(I);
IF K EQ 1
THEN
LOOP; [J = 1 to 4]
CVTQNOW(J) = 0;
QNOW.QE(J) = '0';
ENDLOOP;
ELSE
CONFIND(I) = CHOW(I);
ENDIF

END COGLOBAL;

ROUTINE PGLOBAL

INOUT (PARAM, CHYTMN, PRBEGIN, CVTPGM, FNOW, CVTPNOW);
[This routine reconciles different versions of parameter information]

IF CHYTMN.ARR.CRSS NE CVTPGM.ARR.CRSS
THEN
FNOW.PARAMETER.ARR.CRSS = PARAM.PARAMETER.ARR.CRSS;
CVTPNOW.ARR.CRSS = CHYTMN.ARR.CRSS;
ENDIF
ELSE
  PARAM.PARAMETER.ARR.CRSS = PHON.PARAMETER.ARR.CRSS;
  CVTPHN.ARR.CRSS = CVTPHN.ARR.CRSS;
IF CNTPMN.ARR.TAIL NE CVTPGN.ARR.TAIL
  THEN
    PHON.PARAMETER.ARR.TAIL = PHON.PARAMETER.ARR.TAIL;
    CVTPHN.ARR.TAIL = CVTPMN.ARR.TAIL;
  ELSE
    PARAM.PARAMETER.ARR.TAIL = PHON.PARAMETER.ARR.TAIL;
    CVTPMN.ARR.TAIL = CVTPMN.ARR.TAIL;
IF CNTPMN.DEP.CRSS NE CVTPGN.DEP.CRSS
  THEN
    PHON.PARAMETER.DEP.CRSS = PARAM.PARAMETER.DEP.CRSS;
    CVTPHN.DEP.CRSS = CNTPMN.DEP.CRSS;
  ELSE
    PARAM.PARAMETER.DEP.CRSS = PHON.PARAMETER.DEP.CRSS;
    CVTPMN.DEP.CRSS = CVTPHN.DEP.CRSS;
IF CNTPMN.DEP.TAIL NE CVTPGN.DEP.TAIL
  THEN
    PHON.PARAMETER.DEP.TAIL = PARAM.PARAMETER.DEP.TAIL;
    CVTPHN.DEP.TAIL = CVTPMN.DEP.TAIL;
  ELSE
    PARAM.PARAMETER.DEP.TAIL = PHON.PARAMETER.DEP.TAIL;
    CVTPMN.DEP.TAIL = CVTPMN.DEP.TAIL;
END PGLOBAL;
ROUTINE WGLOBAL

INOUT (WXLOG, CNTWX, WMO, CNTWXN, WMO, CNTWXNOW)
[This routine reconciles different versions of wind and weather planning log information]
LOOP; J = 11 to 13 [From 11th to 13th message for AT]

WNOW.TABLE(J).TIME = WXLOG.TABLE(J).TIME;
WNOW.TABLE(J).CEL = WXLOG.TABLE(J).CEL;
WNOW.TABLE(J).VIS = WXLOG.TABLE(J).VIS;
WNOW.TABLE(J).DIR = WXLOG.TABLE(J).DIR;
WNOW.TABLE(J).VEL = WXLOG.TABLE(J).VEL;
WNOW.TABLE(J).REMARK = WXLOG.TABLE(J).REMARK;
CNYTMI(J) = CNTWX.TABLE(J).TIME;
CNYTMI(J) = CNTWX.TABLE(J).CEL;
CNYTMI(J) = CNTWX.TABLE(J).VIS;
CNYTMI(J) = CNTWX.TABLE(J).DIR;
CNYTMI(J) = CNTWX.TABLE(J).VEL;
ENDLOOP;
END WGLOBAL;

ROUTINE QGLOBAL

INOUT (QUELEN, QYQILN, QREQLN, QYTQMN, QNOW, QYTQNOW)
[This routine reconciles different versions of current departure queue information]
LOOP; J = 1 to 4 [up to 4 departure runways]
IF CNTQMN(J) NE QYTQNOW(J)
THEN QNOW.QL(J) = QUELEN.QL(J);
CNYTQMN(J) = CNTQMN(J);
ELSE QUELEN.QL(J) = QNOW.QL(J);
CNYTQMN(J) = QYTQNOW(J);
ENDLOOP;
END QGLOBAL;
ROUTINE SGLOBAL

INPUT (SRFLG, CNVTADP, SBDIN, CVTSW, SNOW, CVTSNO)
[This routine reconciles different versions of airport planning log information]

LOOP; [J = 11 to 13] [from 11th to 13th message for AC]
  SNOW.TABLE(J).TIME = SURFLOG.TABLE(J).TIME;
  SNOW.TABLE(J).RV = SURFLOG.TABLE(J).RV;
  SNOW.TABLE(J).SURF = SURFLOG.TABLE(J).SURF;
  SNOW.TABLE(J).BREAK = SURFLOG.TABLE(J).BREAK;
  SNOW.TABLE(J).CLOSED = SURFLOG.TABLE(J).CLOSED;
  SNOW.TABLE(J).OPEN = SURFLOG.TABLE(J).OPEN;
  SNOW.TABLE(J).REMARS = SURFLOG.TABLE(J).REMARS;
  CVTSNO.TABLE(J).TIME = CNVTADP.TABLE(J).TIME;
ENDLOOP;

END SGLOBAL;
ROUTINE GGLOBAL

INOUT (OAGLOG, CNVTOA, CBEGIN, CVTGN8N, GN01, CVTGNOW);

[this routine reconciles different versions of demand planning log information]

LOOP: [J = 0 to 23] [For 24 hours]

GN01.TABLE(J).TTLAAR = OAGLOG.TABLE(J).TTLAAR;
GN01.TABLE(J).TTLDEP = OAGLOG.TABLE(J).TTLDEP;
GN01.TABLE(J).KUBBS = OAGLOG.TABLE(J).KUBBS;
GN01.TABLE(J).GCT = OAGLOG.TABLE(J).GCT;
GN01.TABLE(J).VAINS = OAGLOG.TABLE(J).VAINS;
GN01.TABLE(J).FARMN = OAGLOG.TABLE(J).FARMN;
GN01.TABLE(J).NORTH = OAGLOG.TABLE(J).NORTH;
GN01.TABLE(J).EAST = OAGLOG.TABLE(J).EAST;
GN01.TABLE(J).SOUTH = OAGLOG.TABLE(J).SOUTH;
GN01.TABLE(J).WEST = OAGLOG.TABLE(J).WEST;

CVTGN8N.TABLE(J).TTLAAR = CNVTOAG.TABLE(J).TTLAAR;
CVTGN8N.TABLE(J).TTLDEP = CNVTOAG.TABLE(J).TTLDEP;
CVTGN8N.TABLE(J).KUBBS = CNVTOAG.TABLE(J).KUBBS;
CVTGN8N.TABLE(J).GCT = CNVTOAG.TABLE(J).GCT;
CVTGN8N.TABLE(J).VAINS = CNVTOAG.TABLE(J).VAINS;
CVTGN8N.TABLE(J).FARMN = CNVTOAG.TABLE(J).FARMN;
CVTGN8N.TABLE(J).NORTH = CNVTOAG.TABLE(J).NORTH;
CVTGN8N.TABLE(J).EAST = CNVTOAG.TABLE(J).EAST;
CVTGN8N.TABLE(J).SOUTH = CNVTOAG.TABLE(J).SOUTH;
CVTGN8N.TABLE(J).WEST = CNVTOAG.TABLE(J).WEST;

ENDLOOP;

END GGLOBAL;
ROUTINE CLOSING

IN (CNVTTPM);

INOUT (ARPT_DATA(I), CVRVT_APT(I));

[This routine closes runways based on wind conditions and weather minima]

LOOP; [J = 1 to 12] [tower imposed closures]

ARPT_DATA(I).RUNWAY(J).CLOSED.ARR = ARPT_DATA(I).RUNWAY(J).TOWER.AER;

ARPT_DATA(I).RUNWAY(J).CLOSED.DEF = ARPT_DATA(I).RUNWAY(J).TOWER.DEP;

[Closed due to wind]

IF (CVRVT_APT(I).RUNWAY(J).CRSS GT CNVTTPM.ARR.CRSS) OR (CVRVT_APT(I).RUNWAY(J).TAIL GT CNVTTPM.ARR.TAIL)

THEN

ARPT_DATA(I).RUNWAY(J).CLOSED.ARR = 'X';

IF (CVRVT_APT(I).RUNWAY(J).CRSS GT CNVTTPM.DEF.CRSS) OR (CVRVT_APT(I).RUNWAY(J).TAIL GT CNVTTPM.DEF.TAIL)

THEN

ARPT_DATA(I).RUNWAY(J).CLOSED.DEF = 'X';

[Closed due to wind]


THEN

ARPT_DATA(I).RUNWAY(J).CLOSED.ARR = 'X';

ENDLOOP;

END CLOSING;
ROUTINE  WIND

INOUT (ARPT DATA(I), CMVT APT(I));

[This routine computes crosswind and tailwind components of prevailing wind and sets up corresponding screen data fields]

$TWO \equiv 2;

\text{ANGLE}(1) \equiv 220. ;
\text{ANGLE}(2) \equiv 220. ;
\text{ANGLE}(3) \equiv 270. ;
\text{ANGLE}(4) \equiv 270. ;
\text{ANGLE}(5) \equiv 320. ;
\text{ANGLE}(6) \equiv 320. ;
\text{ANGLE}(7) \equiv 60. ;
\text{ANGLE}(8) \equiv 60. ;
\text{ANGLE}(9) \equiv 90. ;
\text{ANGLE}(10) \equiv 90. ;
\text{ANGLE}(11) \equiv 140. ;
\text{ANGLE}(12) \equiv 140. ;

\text{ARPT DATA}(I).\text{RUNWAY.DIR} \equiv \text{ARPT DATA}(I).\text{WIND.DIR};
\text{ARPT DATA}(I).\text{RUNWAY.VEL} \equiv \text{ARPT DATA}(I).\text{WIND.VEL};
\text{CMVT APT}(I).\text{RUNWAY.DIR} \equiv \text{CMVT APT}(I).\text{WIND.DIR};
\text{CMVT APT}(I).\text{RUNWAY.VEL} \equiv \text{CMVT APT}(I).\text{WIND.VEL};

\text{ANGLE} \equiv (\text{ANGLE} - \text{CMVT APT}(I).\text{RUNWAY.DIR}) \times 0.01745; \text{[convert to radians]}
\text{ARPT DATA}(I).\text{RUNWAY.RVR} \equiv (2) ^ \times ;

\text{LOOP};

\text{CMVT APT}(I).\text{RUNWAY}(J).\text{CRSS} \equiv \text{CMVT APT}(I).\text{WIND.VEL} \times \text{ABS(SIN(ANGLE(J))});
\text{CMVT APT}(I).\text{RUNWAY}(J).\text{CRSS} \equiv \text{FLOAT(CEIL(\text{CMVT APT}(I).\text{RUNWAY}(J).\text{CRSS} + .5))};
\text{ARPT DATA}(I).\text{RUNWAY}(J).\text{CRSS} \equiv \text{SUBSTR}(\text{CMVT APT}(I).\text{RUNWAY}(J).\text{CRSS.$TWO \equiv 1,2});
IF \( \text{ABS(ANGLE(J))} \geq 1.57079 \) THEN

\[
\text{CNVRT.APT(I).RUNWAY(J).TAIL} = 0.0;
\]
\[
\text{ARPT_DATA(I).RUNWAY(J).TAIL} = ' 0';
\]

ELSE

\[
\text{CNVRT.APT(I).RUNWAY(J).TAIL} = \text{CNVRT.APT(I).WIND.VEL} \times \cos(\text{ANGLE(J)})
\]
\[
\text{CNVRT.APT(I).RUNWAY(J).TAIL} = \text{FLOAT(FLOOR(CNVRT.APT(I).RUNWAY(J).TAIL + 0.5))}
\]
\[
\text{ARPT_DATA(I).RUNWAY(J).TAIL} = \text{SUBSTR(CNVRT.APT(I).RUNWAY(J).TAIL,1,2)}
\]

ENDLOOP;

END WIND;
ROUTINE MINIMA

IN (RWYPEQ(I), RWYMIN)
INOUT (APTSTAT(I), CNVTAPT(I));

[This routine computes ceiling and visibility minima based on existing airport's equipment status]

$THREE = 3;
$FOUR = 4;

LOOP: [J = 1 to 12] [for each runway]

IF RWYPEQ(I),RUNWAY(J),CATII EQ (2) ' ' THEN [CATII is up]

CNVTAPT(I).RWY(J).CEIL = RWYMIN(J).CATII.MONE.CEIL;
CNVTAPT(I).RWY(J).VIS = RWYMIN(J).CATII.MONE.VIS;

APTSTAT(I).RWY(J).CEIL = SUBSTR(P(CNVTAPT(I).RWY(J).CEIL, $FOUR),1,4);
C = SUBSTR(P(100.0 * CNVTAPT(I).RWY(J).VIS, $THREE),1,3);
APTSTAT(I).RWY(J).VIS = SUBSTR(C,1,1) CONCATENATE '.' CONCATENATE SUBSTR(C,2,2);

ELSE [CATII is down]

IF (RWYPEQ(I).RUNWAY(J).LOC NE (2) ' ') AND (RWYPEQ(I).RUNWAY(J).HDB_VOR NE (2) ' ') THEN [both localizer and HDB_VOR are down]

CNVTAPT(I).RWY(J).CEIL = 10000.0;
CNVTAPT(I).RWY(J).VIS = 5.0;
APTSTAT(I).RWY(J).CEIL = (4) ' ';
APTSTAT(I).RWY(J).VIS = (4) ' ';

ELSE [localizer or HDB_VOR are not down]

IF (RWYPEQ(I).RUNWAY(J).LOC NE (2) ' ') AND (RWYPEQ(I).RUNWAY(J).HDB_VOR NE (2) ' ')


THEN [Localizer is down and NDB_VOR is up]

\[ \text{CNVTAPT}(I).\text{RWY}(J).\text{CEIL} = \text{RWYMIN}(J).\text{NDB_VOR}.\text{NONE}.\text{CEIL}; \]
\[ \text{CNVTAPT}(I).\text{RWY}(J).\text{VIS} = \text{RWYMIN}(J).\text{NDB_VOR}.\text{NONE}.\text{VIS}; \]

\[ \text{IF} \ \text{RMTQP}(I).\text{RUNWAY}(J).\text{RAIL NE} \ (2) \ ' ' \]
\[ \text{THEN} [\text{RAIL is also down}] \]
\[ \text{CNVTAPT}(I).\text{RWY}(J).\text{CEIL} = \max(\text{CNVTAPT}(I).\text{RWY}(J).\text{CEIL}, \]
\[ \text{RWYMIN}(J).\text{NDB_VOR}.\text{RAIL}.\text{CEIL}); \]
\[ \text{CNVTAPT}(I).\text{RWY}(J).\text{VIS} = \max(\text{CNVTAPT}(I).\text{RWY}(J).\text{VIS}, \]
\[ \text{RWYMIN}(J).\text{NDB_VOR}.\text{ALS}.\text{VIS}) \]
\[ \text{APTSTAT}(I).\text{RWY}(J).\text{CEIL} = \text{SUBSTR}(\text{CNVTAPT}(I).\text{RWY}(J).\]
\[ \text{CEIL}, \$\text{FOUR}, 1,4); \]
\[ C = \text{SUBSTR}(100. \ * \ \text{CNVTAPT}(I).\text{RWY}(J).\text{VIS}, \$\text{THRES}, 1,3); \]
\[ \text{APTSTAT}(I).\text{RWY}(J).\text{VIS} = \text{SUBSTR}(C, 1, 1) \text{ CONCATENATE} \ ' ' \]
\[ \text{CONCATENATE} \ \text{SUBSTR}(C, 2, 2); \]

ELSE [localizer is up]

\[ \text{IF} \ \text{RMTQP}(I).\text{RUNWAY}(J).\text{LOC EQ} \ (2) \ ' ' \ \text{AND} \ \text{RMTQP}(I).\]
\[ \text{RUNWAY}(J).\text{GS NE} \ (2) \ ' ' \]
\[ \text{THEN} [\text{glide slope is down}] \]
\[ \text{CNVTAPT}(I).\text{RWY}(J).\text{CEIL} = \text{RWYMIN}(J).\text{LOC}.\text{NONE}.\text{CEIL}; \]
\[ \text{CNVTAPT}(I).\text{RWY}(J).\text{VIS} = \text{RWYMIN}(J).\text{LOC}.\text{NONE}.\text{VIS}; \]

\[ \text{IF} \ \text{RMTQP}(I).\text{RUNWAY}(J).\text{MM NE} \ (2) \ ' ' \]
\[ \text{THEN} [\text{middle marker is down}] \]
\[ \text{CNVTAPT}(I).\text{RWY}(J).\text{CEIL} = \max(\text{CNVTAPT}(I).\text{RWY}(J).\text{CEIL}, \]
\[ \text{RWYMIN}(J).\text{LOC}.\text{MM}.\text{CEIL}); \]
\[ \text{CNVTAPT}(I).\text{RWY}(J).\text{VIS} = \max(\text{CNVTAPT}(I).\]
\[ \text{RWY}(J).\text{VIS}, \text{RWYMIN}(J).\text{LOC}.\text{MM}.\text{VIS}); \]
IF RVYQP(1).RUNWAY(J).RAIL H# (2) = 2
THEN [RAIL is also down]

    CNTAPT(1).RNY(J).CEIL = 
    MAX(CNTAPT(1).RNY(J).CEIL,RNYMIN(J)
    .LOC.RAIL.CEIL);
    CNTAPT(1).RNY(J).VIS = 
    MAX(CNTAPT(1).RNY(J).VIS,RNYMIN(J)
    .LOC.RAIL.VIS);

IF RVYQP(1).RUNWAY(J).ALS H# (2) = 2
THEN [ALS is also down]

    CNTAPT(1).RNY(J).CEIL = 
    MAX(CNTAPT(2).RNY(J).CEIL,RNYMIN(J)
    .LOC.ALS.CEIL);
    CNTAPT(1).RNY(J).VIS = 
    MAX(CNTAPT(2).RNY(J).VIS,RNYMIN(J)
    .LOC.ALS.VIS);
    APTSTRAT(1).RNY(J).CEIL = 
    SUBSTR(P(CNTAPT(1).RNY(J).CEIL,$FOU R),1,4);
    C = SUBSTR(P(100. *
    CNTAPT(1).RNY(J).VIS,$THREE),1,3);
    APTSTRAT(1).RNY(J).VIS = 
    SUBSTR(C,1,1) CONCATENATE 
    CONCATENATE SUBSTR(C,T,27);

ELSE [localizer is up and glide slope is up]

    IF (RVYQP(1).RUNWAY(J).LOC H# (2) = 1) AND
    (RVYQP(1).RUNWAY(J).GS H# (2) = 1)
THEN [glide slope is down]

CVYTAAT(I).RUT(J).CEIL =
     CVYTHM(J).ILS.NOME.CEIL;

CVYTAAT(I).RUT(J).VIS =
     CVYTHM(J).ILS.NOME.VIS;

IF RUNYQP(I).RUNWAY(J).WHE (2)"

THEN [middle marker is also down]

CVYTAAT(I).RUT(J).CEIL =
     MAX(CVYTAAT(I).RUT(J).CEIL, CVYTHM(J)
     .ILS.WM.CEIL);

CVYTAAT(I).RUT(J).VIS =
     MAX(CVYTAAT(I).RUT(J).VIS, CVYTHM(J)
     .ILS.WM.VIS);

IF (RUNYQP(I).RUNWAY(J).RAME (2)"
OR (RUNYQP(I).RUNWAY(J).ALS WE (2)"

THEN [RAIL is also down or ALS is down]

CVYTAAT(I).RUT(J).CEIL =
     MAX(CVYTAAT(I).RUT(J).CEIL, CVYTHM(J)
     .RAIL.ALS.CEIL);

CVYTAAT(I).RUT(J).VIS =
     MAX(CVYTAAT(I).RUT(J).VIS, CVYTHM(J)
     .RAIL.ALS.VIS);

IF (RUNYQP(I).RUNWAY(J).TDE WE (2)"

THEN [TDE is also down]

CVYTAAT(I).RUT(J).CEIL =
     MAX(CVYTAAT(I).RUT(J).CEIL, CVYTHM(J)
     .JLS.TDE.CEIL);
CMVAPT(I).RVY(J).VIS = MAX(CMVAPT(I).RVY(J).VIS.RWYMIN(J). ILS.TEX.Vis);

IF (RVMEQP(I).RUNWAY(J).CL ML (2) ' 
THEN (CL is also down)

CMVAPT(I).RVY(J).CELL = MAX(CMVAPT(I).RVY(J).CELL. RWYMIN(J). ILS.CL.Vis);
CMVAPT(I).RVY(J).VIS = MAX(CMVAPT(I).RVY(J).VIS.RWYMIN(J). ILS.CL.Vis);
C = SUBSTR(P(100. * CMVAPT(I). RVY(J).VIS,$THREE),1,3);
APTSYAT(I).RVY(J).VIS = SUBSTR(C,1,1) CONCATIMATE '.'
CONCATIMATE SUBSTR(C,2,2);

IF RVMEQP(I).RUNWAY(J).ML ML (2) ' 
THEN (ML is down)

CMVAPT(I).RVY(J).VIS = 2.0;
APTSYAT(I).RVY(J).VIS = '2.00';

END MINIMA;
ROUTINE ELIG

IN (CHFCOQ, APTSTAT(I), CVTAPT(I), ENTERG(1));

INOUT (ELIGBLY(I));

[This routine determines eligibility of configurations based on runway closures, weather conditions, and equipment status]

PERFORM CONFIGURATION_ID_SET_UP;

$THW = 2;
ELIGBLY(I).ID = (73) '0';
ELIGBLY(I).NUM = 0;

IF (CVTAPT(I).WX.CEIL LT 100.) OR (CVTAPT(I).WX.VIS LT .25) [if ceiling is below 100 and visibility is below .25]

THEN ELIGBLY(I).ID = (73)'1'; [all configurations are ineligible]

PERFORM BELOW_200_CEILING_PlUS_EQUIPMENT_OUTAGE_ELIGIBILITY_CHECK;

PERFORM RUNWAY_CLOSURE_ELIGIBILITY_SET_UP;

LOOP: [J = 1 To 73] [up to 73 possible configurations]

EFLAG = '0'; [set eligibility flag to 'eligible']

PERFORM RUNWAY_CLOSURE_ELIGIBILITY_CHECK;

IF [Configuration J is ineligible]

THEN:

ELSE

PERFORM BELOW_200_CEILING_ELIGIBILITY_CHECK;

IF [configuration J is ineligible]

THEN:

ELSE
PERFORM BELOW_5 VIS_PLUS_NON_RVR_CONFIGURATION_ELIGIBILITY_CHECK;

IF [configuration J is ineligible]
THEN;
ELSE
PERFORM BELOW_1000_CEIL_Below_3 VIS_ELIGIBILITY_CHECK;

IF [configuration J is ineligible]
THEN;
ELSE
PERFORM BETWEEN 4800 TO 200_CEILING AND 5 TO .25 VIS
PLUS_EQUIPMENT OUTAGE_ELIGIBILITY_CHECK;

IF [configuration J is ineligible]
THEN;
ELSE
PERFORM HOLD_SHORT_ELIGIBILITY_CHECK;
END LOOP;

LOOP: [J = 1 to 73]

IF SUBSTR(ELIGIBILITY(1).ID,J,1) = 'O'
THEN ELIGIBILITY(1).NUM = ELIGIBILITY(1).NUM + 1;
END LOOP;

END ELIG;
PROCESS CONFIGURATION ID SET UP

This process initializes certain necessary variables for MIG routine.

```plaintext
BZERO = (12)'0'B;

(set up parallel runway configuration ID's)
PARAPP(1) = '1107000000000'B CONCATENATE BZERO; [48, 4L]
PARAPP(2) = '0011000000000'B CONCATENATE BZERO; [9K, 9L]
PARAPP(3) = '0000110000000'B CONCATENATE BZERO; [14K, 14L]
PARAPP(4) = '0000001100000'B CONCATENATE BZERO; [24K, 24L]
PARAPP(5) = '0000000011000'B CONCATENATE BZERO; [27K, 27L]
PARAPP(6) = '0000000000111'B CONCATENATE BZERO; [32K, 32L]

(set up certain dual runway configuration ID's)
DUALAPP(1) = '1010000000000'B CONCATENATE BZERO; [48, 4K]
DUALAPP(2) = '0010100000000'B CONCATENATE BZERO; [9K, 9K]

(set up triple runway configurations)
TRIPAPP(1) = '1011000000000'B CONCATENATE BZERO; [48, 9K, 9L]
TRIPAPP(2) = '0010101000000'B CONCATENATE BZERO; [9K, 14K, 14L]
TRIPAPP(3) = '0001010100000'B CONCATENATE BZERO; [14K, 14K, 24K]
TRIPAPP(4) = '0000110100000'B CONCATENATE BZERO; [14K, 24K, 24L]
TRIPAPP(5) = '0000010100000'B CONCATENATE BZERO; [14K, 24K, 24L]
TRIPAPP(6) = '0000001010000'B CONCATENATE BZERO; [27K, 27L, 27L]
TRIPAPP(7) = '0000000101000'B CONCATENATE BZERO; [27K, 27L, 27L]
TRIPAPP(8) = '0000000010100'B CONCATENATE BZERO; [27K, 27L, 27L]

(set up bold short configurations)
HLDSHRT(1) = '0000100000100'B CONCATENATE BZERO; [14K, 27L]
HLDSHRT(2) = '0000000010100'B CONCATENATE BZERO; [27L, 32L]
HLDSHRT(3) = '0000000010010'B CONCATENATE BZERO; [27L, 32L]
HLDSHRT(4) = '0010100000000'B CONCATENATE BZERO; [9K, 14K]
```

END CONFIGURATION_ID_SET_UP;
PROCESS BELOW_200_CEILING_PLUG_EQUIPMENT_OUTAGE_ELIGIBILITY_CHECK

(This determines eligibility of configurations with ceiling below 200 and certain equipment out)

IF

(CURRENT_X.Y_CEIL_LT_200.) AND

(RWQRP(I), RWMAY(5).LOC ME (2) ' ') OR
(RWQRP(I).RWMAY(5).GS ME (2) ' ') OR
(RWQRP(I).RWMAY(6).LOC ME (2) ' ') OR
(RWQRP(I).RWMAY(6).GS ME (2) ' ') OR
(RWQRP(I).RWMAY(6).OM ME (2) ' ') OR
(RWQRP(I).RWMAY(6).SW ME (2) ' ') OR
(RWQRP(I).RWMAY(6).WME ME (2) ' ') OR
(RWQRP(I).RWMAY(6).ALS ME (2) ' ') OR
(RWQRP(I).RWMAY(6).ALS ME (2) ' ')

THEN ELIGIBLE(I).ID = (73) '1';

(if the prevailing ceiling is below 200 and any one of the following equipment: localizer, glide slope, middle marker, outer marker, or ALS is out; then all configurations are ineligible)

END BELOW_200_CEILING_PLUG_EQUIPMENT_OUTAGE_ELIGIBILITY_CHECK;
PROCESS RUNWAY_CLOSURE_ELIGIBILITY_SET_UP
[This process sets up certain necessary variables for ELIG routine in order to check for eligibility]

IF APTSTAT(I).RUNWAY(J).CLOSED.ARE EQ (2) ' ' THEN AINELIG = '0'B; ELSE AINELIG = '1'B;
LOOP: [J = 2 To 12]

IF APTSTAT(I).RUNWAY(J).CLOSED.ARE EQ (2) ' ' THEN AINELIG = AINELIG CONCATENATE '0'B; ELSE AINELIG = AINELIG CONCATENATE '1'B;
ENDLOOP;

IF APTSTAT(I).RUNWAY(J).CLOSED.DEF EQ (2) ' ' THEN DINELIG = '0'B; ELSE DINELIG = '1'B;
LOOP: [J = 2 to 12]

IF APTSTAT(I).RUNWAY(J).CLOSED.DEF EQ (2) ' ' THEN DINELIG = DINELIG CONCATENATE '0'B; ELSE DINELIG = DINELIG CONCATENATE '1'B;
ENDLOOP;

INELIG = AINELIG CONCATENATE DINELIG; [set up an ID for closed runways]
[set up an ID for non_RVE runways]
RVECK = '0'B;
LOOP:  \[ K = 1 \text{ to } 12 \]

IF RWRK(1).RUNWAY(K).RVR NE (2) ' '  
THEN RVRCX = RVRCX CONCATenate '1'B;  
ELSE RVRCX = RVRCX CONCATenate '0'B;  
ENDLOOP;  

RVRCX = RVRCX CONCATenate BZERO;  

END RUNWAY_CLOSURE_ELIGIBILITY_SET_UP;
PROCESS RUNWAY_CLOSURE_ELIGIBILITY_CHECK
[This process determines eligibility of configurations with runways closed]

IF
((CONFIG(J).ID) AND (INELIG)) NE (24) '0' B
THEN
EFLAG = '1' B
[if one or more of closed runways are in configuration J then that configuration is ineligible]
END RUNWAY_CLOSURE_ELIGIBILITY_CHECK;

PROCESS BELOW_200_CEILING_ELIGIBILITY_CHECK
[This process determines eligibility of configurations with ceiling below 200]

IF
((CONVAP(1).WX.CEIL LT 200.) AND ((CONFIG(J).ID) AND (PARAP(3)) NE PARAP(3)))
THEN
EFLAG = '1' B
[if the ceiling is below 200 and configuration J is other than parallel 14's then it is ineligible]
END BELOW_200_CEILING_ELIGIBILITY_CHECK

PROCESS BELOW_.5_VIS_PLUS_NON_RVR_CONFIGURATION_ELIGIBILITY_CHECK
[This process determines eligibility of configurations with visibility below .5 and non-RVR runways]

IF
((CONVAP(1).WX.VIS LT .5) AND ((CONFIG(J).ID) AND (RVR(R)) NE '0' B
THEN
EFLAG = '1' B
[if the visibility is below .5 and there are non_RVR runways in configuration J then it is ineligible]
END BELOW_.5_VIS_PLUS_NON_RVR_CONFIGURATION_ELIGIBILITY_CHECK;
PROCESS BELOW_800_CEIL_2_VIS_ELIGIBILITY_CHECK

[This process determines eligibility of configurations with ceiling and visibility below 800 and 2 respectively]

IF CHVTAPT(1).WX.CEIL LT 800 OR (CHVTAPT(1).WX.VIS LT 2)
THEN
   FLAG = '0'B;
REPEAT UNTIL (FLAG = '1'B); [K = 1 to 6]
   IF ((CMFQRQ(J).ID) AND (PARAPP(K) EQ PARAPP(K))
      THEN FLAG = '1'B;
ENDREPEAT;

IF FLAG NE '1'B
THEN RFLAG = '1'B;

[if visibility is below 2 or ceiling is below 800 all non-parallel configurations are ineligible]

END BELOW_800_CEIL_2_VIS_ELIGIBILITY_CHECK;
PROCESS BELOW_1000_CEIL_3_VIS_ELIGIBILITY_CHECK;
[This process determines eligibility of configurations with ceiling and visibility below 1000 and 3 respectively]

IF (CONTAPE(1).EX_CEIL LT 1000.) OR (CONTAPE(1).EX_VIS LT 3)) AND
((CHPQI(1).ID) AND (TRIPAPP(1)) OR
((CHPQI(1).ID) AND (TRIPAPP(2)) OR TRIPAPP(2)) OR
((CHPQI(1).ID) AND (TRIPAPP(3)) OR TRIPAPP(3)) OR
((CHPQI(1).ID) AND (TRIPAPP(4)) OR TRIPAPP(4)) OR
((CHPQI(1).ID) AND (TRIPAPP(5)) OR TRIPAPP(5)) OR
((CHPQI(1).ID) AND (TRIPAPP(6)) OR TRIPAPP(6)) OR
((CHPQI(1).ID) AND (TRIPAPP(7)) OR TRIPAPP(7)) OR
((CHPQI(1).ID) AND (TRIPAPP(8)) OR TRIPAPP(8)) OR
((CHPQI(1).ID) AND (DualAPP1(1)) OR DualAPP1(1)) OR
((CHPQI(1).ID) AND (DualAPP1(2)) OR DualAPP1(2))
THEN EFLAG = '1';
[if the ceiling is below 1000 and visibility is below 3 then triple and certain dual configurations are ineligible]

END BELOW_1000_CEIL_3_VIS_ELIGIBILITY_CHECK;
This process determines eligibility of configurations with ceiling between 200 and 4800, visibility between .25 and 5 and certain equipment inoperable.

IF (((CNVTAP(I).WX.CEIL LT 1000) AND (CNVTAP(I).WX.CEIL GE 200)) OR ((CNVTAP(I).WX.VIS LT 3.) AND (CNVTAP(I).WX.VIS GE .25))) THEN

REPEAT UNTIL (EFLAG = '1B); [K = 2 to 12 BY 2]

IF (ENTRYQ(1).RECEP(K).GS NE (2) OR ENTRYQ(1).RECEP(K-1).GS NE (2) OR ENTRYQ(1).RECEP(K).ON NE (2) OR ENTRYQ(1).RECEP(K-1).ON NE (2) OR ENTRYQ(1).RECEP(K).ALS NE (2) OR ENTRYQ(1).RECEP(K-1).ALS NE (2) AND ((CMFGQ(1).ID) AND (PARAM(K/2)) OR PARAP(K)/2))

THEN EFLAG = '1B;

[if ceiling is between 1000 to 200 and visibility is between 3 and .25 and any of the following equipment: glide slope, outer marker, middle marker, or ALS is out then parallel configurations are ineligible]

ENDREPEAT

IF (((CNVTAP(I).WX.CEIL LT 4800) AND (CNVTAP(I).WX.CEIL GE 200)) OR ((CNVTAP(I).WX.VIS LT 3.) AND (CNVTAP(I).WX.VIS GE .25))) THEN

REPEAT UNTIL (EFLAG = '1B); [K = 2 to 12 BY 2]

IF (ENTRYQ(1).RECEP(K).LOC NE (2) OR ENTRYQ(1).RECEP(K-1).LOC NE (2) OR ENTRYQ(1).RECEP(K-1).LOC NE (2) OR ENTRYQ(1).RECEP(K-1).LOC NE (2) AND (CMFGQ(1).ID) AND (PARAM(K/2)) OR PARAP(K)/2))

THEN EFLAG = '1B;

[if ceiling is between 4800 and 200, and visibility is between 5 and .25 and localizer is out, then parallel configurations are ineligible]

ENDREPEAT;

END BETWEEN_4800_TO_200_CEILING_AND_5_TO_.25_VISIBILITY_PLUS_EQUIPMENT_OUTAGE_ELIGIBILITY_CHECK;
process hold_short_eligibility_check
[This process determines eligibility for hold short configurations]

if ((aptstat(i).runway(j).surf_nr (2) ' ') or (aptstat(i).runway(j).hwx_nr (2) ' ') and
    (confqr(i).id and hldshrt(3)) or hldshrt(3))
then elflag = '1'b;

if ((aptstat(i).runway(10).surf_nr (2) ' ') or (aptstat(i).runway(10).hwx_nr (2) ' ') and
    (confqr(i).id and hldshrt(1)) or (confqr(i).id and hldshrt(2)) or hldshrt(2))
then elflag = '1'b;

if (aptstat(i).runway(5).hwx_nr (2) ' ') and ((confqr(i).id and hldshrt(4)) or hldshrt(4)) or
   ((confqr(i).id and hldshrt(1)) or hldshrt(1))
then elflag = '1'b;

end hold_short_eligibility_check;
ROUTINE FILES

IN (APTSTAT(I), CNVTAPT(I));

INOUT (FILENUM(I), CNVDTN(I));

(This routine determines capacity file number for each configuration and sets CNVDTN variable to indicate VFR(=1) or IFR(=2))

LOOP: [M = 1 to 73] [determine appropriate capacity file]

IF (CNVTAPT(I).WX.CEIL LT 800) OR (CNVTAPT(I).WX.VIS LT 2)

THEN

   CNVDTN(I) = 2;
   FILENUM(I).CONF(N) = 3;

   REPEAT WHILE (FILENUM(I).CONF(N) EQ 3); [K = 1 to 12]

   IF (APTSTAT(I).RUNWAY(K).BLK EQ 'X ')

      THEN FILENUM(I).CONF(N) = 4;

   ENDREPEAT;

ELSE

   CNVDTN(I) = 1;
   FILENUM(I).CONF(N) = 1;

   REPEAT WHILE (FILENUM(I).CONF(N) EQ 1); [K = 1 to 12]

   IF (APTSTAT(I).RUNWAY(K).BLK EQ 'X ')

      THEN FILENUM(I).CONF(N) = 2;

   ENDREPEAT;

ENDLOOP;

END FILES;
ROUTINE PERCENT

IN (CNVTDIS(I), CNFGQQ);

INOUT (FLCARR(I));

[This routine computes north and south demands based on fix-to-runway assignments plus percentage of arrivals]

PERFORM INITIALIZATION (PERCENT);

PFCARR(I).TOTARR = CNVTDIS(I).ARR_TOTAL;
PFCARR(I).TOTDEP = CNVTDIS(I).DEP_TOTAL;

LOOP: [J = 1 to 73]

PFCARR(I).CONF(J).NARRDEN = 0;
PFCARR(I).CONF(J).SARRDEN = 0;
[compute total arrival demand for north complex]

LOOP: [K1 = 1 to 6]

IF (CNFGQQ(J).ID AND COMPLEX.NORTH1(K1)) NE 0

THEN TEMP = COMPLEX.NORTH1(K1);

LOOP: [K2 = 1 to 6]

IF CNFGQQ(J).ARR(K2) EQ TEMP

THEN PFCARR.CONF(J).NARRDEN = PFCARR.CONF(J).NARRDEN +
DURNT.ARR(K2);

ENDLOOP;

ENDLOOP;
[compute total arrival demand for south complex]

[compute total departure demand for north complex]
LOOP: [K1 = 1 to 6]
    IF (CMFGRQ(J).ID AND COMPLEX.WORTH1(K1)) NE 0
    THEN
        TEMP = COMPLEX.WORTII2(K1);
        LOOP: [K2 = 1 to 5]
            IF CMFGRQ(J).DEP(K2) EQ TEMP
            THEN PRCARR(I).CONF(J).DEPDEM = PRCARR(I).CONF(J).DEPDEM + DEMP.
                DEP(K2);
        ENDLOOP;
    ENDLOOP;

[compute total departure demand for south complex]

[compute percentages]
IF (PRCARR(I).CONF(J).DEPDEM + PRCARR(I).CONF(J).MAREDDE) EQ 0
THEN PRCARR(I).CONF(J).PERCENT = 0.5;
    PRCARR(I).CONF(J).DEPDEM);
IF (PRCARR(I).CONF(J).DEPDEM + PRCARR(I).CONF(J).SAREDDE) EQ 0
THEN PRCARR(I).CONF(J).SPREDT = 0.5;
    PRCARR(I).CONF(J).DEPDEM);
ENDLOOP;

END PERCENT;
PROCESS INITIALIZATION
(This process performs initialization for PERCENT routine)

BZERO = (12) '0'B;

\Cop(1) = '010000000000'B CONCATENATE BZERO;
\Cop(2) = '000100000000'B CONCATENATE BZERO;
\Cop(3) = '000010000000'B CONCATENATE BZERO;
\Cop(4) = '000001000000'B CONCATENATE BZERO;
\Cop(5) = '000000100000'B CONCATENATE BZERO;
\Cop(6) = '000000010000'B CONCATENATE BZERO;

\Cop(1) = BZERO CONCATENATE '010000000000'B;
\Cop(2) = BZERO CONCATENATE '000100000000'B;
\Cop(3) = BZERO CONCATENATE '000010000000'B;
\Cop(4) = BZERO CONCATENATE '000001000000'B;
\Cop(5) = BZERO CONCATENATE '000000100000'B;
\Cop(6) = BZERO CONCATENATE '000000010000'B;

DUMMY.ARR(1) = CWSTRM(1).ARR.KURS;
DUMMY.ARR(2) = CWSTRM(1).ARR.CYT;
DUMMY.ARR(3) = CWSTRM(1).ARR.PLANT;
DUMMY.ARR(4) = CWSTRM(1).ARR.PAINS;
DUMMY.ARR(5) = CWSTRM(1).ARR.PALIN;
DUMMY.ARR(6) = CWSTRM(1).ARR.NEX_A;

DUMMY.DEF(1) = CWSTRM(1).DEF.NORTH;
DUMMY.DEF(2) = CWSTRM(1).DEF.EAST;
DUMMY.DEF(3) = CWSTRM(1).DEF.SOUTH;
DUMMY.DEF(4) = CWSTRM(1).DEF.WEST;
DUMMY.DEF(5) = CWSTRM(1).DEF.NEX_B;

END INITIALIZATION;
ROUTINE QFIX
IN (CHFGRC(CONFIND(1)));

INPUT (QUELEM);

[This routine updates departure runways for current operating configuration in current departure queue screen]

LOOP: [J = 1 to 4]
QUELEM.DEPRUN(J) = CHFGRC(CONFIND(1)).DEP_RUN(J);

ENDLOOP;

END QFIX;
ROUTINE CAPSAT

IN (PRCARR(I), CHPGRO, CAPFILE, FILENUM(I), ELGRTY(I));

INOUT (INFORM(I));

This routine computes capacity and performs demand balancing for each eligible configuration

SWITCH(1) = 2;
SWITCH(2) = 1;

IF PRCARR(I).TOTARR + PRCARR(I).TOTTDP EQ 0.

THEN ATOTPRC = .5;
ELSE ATOTPRC = PRCARR(I).TOTARR/(PRCARR(I).TOTARR + PRCARR(I).TOTTDP);

LOOP: [I = 1 to 73] [up to 73 eligible configurations]

IF SUM(ELGRTY(I).ID,N,I) EQ 0'S

THEN [If the configuration N is eligible]

FLAG = 0;

PERFORM CAPACITY CURVE SELECTION;

IF FLAG EQ 0

THEN [Demand balancing]

CALL DEAL;

IN (CAPACITY,PAIR,PRCARR(I).TOTARR,PRCARR(I).TOTTDP,
PRCARR(I).CONF(N), HARRDNM,PRCARR(I).CONF(N),HREPDM);

(I).CONF(N).HEPFRONT);

[This routine balances demand]

IF PRCARR(I).CONF(N).HFRONT OR 0.

2-120
THEN [not saturated]
  FRONT = PSCARR(1).COMP(N).BMPFRONT; [using balanced percentage of arrivals]
  PERFORM NORTH COMPLEX CAPACITY CALCULATIONS;
  FRONT = PSCARR(1).COMP(N).BMPFRONT; [using balanced percentage of arrivals]
  PERFORM SOUTH COMPLEX CAPACITY CALCULATIONS;
  SRECAP = INFORM(1).COMP(N).MARECAP + INFORM(1).COMP(N).SRECAP;
  DEPCAP = INFORM(1).COMP(N).DEPECAP + INFORM(1).COMP(N).SDEPCAP;
ELSE [saturated]
  FRONT = PSCARR(1).COMP(N).MFRONT; [using unbalanced percentage of arrivals];
  PERFORM NORTH COMPLEX CAPACITY CALCULATIONS;
  FRONT = PSCARR(1).COMP(N).MFRONT; [using unbalanced percentage of arrivals]
  PERFORM SOUTH COMPLEX CAPACITY CALCULATIONS;
  SRECAP = INFORM(1).COMP(N).MARECAP + INFORM(1).COMP(N).SRECAP;
  DEPCAP = INFORM(1).COMP(N).MDEPCAP + INFORM(1).COMP(N).SDEPCAP;
IF FLAG EQ 1
  THEN [north only configuration]
  PERFORM NORTH_ONLY_CAPACITY_COMPUTATION;
ELSIF
  PERFORM SOUTH_ONLY_CAPACITY_COMPUTATION;
  PERFORM CONSTRAINT CAPACITY OF EITHER AIRPORT;
  PERFORM SATURATION COMPUTATION;
  PERFORM CHANGE DUE TO DEMAND BALANCING COMPUTATION;
  PERFORM FINAL SATURATION CHECK;
ELSE [for ineligible configuration]
  INFORM(1).COMP(N).CAPACITY = -1.0;
  INFORM(1).COMP(N).INDEX = 999;
END LOOP;
END CAPSAT;
PROCESS CAPACITY CURVE SELECTION

(This process selects proper capacity curve for north and south complexes]

INFORM(1).CONF(N).INDEX = H;
L = FILENUM(1).CONF(N);
M(1) = CHFGRQ(N).NORTH;  [obtain north and south complex indices]
M(2) = CHFGRQ(N).SOUTH;

LOOP: [ R = 1 to 2] [retrieve north and south capacity curves from CAPPILA]

IF M(R) NE 0

THEN

LOOP: [ J = 1 to 14]
CAPACITY(R,J) = CAPPILA(L).KEY(M(R)).CAP(J);

ENDLOOP;
PAIR(R) = CAPPILA(L).KEY(M(R)).PAIR;
FLAG = SWITCH(FLAG);

ENDLOOP;
CAP1 = CAPACITY(1,*);
CAP2 = CAPACITY(2,*);

END CAPACITY_CURVE_SELECTION;
PROCESS NORTH_COMPLEX_CAPACITY_CALCULATIONS
[This process computes capacity of north complex]

CALL CAPCAL;

IN (PAIR(1), CAP1, PRENT);

OUT (ACAP, DCAP);

[This routine computes arrival and departure capacities of a complex based on percentage of
arrivals and a particular capacity curve]

INFORM(1).COMP(N).NAMECAP = ACAP;
INFORM(1).COMP(N).NAMEPCAP = DCAP;

END NORTH_COMPLEX_CAPACITY_CALCULATIONS;

PROCESS SOUTH_COMPLEX_CAPACITY_CALCULATIONS
[This process computes capacity of south complex]

CALL CAPCAL;

IN (PAIR(2), CAP2, PRENT);

OUT (ACAP, DCAP);

[This routine computes arrival and departure capacities of a complex based on percentage of
arrivals and a particular capacity curve]

INFORM(1).COMP(N).NAMECAP = ACAP;
INFORM(1).COMP(N).NAMEPCAP = DCAP;

END SOUTH_COMPLEX_CAPACITY_CALCULATIONS;
PROCESS NORTH_ONLY_CAPACITY_CALCULATION

(This process computes capacity for north only configurations)

CALL CAPCAL;

IN (PAIR(1), CAP(2), ATOTPRC);

OUT (ACAP, DCAP);

IF (PRCARR(1).TOTARR + PRCARR(1).TOTDEF) LE (ACAP + DCAP)
THEN [not saturated]

PRCARR(1).CONF(N).REPRCHT = ATOTPRC;
PRCARR(1).CONF(N).REPREMT = 0.1;
PRCARR(1).CONF(N).SHAREDFEM = PRCARR(1).TOTARR;
PRCARR(1).CONF(N).SHREDPERM = PRCARR(1).TOTDEF;
PRCARR(1).CONF(N).SHAREDEF = 0.1;
PRCARR(1).CONF(N).SREPERM = 0.1;
INFORM(1).CONF(N).MARKCAP = ACAP;
INFORM(1).CONF(N).MDPRCAP = DCAP;
INFORM(1).CONF(N).SARKCAP = 0.1;
INFORM(1).CONF(N).SDPRCAP = 0.1;
MARKCAP = ACAP;
MDPRCAP = DCAP;

ELSE [saturated]

PRCARR(1).CONF(N).REPRCHT = -1.0;
PRCARR(1).CONF(N).REPREMT = -1.0;
PRCARR(1).CONF(N).SHAREDFEM = -1.0;
PRCARR(1).CONF(N).SHREDPERM = -1.0;
PRCARR(I).CONF(N).RSRRSM = -1.0;
PRCARR(I).CONF(N).SDRSRS = -1.0;
INFORM(I).CONF(N).SAERCAP = ACAP;
INFORM(I).CONF(N).SDEPCAP = DCAP;
INFORM(I).CONF(N).SAERCAP = 0.0;
INFORM(I).CONF(N).SDEPCAP = 0.0;
ACRCA = ACAP;
BRCAP = BCAP;

END  NORTH_ONLY_CAPACITY_CALCULATIONS;
PROCESS SOUTH ONLY_CAPACITY_CALCULATION
[This process computes capacity for south only configuration]

CALL CAPCAL;

IN (PAIR(2), CAP(2), ATOTPNC);

OUT (ACAP, DCAP);

IF (PFCARR(1).TOTARR + PFCARR(1).TOTDEF) LE (ACAP + DCAP)
THEN [not saturated]

PFCARR(1).CONF(N).MFRCHT = 0.1;
PFCARR(1).CONF(N).MFRCHTI = 0.1;
PFCARR(1).CONF(N).MFRCHM = 0.1;
PFCARR(1).CONF(N).MFRCHV = 0.1;
PFCARR(1).CONF(N).MFRCHW = 0.1;
INFORM(1).CONF(N).MFRCHI = 0.1;
INFORM(1).CONF(N).MFRCHJ = 0.1;
INFORM(1).CONF(N).MFRCHK = 0.1;
INFORM(1).CONF(N).MFRCHL = 0.1;
INFORM(1).CONF(N).MFRCHM = 0.1;
INFORM(1).CONF(N).MFRCHN = 0.1;
INFORM(1).CONF(N).MFRCHO = 0.1;
ARMCAP = ACAP;
DEPCAP = DCAP;

ELSE [saturated]

PFCARR(1).CONF(N).MFRCHT = -1.0;
PFCARR(1).CONF(N).MFRCHTI = -1.0;
PFCARR(1).CONF(N).MFRCHM = -1.0;
PFCARR(1).CONF(N).MFRCHV = -1.0;
PFCARR(1).CONF(N).MFRCHW = -1.0;
INFORM(1).CONF(N).MFRCHI = -1.0;
INFORM(1).CONF(N).MFRCHJ = -1.0;
INFORM(1).CONF(N).MFRCHK = -1.0;
INFORM(1).CONF(N).MFRCHL = -1.0;
INFORM(1).CONF(N).MFRCHM = -1.0;
INFORM(1).CONF(N).MFRCHN = -1.0;
INFORM(1).CONF(N).MFRCHO = -1.0;
ARMCAP = ACAP;
DEPCAP = DCAP;

END SOUTH ONLY_CAPACITY_CALCULATIONS;
PROCESS CONSTRAIN_CAPACITY_OF_ENTIRE_AIRPORT
   [This process constrain capacity for entire airport]
   TOTPRC = ARRCAP/(ARRCAP + DEFCAP); 
   IF ATOTPRC GT TOTPRC
      THEN DEFCAP = (1.0 - ATOTPRC) * ARRCAP/ATOTPRC;
   ELSEIF ATOTPRC LT TOTPRC
      THEN ARRCAP = ATOTPRC * DEFCAP/(1.0 - ATOTPRC);
   INFORM(1),CONF(N).CAPACITY = ARRCAP + DEFCAP; [total airport capacity (constrained)]
END CONSTRAIN_CAPACITY_OF_ENTIRE_AIRPORT;
PROCESS SATURATION COMPUTATION
[This process computes saturation level]

IF PRCARR(I).CONF(N).BMPRET OR 0.
THEN [not saturated]

[for north complex]

IF CAP GT 0.
THEN INFORM(I).CONF(N).HSAT = DEM/CAP;
ELSE INFORM(I).CONF(N).HSAT = 1.0;

[for south complex]

IF CAP GT 0.
THEN INFORM(I).CONF(N).SSAT = DEM/CAP;
ELSE INFORM(I).CONF(N).SSAT = 1.0;


ELSE [saturated]

[for north complex]
IF CAP GT 0.
    THEN INFORM(I).CONF(N).MSAT = DEM/CAP;
    ELSE INFORM(I).CONF(N).MSAT = -1.0;


IF CAP GT 0.
    THEN INFORM(I).CONF(N).ESSAT = DEM/CAP;
    ELSE INFORM(I).CONF(N).ESSAT = -1.0;

        CONF(N).CAPACITY;

END SATURATION_COMPUTATION;
PROCESS  CHANGE_DUE_TO_DEMAND_BALANCING_COMPUTATION
[This process computes changes in demand as result of demand balancing]

IF  PECARR(I).CONF(N).BMPFNT  GE  0.
    THEN  [not saturated]
        INFORM(I).CONF(N).CHANGEMARR = FLOOR(PECARR(I).CONF(N).HARDEM + .5) -
            FLOOR(PECARR(I).CONF(N).BMPFNT + .5);  
        INFORM(I).CONF(N).CHANGEND = FLOOR(PECARR(I).CONF(N).SDEPDEM + .5) -
            FLOOR(PECARR(I).CONF(N).BDEPDEM + .5);  
    ELSE  [saturated]
        INFORM(I).CONF(N).CHANGEMARR = 0.;  
        INFORM(I).CONF(N).CHANGEND = 0.;  
END  CHANGE_DUE_TO_DEMAND_BALANCING_COMPUTATION;

PROCESS  FINAL_SATURATION_CHECK
[This process checks saturation level and set appropriate variables]

    THEN
        PECARR(I).CONF(N).BMPFNT  =  -1.0;  
        PECARR(I).CONF(N).BDFNT  =  -1.0;  
        PECARR(I).CONF(N).HARDEM  =  -1.0;  
        PECARR(I).CONF(N).SDEPDEM  =  -1.0;  
        PECARR(I).CONF(N).BDEPDEM  =  -1.0;  
    END  FINAL_SATURATION_CHECK;
ROUTINE CAPCAL

IN (PNM, CAPFILE, PRCNT);
OUT (ACAP, DCAP);

(This routine computes arrival and departure capacity of a complex based on percentage of arrivals
and a particular capacity curve)

IF (PNM EQ 1) OR (PRCNT EQ 0.)
THEN [one pair of points only, or no arrivals]
ACAP = CAPFILE(1);
DCAP = CAPFILE(2);

ELSEIF PRCNT LE 1.0
THEN [some departures]
FLAG = 0;
R = PRCNT/(1.0 - PRCNT);
RATIO1 = CAPFILE(1)/CAPFILE(2);

REPEAT WHILE (FLAG EQ 0); [I = 2 to PNM]
RATIO1 = RATIO2;
RATIO2 = CAPFILE (2*I-1)/CAPFILE(2*I);

IF (R EQ RATIO1) AND (R LE RATIO2)
THEN
S = (CAPFILE(2*I) _ CAPFILE(2*I-2))/(CAPFILE(2*I-1) _ CAPFILE(2*I-3));
DCAP = (CAPFILE(2*I-2) _ S*CAPFILE(2*I-3))/(1-S*R)
ACAP = R*DCAP;
FLAG = 1;
ENDREPEAT;
IF FLAG NE 0
THEN [lots of departures]
ACAP = CAPPHL (RUNP2-1);  
DCAP = ACAP/R;
ELSE [all arrivals]
ACAP = CAPPHL (RUNP2-1);  
DCAP = 0.;
END CAPCAL;
ROUTINE DBAL


[This routine performs demand balancing]

PHRSM = PAIR;
A = PRECARR(I).TOTARR;
D = PRECARR(I).TOTDEP;
MARRDEN = PRECARR(I).CONF(N).MARRDEN;
MDPDM = PRECARR(I).CONF(N).NDPDM;
ENPROMT = PRECARR(I).CONF(N).ENPROMT;
ESPROMT = PRECARR(I).CONF(N).ESPROMT;
BMARRDEN = PRECARR(I).CONF(N).BMARRDEN;
BMDPDM = PRECARR(I).CONF(N).BMDPDM;

SWITCH(1) = 2;
SWITCH(2) = 1;
SAT = 1.0;
INDEX = 0;
CURVE = 0;

IF (A NE 0.) AND (D NE 0.)
THEN [if both arrival and departure demands are non-zero]

ENPROMT = .5;
ESPROMT = .5;
BMARRDEN = 0;
BMDPDM = 0;

ELSE

LOOP; [X = 1 to 2]

IF PHRSM(X) GT 1
I to 2^PNUM (K) - CAPACITY(K, J) = 0.

IF c1(z * npum(x)) Na 0.
P1 - P1 + 1.
C1(2*P1-1) = CL(2*P1-3);
C1(2*P1) = 0;
P1 = 1;
CM.C) = CAPhCTT(K, I);
C1(2*CAPACIT(K, I)) = 0.

IF CiC 0.0.
OR (C1(2) 0.0*LowE;
RJ = I to 2^PmUS1V12(Z))
C2(j) = CAPACITY(J, 3);

IF C2(2*PNM(SNITU(K))) 30.
P2 = P2 + PUMI(SNITCHM);
ELSE

P2 = 2;

IF CAPACITY(SWITCH(K),1) > 0.

THEN
C2(1) = 0.1;
C2(2) = 0.1;
C2(3) = CAPACITY(SWITCH(K),1);
C2(4) = CAPACITY(SWITCH(K),2);

ELSE
C2(1) = CAPACITY(SWITCH(K),1);
C2(2) = CAPACITY(SWITCH(K),2);
C2(3) = 0.1;
C2(4) = 0.1;

IF (C2(2) < 0.) OR (C2(3) < 0)

THEN

IF K = 1

THEN
ARDSM = HARDSM;
DEPSM = HDEPSM;

ELSE
ARDSM = A - HARDSM;
DEPSM = D - HDEPSM;

CALL END;

IN (CI, C2, P1, P2, A, D, ARDSM, DEPSM);
OUT (ARDSM, DEPSM);

[This routine performs demand balancing algorithm]

IF KROMIN IT SAV
THEN
CORNER = INDEX;
CURVE = k;
SAT = RHOMIN;
ENDLOOP;
[check integer aircraft]
IF CURVE = 0
THEN
BTRNT = -1.0;
BFRONT = -1.0;
BMAIENDM = -1.0;
BMAIEND = -1.0;
BMSIPE = -1.0;
BMSIPE = -1.0;
ELSEIF (CORNER GT PHMN (CURVE)) AND (PHMN(CURVE) EQ 1)
THEN
XARR = SAT * CAPACITY (CURVE,2*PHMN(CURVE)-1);
YDEP = 0.1;
ELSEIF (CORNER GT PHMN (CURVE)) AND (PHMN(CURVE) NE 1)
THEN
XARR = SAT * CAPACITY (CURVE,1);
YDEP = SAT * CAPACITY (CURVE,2);
ELSE
XARR = SAT * CAPACITY (CURVE, 2 * CORNER - 1);
YDEP = SAT * CAPACITY (CURVE, 2 * CORNER);
CAPFILE1 = CAPACITY(CURVE, 0);
CAPFILE2 = CAPACITY(SWITCH(CURVE), 0);
X = FLOAT(FLOOR(XARR + .5));
Y = FLOAT(FLOOR(YDEP + .5));
IF (X + Y) GT 0;
THEN
\[ PCT1 = \frac{x}{x + y}; \]
\[ \text{CALL \ CAPCAL;} \]
\[ \text{IN \ (PRISM(CURVE),CAPFILE1, PCT1);} \]
\[ \text{OUT \ (ACAPI, DCAP);} \]
\[ S1 = \frac{x + y}{(ACAP1 + DCAP1)}; \]
ELSE \[ S1 = 0.0; \]
IF \[ (A - x \neq 0.0) \text{ AND } (D - y \neq 0.0) \]
THEN
\[ PCT2 = \frac{A - x}{A + D - x - y}; \]
\[ \text{CALL \ CAPCAL;} \]
\[ \text{IN \ (PRISM(SWITCH(CURVE)),CAPFILE2, PCT2);} \]
\[ \text{OUT \ (ACAP2, DCAP2);} \]
\[ S2 = \frac{A + D - x - y}{(ACAP2 + DCAP2)}; \]
ELSIF \[ (A - x \neq 0.0) \]
THEN
\[ PCT2 = 1.0; \]
\[ \text{CALL \ CAPCAL;} \]
\[ \text{IN \ (PRISM(SWITCH(CURVE)),CAPFILE2, PCT2);} \]
\[ \text{OUT \ (ACAP2, DCAP2);} \]
\[ S2 = \frac{D - y}{(ACAP2 + DCAP2)}; \]
IF \[ (S1 \neq 1.0) \text{ AND } (S2 \neq 1.0) \]
THEN IF CURVE EQ 1

THEN
RESPLCNT = FCT1;
RESPLCNT = FCT2;
BAARDEM = Y;
BAARDEM = A - X;
ENDFORM = Y;
ENDFORM = D - Y;

ELSE
RESPLCNT = FCT1;
RESPLCNT = FCT2;
BAARDEM = X;
BAARDEM = A - X;
ENDFORM = Y;
ENDFORM = D - Y;

ELSE
PERFORM LOWER_LEFT_LOOKUP;
IF (S1 > 1.0) AND (S2 > 1.0)
THEN PERFORM ASSIGNMENT;
ELSE PERFORM UPPER_LEFT_LOOKUP;
IF (S1 > 1.0) AND (S2 > 1.0)
THEN PERFORM ASSIGNMENT;
ELSE PERFORM UPPER_RIGHT_LOOKUP;
IF (S1 > 1.0) AND (S2 > 1.0)
THEN PERFORM ASSIGNMENT;
ELSE PERFORM LOWER_RIGHT_LOOKUP;
IF ($1 \geq 1.0$) AND ($S_2 \geq 1.0$) THEN PERFORM ASSIGNMENT;
ELSE
  IMPFRONT = -1.0;
  BRFRONT = -1.0;
  BSAIZEM = -1.0;
  ENSDEFORM = -1.0;
  RESALEM = -1.0;
  ESDEFORM = -1.0;
END DRL;
PROCESS ASSIGNMENT

IF CURVE EQ 1

THEN
BSPFRONT = PCT1;
BSPFRONT = PCT2;
EMARKDEN = X1;
EXPAREM = A - X1;
BDSPFREM = Y1;
BDSPFRM = D - Y1;

ELSE
BSPFRONT = PCT1;
BSPFRONT = PCT2;
EMARKDEN = X1;
EXPAREM = A - X1;
BDSPFREM = Y1;
BDSPFRM = D - Y1;

END ASSIGNMENT;
PROCESS LOWER_LEFT_LOOKUP

X = FLOAT(FLOOR(XARR));
Y = FLOAT(FLOOR(YREP));

IF (X + Y) GT 0.
  THEN
    PCT1 = X/(X + Y);
    CALL CAPCAL;
      IN (PMNM(CURVE),CAPFILE1,PCT1);
      OUT (ACAP1, DCAP1);
    S1 = (X + Y)/(ACAP1 + DCAP1);
  ELSE
    S1 = 0.;
  ENDIF A - X GT 0.
  THEN
    PCT2 = (A - X)/(A + D - X - Y);
    CALL CAPCAL;
      IN (PMNM(SWITCH(CURVE)),CAPFILE2,PCT2);
      OUT (ACAP2, DCAP2);
    S2 = (A + D - X - Y)/(ACAP2 + DCAP2);

END LOWER_LEFT_LOOKUP;
PROCESS UPPER_LEFTLOOKUP

$X = \text{float(ceil}(XARR));$

IF $(X + Y) \geq 0.$

THEN

$PCT1 = \frac{X}{X + Y};$

CALL CAPCAL;

IN (PHRM(CURVE), CAPFILE1, PCT1);

OUT (ACAPI, DCAPI);

$S1 = \frac{X + Y}{ACAPI + DCAPI};$

ELSE $S1 = 0.;$

IF $(A - X \not\geq 0.)$

THEN

$PCT2 = \frac{A - X}{A + D - X - Y};$

ELSE $PCT2 = 0.;$

CALL CAPCAL;

IN (PHRM(MNCH(CURVE)), CAPFILE2, PCT2);

OUT (ACAP2, DCAP2);

$S2 = \frac{A + D - X - Y}{ACAP2 + DCAP2};$

END UPPER_LEFTLOOKUP;
PROCESS UPPER_RIGHT_LOOKUP
Y = FLOAT(CELL(THEP));
IF (X + Y) GT 0.
THEN
   PCT1 = X/(X + Y);
   CALL CAPCAL;
   IN (PMEM(CURVE), CAPFILE1, PCT1);
   OUT (ACAP1, DCAP1);
   S1 = (X + Y)/(ACAP1 + DCAP1);
ELSE
   S1 = 0;
IF (A - X GT 0.) AND (B - Y GT 0.)
THEN
   PCT2 = (A-X)/(A + B_X_Y);
   CALL CAPCAL;
   IN (PMEM(DTARC(CURVE)), CAPFILE2, PCT2);
   OUT (ACAP2, DCAP2);
   S2 = (A + B_X_Y)/(ACAP2 + DCAP2);
ELSEIF (A_X) GT 0.
THEN

    PCT2 = 1.0;
    CALL CAPCAL;

    IN (PHER(SMITH(CURVE2)),CAPFILE2, PCT2);
    OUT (ACAP2, DCAF2);
    S2 = (A - X)/(ACAP2 + DCAF2);

ELSE

    PCT2 = 0.1;
    CALL CAPCAL;

    IN (PHER(SMITH(CURVE2)),CAPFILE2, PCT2);
    OUT (ACAP2, DCAF2);
    S2 = (D - Y)/(ACAP2 + DCAF2);

END UPPER_RIGHT_LOOKUP;
PROCESS LOWER_RIGHT_LOOKUP;
    X = FLOAT(FLOOR(XARR));
    IF (X + Y) GT 0.
        THEN PCT1 = X/(X + Y);
            CALL CAPCAL;
            IN (PMEM(CURVE), CCPFILE1, PCT1);
            OUT (ACAP1, DCAPI);
            S1 = (X + Y)/(ACAP1 + DCAPI);
        ELSE S1 = 0;
    IF (D - Y GE 0.)
        THEN PCT2 = (A - X)/(A + D - X - Y);
            ELSE PCT2 = 1.0;
            CALL CAPCAL;
            IN (PMEM(SNITCH(CURVE)), CCPFILE2, PCT2);
            OUT (ACAP2, DCAPI2);
            S2 = (A + D - X - Y)/(ACAP2 + DCAPI2);
    END LOWER_RIGHT_LOOKUP;
ROUTINE RHO

IN (C1, C2, P1, P2, A, A, ARXDEM, DEPDEM)

OUT (RHOMIN, INDEX);

This routine performs demand balancing algorithm

RHOMIN = 1.0;
DELTA = 999999.9;
FLAG = 0.1
INDEX = 0;

LOOP: [J = P1 to 1 by -1]

IF -FLAG EQ 0

THEN K = 1;

ELSE FLAG = 0;

IF C1(2*K-1) GT 0.

THEN RATIO1 = C1(2*K)/C1(2*K-1);

ELSEIF C1(2*K) GT 0.

THEN RATIO1 = 999999;

ELSE RATIO1 = 0.1

REPEAT UNTIL (K EQ P2) OR (FLAG EQ 1);

NUM = C1(2*K+1) - C1(2*K);
DEN = C1(2*K+1) - C1(2*K-1);

IF DEN GT 0.

THEN M = NUM/DEN;

B = C1(2*K) - N*P2*2(K-1);

T = C1(2*K) - M*C1(2*K-1) + B;
IF T GT 0.
THEN R = (D - N*A)/T;
ELSE IF D GT 0.
THEN R = 1.0;
ELSE R = A/C2(2^K+1) + C1(2^J-1));
ELSE T = C1(2^K+1) + C1(2^J-1));
IF T GT 0.
THEN R = A/T;
ELSE IF A GT 0.
THEN R = 1.0;
ELSE R = D/(C1(2^J) + C2(2^K));
RATIO1 = RATIO2;
IF C2(2^K+1) GT 0.
THEN RATIO2 = C2(2^K+2)/C2(2^K+1); 
ELSE RATIO 2 = 9999999.
X = A - R*C1(2^J-1); 
Y = B - R*C1(2^J); 
IF ABS(Y) LT .001 
THEN Y = 0.;
IF ABS(X) GT .0001;
THEN RATIO = Y/X;
ELSE

MUG

- 999999;

1 - 0;

unhar

(U11 a3*101) A

(3*110 IN10)

ma

ILA - Is1

£1831

(a0*C2(2*9-31)

Li

X)

AND

(X

Li

a’OC2(2*K+1)

AS(M~2(2*K)

Li

T)

i

(T

SI

1*C(242))

IlIm

FLAG

- 1;

TUimi

(1 GI) An

(IL

l1.-0)

Oa(L

Ll

ROOM + .01)

TD

I

(C1(2*J) + Cl(2*J-1));

IF A85(1

ROMWua IL

O133611

(AS(M331+

33633

- M3)

LI

33.1*)

m M1*

- ADSCMiD

+ DPM

- MO6;

Js

3.53

K-K+1;

DU-;

35350;
2.3 O'Hare Status Summary Screen

The processing associated with the O'Hare Status Summary Screen is described on pages 2-150 to 2-176.
STRUCTURE MESSAGE MAKER [data structure where current log messages that appear on O'Hare status summary screen are stored]

GROUP TABLE(106) [up to 108 messages can be constructed]

INT TIME [integer signifying time associated with each log message]

CHR MSG [character string of length 80 for each message]

ENDSTRUCTURE;

INT COUNT [an integer signifying number of available messages initialized to zero]

INT IMKEEP [an integer array of size 108; is used as a flag, if 1st element of this array is equal to 1 it implies that 1st log message is newly added, etc.]

STRUCTURE ON_LOADLIST [a structure of pointers, one for each data field on screen used by panel manager for loading and unloading data to and from screen]

GROUP VX

PTR CHIL [pointer for ceiling data field]

PTR VIS [pointer for visibility data field]

GROUP WIND

PTR DIR [pointer for wind direction data field]

PTR VEL [pointer for wind velocity data field]

PTR ARRNUM(3) [pointers for current configuration's arrival runways data fields]

PTR DEPNUM(4) [pointers for current configuration's departure runways data fields]
PTR CAP [pointer for current configuration's capacity data field]
PTR CAPCF [pointer for percentage of highest capacity data field]
PTR SCROLL [pointer for scroll data field]
PTR LOG MSG(13) [pointers for log message data fields]
PTR MSG [pointer for screen message data field]

BITS FENCE [32 bit variable as prescribed by DNS manual initialized to string of (32) '1'B]  
ENDSTRUCTURE;
ROUTINE USTAT

IN (OSTAT, APTSTAT(1), INFORM(1), CNVGRQ (CONFIND(1)), CONFIND(1), EQFLOG, CNVTRQP, WELOG, CNVTRX, SUBFLOG, CNVTRSP);

INOUT (CGMINS, RSTATS);  [This routine prepares information used on O'Hare status summary screen, and stores that information in structure OSTAT]

$POUR = A;
$THREE = 3;

OSTAT.WX.CEIL = APTSTAT(1).WX.CEIL;  [set prevailing ceiling]
OSTAT.WX.VIS = APTSTAT(1).WX.VIS;  [set prevailing visibility]
OSTAT.WIND.DIR = APTSTAT(1).WIND.DIR;  [set wind direction]
OSTAT.WIND.VEL = APTSTAT(1).WIND.VEL  [set wind velocity]

LOOP:  [J = 1 to 3]

OSTAT.ARR(J) = CNVGRQ (CONFIND(1)).ARR_BMV(J);  [set current operating configuration's arrival runways]

ENDLOOP;

LOOP:  [J = 1 to 4]

OSTAT.DEP(J) = CNVGRQ (CONFIND(1)).DEF_BMV(J);  [set current operating configuration's departure runways]

ENDLOOP;

IF INFORM(1).CONF(CONFIND(1)).CAPACITY EQ -1

THEN  [If current operating configuration is ineligible, blank out capacity data field on screen and produce appropriate message]

OSTAT.CAPACITY = (5) ' ';  
OSTAT.PCT_BC = (3) ' ';  
OSTAT.NEC = SUBSTR(OSTAT.NEC, 1, 24) CONCATENATE ' ***CURRENT CONFIGURATION IS INELIGIBLE***'
ELSE

ONSTAT.CAPACITY = F(INFORM(1).CONF(OFIND(1))).CAPACITY,FOUR); [set current operating configuration's capacity; it is obtained after conversion of numerical data to character data]

PERFORM PERCENTAGE_OF_HIGHEST_CAPACITY_CALCULATION;

ONSTAT.PCT_HC = SUBSTR(F(TAB,THR1R),1,3); [numerical value is converted to character data and stored in appropriate variable]

ONSTAT.SCROLL = (4) ' '; [scroll data field on screen is blanked out]

LOOP: [J = 1 to 13]

ONSTAT.LOG_MSG(J) = 80 ' '; [log messages data fields on screen is blanked out]

ENDLOOP;

LOOP: (J = 1 to 100) [initialize MESSAGE MAKER]

MESSAGE MAKER.TABLE(J).MSG = (80) ' ';

MESSAGE MAKER.TABLE(J).TIME = 0;

ENDLOOP;

PERFORM EQUIPMENT_LOG_MESSAGE_GENERATION; [generate log messages from equipment planning log screen]

PERFORM WEATHER_AND_WIND_LOG_MESSAGE_GENERATION; [generate log messages from weather and wind planning log screen]

PERFORM AIRPORT_PLANNING_LOG_MESSAGE_GENERATION; [generate log messages from airport planning log screen]

PERFORM LOG_MESSAGE_SORT [sort messages on time key]

PERFORM FLAG_NEW_MESSAGES; [flag new messages generated in order to highlight them for user's attention on O'Hare status summary screen]

PERFORM OLD_MESSAGE_TABLE_GENERATION; [generate a copy of existing messages in order to compare them later with table of new messages and flag new entries]
ALT1 = ONSTAT.MSG; [create ALT1 and ALT2 to use for 'return to
ALT2 = ONSTAT.SCROLL; [previously stored data' function]

REPEAT UNTIL (ESTATUS NE PF12);
            ONSTAT.MSG = ALT1;
            ONSTAT.SCROLL = ALT2;

            REPEAT UNTIL (ESTATUS NE PF1);
                CALL HSSCREEN;
                IN (ONSTAT,MESSAGE,MAKE,COMMAND,KEEP);
                IMOUT (ESTATUS);
            [this routine controls O'Hare status summary screen]

            ENDREPEAT;

        ENDREPEAT;

END HSTAT;
PROCESS PERCENTAGE_OF_HIGHEST_CAPACITY_CALCULATION

[This process computes percentage of capacity of current operating configuration to highest available capacity]

TAB = 0.;

LOOP: [J = 1 to 73; for 73 possible configurations]

IF INFORM(1).CONF(J).INDEX NE 999
[If Jth configuration is eligible]

THEN IF INFORM(1).CONF(J).CAPACITY GT TAB

THEN TAB = INFORM(1).CONF(J).CAPACITY;
[compute percentage of highest capacity]

ENDDO; [TAB contains highest capacity available]

TAB = INFORM(1).CONF(CONFIND(1)).CAPACITY*100./TAB;

END PERCENTAGE_OF_HIGHEST_CAPACITY_CALCULATION;
PROCESS EQUIPMENT LOG MESSAGE GENERATION

[This process generates appropriate log messages for O'Hare status summary screen from equipment planning log information]

LOOP; 

IF EQPLOG.TABLE(J).NUM NE (3) ' ' [check for an existing message]

THEN [if a message is found, begin constructing part of message containing runway identifier and equipment]

AUX1 = (3) ' ' CONCATENATE EQPLOG.TABLE(J).NUM CONCATENATE (5) ' ';

IF EQPLOG.TABLE(J).EQUIPMENT NE (11) ' ';

THEN AUX1 = AUX1 CONCATENATE EQPLOG.TABLE(J).EQUIPMENT;

IF EQPLOG.TABLE(J).REMARKS NE (39) ' ';

THEN AUX1 = EQPLOG.TABLE(J).REMARKS

IF EQPLOG.TABLE(J).OTS NE (4) ' ';

THEN COUNT = COUNT + 1; [increment message counter]

MESSAGE MAKER.TABLE(COUNT).TIME = CENTERP.TABLE(J).OTS;

[construct message with OTS time]

MESSAGE MAKER.TABLE(COUNT).MSG = (3) ' ' CONCATENATE EQPLOG.TABLE(J).OTS CONCATENATE AUX1 CONCATENATE ' OTS ' CONCATENATE AUX2;

IF EQPLOG.TABLE(J).RTS NE (4) ' ';

THEN COUNT = COUNT + 1; [increment message counter]
MESSAGE MAKER TABLE(COUNT).TIME = CNVTQF TABLE(J).ETS;
[construct message with ETS time]

MESSAGE MAKER TABLE(COUNT).MSG = (J)' ' CONCATENATE EQLOG TABLE(J).ETS CONCATENATE AUX1 CONCATENATE 'ETS ' CONCATENATE AUX2;

END LOOP;

END EQUIPMENT LOG MESSAGE GENERATION;
PROCESS WEATHER AND WIND LOG MESSAGE GENERATION

[This process generates appropriate log messages for O'Hare status summary screen from weather and wind planning log information]

LOOP: \([J = 1 \text{ to } 13; \text{ up to } 13 \text{ weather and wind planning log messages}]

IF WELOG.TABLE(J).TIME NE (4) ' ' [check for an existing message]

THENIP (WELOG.TABLE(J).CEIL NE (5) ' ') OR (WELOG.TABLE(J).VIS NE (5) ' ')

THEN

AUX = ' ';

COUNT = COUNT + 1; [increment message counter]

MESSAGE MAKER.TABLE(COUNT).TIME = CWVTWK.TABLE(J).TIME;

AUX = (3) ' ' CONCATENATE WELOG.TABLE(J).TIME CONCATENATE (3) ' ';

IF ((WELOG.TABLE(J).CEIL NE (5) ' ') AND (WELOG.TABLE(J).VIS NE (5) ' ')) THEN

AUX=AUX CONCATENATE 'HM' CONCATENATE WELOG.TABLE(J).CEIL CONCATENATE (3) ' ' CONCATENATE WELOG.TABLE(J).VIS CONCATENATE (5)

ELSEIF WELOG.TABLE(J).CEIL NE (5) ' ' THEN

AUX=AUX CONCATENATE 'CEIL ' CONCATENATE WELOG.TABLE(J).CEIL CONCATENATE (13) ' ';

ELSE AUX=AUX CONCATENATE 'VIS ' CONCATENATE WELOG.TABLE(J).VIS CONCATENATE (13) ' ';

IF WELOG.TABLE(J).REMARKS NE (35) ' ' THEN

AUX=AUX CONCATENATE WELOG.TABLE(J).REMARKS;

MESSAGE MAKER.TABLE(COUNT).MSG =AUX
[message is constructed with weather information and stored]

IF (WELOG.TABLE(J).DIR NE (3) ' ') OR (WELOG.TABLE(J).VIS (5) ' ')

2-158
THEN
AUX = ' ';
COUNT = COUNT + 1; [increment message counter]
MESSAGE MAKER.TABLE(COUNT).TIME = COUNTER.TABLE(J).TIME;
AUX = (3) ' ' CONCATENATE WLOG.TABLE(J).TIME CONCATENATE (3) ' ';
IF ((WLOG.TABLE(J).DIR NE(5) ' ') AND(WLOG.TABLE(J).VEL NE(5) ' '))
THEN
AUX=AUX CONCATENATE 'WIND ' CONCATENATE WLOG.TABLE(J).DIR CONCATENATE WLOG.TABLE(J).VEL CONCATENATE (7) ' ';
ELSEIF WLOG.TABLE(J).DIR NE(5) ' '
THEN
AUX=AUX CONCATENATE 'DIR ' CONCATENATE WLOG.TABLE(J).DIR CONCATENATE (12) ' ';
ELSE
AUX=AUX CONCATENATE 'VEL ' CONCATENATE WLOG.TABLE(J).VEL CONCATENATE (12) ' ';
IF WLOG.TABLE(J).REMARKS(35) ' '
THEN
AUX=AUX CONCATENATE WLOG.TABLE(J).REMARKS;
MESSAGE MAKER.TABLE(COUNT).MSG=AUX;
[message is constructed with wind information and stored]
ENDLOOP;
END WEATHER AND WIND LOG MESSAGE GENERATION;
PROCESS AIRPORT PLANNING LOG MESSAGE GENERATION

[This process generates appropriate log messages for O'Hare status summary screen from airport planning log information]

LOOP: [J = 1 to 13; up to 13 airport planning log messages]

IF SURFLOG.TABLE(J).TIME NE (4) ' '
THEN IF SURFLOG.TABLE(J).SURF NE (5) ' '
THEN

AUX = ' '
COUNT = COUNT + 1; [increment message counter]
MESSAGE MAKER.TABLE(COUNT).TIME = CURRSPF.TABLE(J).TIME;
AUX = (3) ' ' CONCATENATE SURFLOG.TABLE(J).TIME CONCATENATE(3) ' ' CONCATENATE SURFLOG.TABLE(J).SURF CONCATENATE 125 ' ' CONCATENATE SURFLOG.TABLE(J).SURF CONCATENATE (125) ' '

IF SURFLOG.TABLE(J).REMARKS NE (27) ' '
THEN AUX = AUX CONCATENATE SURFLOG.TABLE(J).REMARKS;
MESSAGE MAKER.TABLE(COUNT).MESSAGE = AUX;
[message is constructed with surface conditions information]

IF SURFLOG.TABLE(J).BEAK NE (5) ' '
THEN

AUX = ' '
COUNT = COUNT + 1; [increment message counter]
MESSAGE MAKER.TABLE(COUNT).TIME = CURRSPF.TABLE(J).TIME;
AUX = (3) ' ' CONCATENATE SURFLOG.TABLE(J).TIME CONCATENATE (3) ' ' CONCATENATE SURFLOG.TABLE(J).BEAK CONCATENATE(5) ' ' CONCATENATE SURFLOG.TABLE(J).BEAK CONCATENATE 125 ' ' CONCATENATE SURFLOG.TABLE(J).BEAK CONCATENATE (125) ' '

IF SURFLOG.TABLE(J).REMARKS NE (27) ' '

...
THEN AUX = AUX CONCATENATE SURFLOG.TABLE(J).REMARKS;
MESSAGE MAKER.TABLE(COUNT).MSG = AUX;

[message is constructed with braking condition information]

IF SURFLOG.TABLE(J).CLOSED NE (6) ‘’
THEN
AUX = ‘1;
COUNT = COUNT + 1; [increment message counter]
MESSAGE MAKER.TABLE(COUNT).TIME = CVTSRF.TABLE(J).TIME;
AUX = (3)’’ CONCATENATE SURFLOG.TABLE(J).TIME CONCATENATE (3)’’ CONCATENATE SURFLOG.TABLE(J).REMARKS CONCATENATE (5)’’ CONCATENATE SURFLOG.TABLE(J).CLOSED CONCATENATE ‘’;

IF SURFLOG.TABLE(J).REMARKS NE (27) ‘’
THEN AUX = AUX CONCATENATE SURFLOG.TABLE(J).REMARKS;
MESSAGE MAKER.TABLE(COUNT).MSG = AUX;

[message is constructed with runway closure information]

IF SURFLOG.TABLE(J).OPEN NE (6) ‘’
THEN
AUX = ‘1;
COUNT = COUNT + 1; [increment message counter]
MESSAGE MAKER.TABLE(COUNT).TIME = CVTSRF.TABLE(J).TIME;
AUX = (3)’’ CONCATENATE SURFLOG.TABLE(J).TIME CONCATENATE SURFLOG.TABLE(J).REMARKS CONCATENATE (5)’’ CONCATENATE SURFLOG.TABLE(J).OPEN CONCATENATE ‘’

IF SURFLOG.TABLE(J).REMARKS NE (27) ‘’
THEN AUX = AUX CONCATENATE SURFLOG.TABLE(J).REMARKS;
MESSAGE MAKER.TABLE(COUNT).MSG = AUX;

[message is constructed with runway opening information]

ENDLOOP;

END AIRPORT PLANNING LOG MESSAGE GENERATION;
PROCESS LOG_MESSAGE_SORT

[This process sorts log messages generated]

LOOP: [J = 1 TO COUNT-1] [sort on time associated with each message]

L = J;

REPEAT WHILE (L GT 0):

IF MESSAGE_MAKER.TABLE(L+1).TIME LT
MESSAGE_MAKER.TABLE(L).TIME

THEN [exchange Lth message with L + 1st message]

TEMP1=MESSAGE_MAKER.TABLE(L).TIME
TEMP2=MESSAGE_MAKER.TABLE(L).MSG;

MESSAGE_MAKER.TABLE(L).TIME=
MESSAGE_MAKER.TABLE(L+1).TIME;
MESSAGE_MAKER.TABLE(L).MSG=
MESSAGE_MAKER.TABLE(L+1).MSG;

MESSAGE_MAKER.TABLE(L+1).TIME=TEMP1;
MESSAGE_MAKER.TABLE(L+1).MSG=TEMP2;

L = L - 1;

ELSE L = 0

ENDREPEAT;

END LOOP

END LOG_MESSAGE_SORT;
PROCESS FLAG NEW MESSAGES
[This process determines which messages are newly added or modified in order to highlight them on O'Hare status summary screen]

\[I_{KEEP} = 0; \text{[initialize } I_{KEEP}\text{]}\]

\[
\text{IF (OLDMES.TABLE}(1).TIME EQ 0) \text{ AND (OLDMES.TABLE}(1).MSG EQ (80) ' ') \text{ THEN}
\]

\[
\text{LOOP; \{J = 1 TO COUNT\}}
\]

\[
I_{KEEP}(J) = 1; \text{[no old messages, all new messages will be highlighted]}\]

\[
\text{ENDLOOP}.
\]

\[\text{ELSE} \text{[new message is compared with old message table and any new entry is flagged]}\]

\[
I_{KEEP} = 0;
\]

\[
L = 1; \text{[initialization]}\]

\[
J = 1; \text{[initialization]}\]

\[
I_{NEW} = 0;\]

\[
\text{REPEAT WHILE (J LE COUNT)}
\]

\[
\text{IF MESSAGE MAKER.JPGLE(J).TIME LT OLDMES.JPGLE(L).TIME \text{ THEN}} \text{[if times of messages are not equal, there exist a new message]}
\]

\[
I_{NEW}=I_{NEW} + 1;
\]

\[
I_{KEEP}(J) = 1;
\]

\[
J = J + 1;
\]

\[
\text{ELSEIF MESSAGE MAKER.JPGLE(J).TIME EQ OLDMES.JPGLE(L).TIME;} \text{ THENIF}
\]

\[
\text{ENDIF}
\]

\[
\text{MESSAGE MAKER.JPGLE(J).MSG NE OLDMES.JPGLE(L).MSG}
\]

\[
\text{ENDIF}
\]
THEN
  IND-IND *1;
  IKEEP(I)=1;
  J = J + 1;
ELSE
  L = L + 1;
  J = J + 1;
ELSE
  L = L + 1;
ENDREPEAT;

END FLAG_NEW_MESSAGES;
PROCESS OLD_MESSAGE_TABLE_GENERATION

[This process copies contents of MESSAGE MAKER into OLDMES constructing a copy of current messages to be used on next cycle as old message table]

LOOP; [J = 1 TO COUNT; a new 'OLD MESSAGE' table is created]

OLDMES.TABLE(J).TIME = MESSAGE MAKER.TABLE(J).TIME;
OLDMES.TABLE(J).MSG = MESSAGE MAKER.TABLE(J).MSG;

ENDLOOP;

END OLD_MESSAGE_TABLE_GENERATION;
ROUTINE NSCREEN

IN (ONSTAT, MESSAGE MAKER, COUNT, INKEEP);

INPUT (STATUS);  
[This routine controls O'Hare status summary screen]

CHR PANEL [character variable of length 8 containing name of DNS panel initialized to 'ONSTATUS'; name of panel that controls O'Hare status summary screen]

INT CURSOR [integer variable containing cursor's position on screen]

BYTE DM(28) [8 bit variable of data mask used in DNS]

INT CONVERT SCROLL [numerical value of scroll data field]

INT TM [integer representing current time]

DM = USD; [set data mask to default intensity (normal)]

DM(28) = USH; [set message data mask to high intensity]

CURSOR = 14; [set cursor to position 14 (on scroll data field]

DELTA = MIN(COUNT, 10); [only up to 10 messages can be displayed at one time DELTA is number of messages to be displayed]

PERFORM SET UP SCREEN POINTERS (STATUS);

TAU = DM; [current time is obtained]

PERFORM CHARACTER TO NUMERICAL CONVERSION OF TIME;

PERFORM DIVISION OF LOG MESSAGES BASED ON CURRENT TIME;

REPEAT UNTIL (STATUS NE ENTER);

IF COUNT NE 0 [if there are log messages]
THEN
PERFORM SCROLL_FUNCTION_SET_UP;
PERFORM SET_UP_SCREEN_LOG_MESSAGES;
ELSE
PERFORM SET_UP_NULL_SCREEN_MESSAGE;
PERFORM DISPLAY_PANEL;
IF ESTATUS EQ ENTER
THEN
DM = FLDDEF; [set data masks to default intensity (normal)]
DM(28) = FLMHIGH; [set message data mask to high intensity]
CALL NCHECK;
INPUT (OSTAT, CONVERT_SCROLL);
[This routine checks for errors occurred on screen as a result of erroneous entry and returns an appropriate screen message advising user with corrections]
IF OSTAT.MSG NE 'DATA ENTERED'
THEN
DM(CURSOR)=FLMHIGH;
[highlight erroneous entry]
CONVERT_SCROLL = 0;
ELSE
OSTAT.SCROLL = (4)'; [Scroll data field is blanked]
OSTAT.MSG = 'DATA ENTERED AT ' CONCATENATE GMT;
[time is adjusted accordingly on screen message line]
ENDREPEAT;
END MESSAGE;
PROCESS SET_UP_SCREEN_POINTERS (STAT)
[This process sets up screen pointers for DNS use]

ON_LOADLIST.WX.CEIL = ADDR(STAT.WX.CEIL);
ON_LOADLIST.WX.VIS = ADDR(STAT.WX.VIS);
ON_LOADLIST.WIND.DIR = ADDR(STAT.WIND.DIR);
ON_LOADLIST.WIND.VEL = ADDR(STAT.WIND.VEL);

LOOP: [J = 1 TO 3]
    ON_LOADLIST.AERR(J) = ADDR(STAT.AER(J));
ENDLOOP;

LOOP: [J = 1 TO 4]
    ON_LOADLIST.AEPR(J) = ADDR(STAT.AEP(J));
ENDLOOP;

ON_LOADLIST.CAP = ADDR(STAT.CAPACITY);
ON_LOADLIST.CAPCY = ADDR(STAT.CAPCY);
ON_LOADLIST.SCROLL = ADDR(STAT.SCROLL);

LOOP: [J = 1 TO 13]
    ON_LOADLIST.LOG.MSG(J) = ADDR(STAT.LOG.MSG(J));
ENDLOOP;

ON_LOADLIST.MSG = ADDR(STAT.MSG);

END SET_UP_SCREEN_POINTERS (STAT);
PROCESS CHARACTER TO NUMERICAL_CONVERSION_OF_TIME
(This process converts time from character to numerical data)

Get STRING (TAUX) edit (TM); Get STRING a PL/I statement converts character data to specified format of numerical data

END CHARACTER_CONVERSION_OF_TIME_TO_NUMERICA;
PROCESS DIVISION OF LOG_MESSAGES_BASED_ON_CURRENT_TIME

[This routine divides log messages into two segments: past and future based on current running time]

IF MESSAGE_MAKER.TABLE(1).TIME GT TM

THEN LL=0; [LL is number of messages before current time]

ELSE

REPEAT WHILE (MESSAGE_MAKER(J).TIME LE TM) [J = 1 TO COUNT]

LL = J;

ENDREPEAT;

LM = COUNT-LL; [LM is number of messages after current time]

R = MIN (5,LL);

E = MIN (5,LM);

IF (E + R) LT DELTA

THEN IF R LT 5

THEN

R = DELTA-R;

IF R LT 5

THEN R = DELTA-R;

IF LL R

THEN INDEX = LL-R + 1; [INDEX points to number of first message to be displayed]

ELSE INDEX = 1;

NEXT = LL + R; [NEXT points to number of last message to be displayed]

END DIVISION_OF_LOG_MESSAGES_BASED_ON_CURRENT_TIME;
PROCESS SCROLL_FUNCTION SET UP

[This process performs scrolling function associated with O'Hara status summary screen by setting up pointers to first and last messages to appear on screen at one time]

INDEX = INDEX + CONVERT_SCROLL; [scroll number is added in]
NEXT = NEXT + CONVERT_SCROLL;

IF NEXT LT 1 [since negative scroll number is permitted]
    THEN
        M = 1; [pointers are set]
        L = 1;
        NEXT = 1;
        INDEX = NEXT-DELTA + 1;
    ELSEIF INDEX GT COUNT
        THEN
            M = COUNT; [pointers are set]
            L = COUNT;
            INDEX = COUNT;
            NEXT = NEXT + DELTA - 1;
        ELSEIF (NEXT LE COUNT) AND (INDEX GE 1)
            THEN
                M = INDEX;
                L = NEXT;
        ELSEIF (INDEX LT 1) AND (NEXT GE 1)
PROCESS SET_UP_SCREEN_LOG_MESSAGES

[This process determines screen messages to be displayed based on pointers calculated before, and sets up pointers for IMS use; and constructs message headings]

FLAG = '0';
K = 1;

IF MESSAGE MAKER.TABLE(J).TIME LE TN
[if first message time is less than current time]

THEN

ONSTAT.LOG.MSG(K) = (22)' ; CONCATENATE
'***RECENT CHANGES FROM' CONCATENATE SUBSTR( MESSAGE MAKER.TABLE(1).MSG,3,5) CONCATENATE 
' ; [set up header]

ON_LOADLIST.LOG.MSG(K) = ADDR(ONSTAT.LOG.MSG(K));
[set up pointer for header for IMS use]

ELSE

ONSTAT.LOG.MSG(K) = 19' ; CONCATENATE '***EXPECTED CHANGES THROUGH' CONCATENATE
SUBSTR(MESSAGE MAKER.TABLE(COUNT).MSG,3,5) CONCATENATE ' ; [set up header]

ON_LOADLIST.LOG.MSG(K) = ADDR(ONSTAT.LOG.MSG(K));
[set up pointer for header for IMS use]

FLAG = '1';

REPEAT WHILE (K LE 13); [J = K TO L]

K = K + 1;

IF MESSAGE MAKER.TABLE(J).TIME LE TN

THEN

IF INKEEP(J) EQ 1 [if it is a new message]
THEN DM(K+14) = FLDMIGH;

[highlight message]
ONSTAT.LOG_MSG(K) = MESSAGE MAKER.TABLE(J).MSG;
ON_LOADLIST.LOG_MSG(K) = ADDR(ONSTAT.LOG_MSG(K));

ELSE

IF FLAG Eq 'G' B

THEN

DM(K+14) = FLDMARK; [darken screen]
K = K+1;
ONSTAT.LOG_MSG(K) = (19)' CONCATENATE ' ******EXPECTED CHANGES THROUGH' CONCATENATE SUBSTR(MESSAGE MAKER.TABLE(COUNT).MSG,1,1) CONCATENATE ' ******';
ON_LOADLIST.LOG_MSG(K) = ADDR(ONSTAT.LOG_MSG(K));
FLAG='1'8;
K = K +1;

IF INKEEP(J) EQ 1 [if it is a new message]

THEN DM(K+14) = FLDMIGH; [highlight message]
ONSTAT.LOG_MSG(K) = MESSAGE MAKER.TABLE(J).MSG;
ON_LOADLIST.LOG_MSG(K) = ADDR(ONSTAT.LOG_MSG(K));
K = K +1;

REPEAT WHILE (K LE 13) [if number of messages are less than 10, darken rest of screen]

DM(K+14)=FLDMARK;
K = K +1;

ENDREPEAT;

END SET_UP SCREEN LOG MESSAGES;
PROCESS SET UP NULL SCREEN MESSAGE
    [This process issues a message indicating there are no log messages]
    ONSTAT.LOG_MSG(1)="(21)" CONCatE "*****no log entries have been made*****";
    ON_LOADLIST.LOG_MSG(1)=ADD(ONSTAT.LOG_MSG(1));
END SET UP NULL SCREEN MESSAGE;

PROCESS DISPLAY PANEL
    [This process invokes PDISPLeY preprocessor which in turn interfaces with DMS panel manager and displays screen]
END DISPLAY PANEL;
ROUTINE HCHECK

INPUT (ONSTAT, CONVerT Scroll);

[This routine checks for errors occurred on screen as a result of erroneous entry and returns an
appropriate screen message assisting user with corrections]

DECIMAL = ".";
ERR1 = 'NUMERIC INPUT REQUIRED'
ERR3 = 'NO DECIMAL POINTS ALLOWED'
NOERR = 'DATA ENTERED';
ONSTAT.MSG = NOERR;

ON CONVERSION BEGIN;

ONSTAT.MSG = ERR1;

RETURN;

Get STRING (ONSTAT.SCROLL) EDIT (CONVerT Scroll);
[conversion to numerical data from character data]

IF VERIf{ (DECIMAL, ONSTAT.SCROLL) = 0 

THEN ONSTAT.MSG = ERR3;

END HCHECK;
2.4 Planning Log Selection Screen

The processing associated with the Planning Log Selection Screen is shown on pages 2-178 to 2-185.
**LOCAL VARIABLES**

STRUCTURE LOGNUM [planning log selection screen information]

CHR CHOICE (4) [character representation (length 2) of selection choices pertaining to 4 possible log screens]

CHR MSG [character variable (length 80) reserved for screen messages]

ENDSTRUCTURE;

STRUCTURE LOG_LOADLIST [a structure of pointers, one for each data field on screen used by panel manager for loading and unloading data to and from screen]

PTR CHOICE (4) [pointers for choice field]

PTR MSG [pointer for message data field]

BITB FENCE [32 bit variable as prescribed by DMS manual initialized to string of (32) '1's]

ENDSTRUCTURE;
ROUTINE LOGS

INOUT (RSTATUS);

[This routine invokes planning log selection screen]

[Initialize screen to blanks]

LOGNUM.CHOICE (1) = '1';
LOGNUM.CHOICE (2) = '2';
LOGNUM.CHOICE (3) = '3';
LOGNUM.CHOICE (4) = '4';
LOGNUM.NSC = (80) '1';

REPEAT UNTIL (RSTATUS NE PF12)

LOGNUM_DATA = LOGNUM;
REPEAT UNTIL (RSTATUS NE PF2)

CALL LSCEEN;

INOUT (LOGNUM_DATA, RSTATUS);

[This routine controls planning log selection screen]

ENDREPEAT;

ENDREPEAT;

END LOGS;
ROUTINE LSSCREEN

INPUT (LOGNUM_DATA, ESTATUS);
 [This routine controls planning log selection screen]

CHRN FRAME [character variable of length 8 containing name of DNS panel initialized to 'LOGGR',
   name of panel that controls planning log selection screen]

INT CURSOR [integer variable containing cursor's position on screen]

BITS DM(5) [8-bit variable of data masks used in DNS]

PERFORM SET_UP_SCREEN_POINTERS (LOGS);

DM-FLDDEF; [set data masks to default intensity (normal)]

DM(5) = FLDHIGH; [set message data mask to high intensity]

CURSOR = 1; [set cursor to position 1]

REPEAT UNTIL (ESTATUS NE ENTER)

PERFORM DISPLAY_PANEL

IF ESTATUS EQ FAL

THEN stop;

IF ESTATUS NE ENTER

THEN;

ELSE

   DM = FLDDEF; [set data masks to default intensity]

   DM (5) = FLDHIGH; [set message data mask to high intensity]

   CALL LCHECK;
INOUT (LOGNUM_DATA);
OUT (CURSOR);

This routine checks for errors occurred on screen as a result of erroneous entry and returns value for cursor pointing to first data field where an error has occurred; an appropriate screen message is issued advising user with corrections.

IF LOGNUM_DATA.$MN$ 'DATA ENTERED'
    THEN DM(CURSOR) = FLMHIG; [highlight erroneous entry]
    ELSE
        CALL LVALID;

        INOUT (LOGNUM_DATA);
        OUT (CURSOR);

        This routine performs data validation checks on screen entries and returns value for cursor pointing to first invalid data field. Also, an appropriate screen message is issued advising user with corrections.

        IF LOGNUM_DATA.$MN$ 'ERROR
            THEN DM(CURSOR) = FLMHIG; [highlights invalid entry]
            ELSE
                CALL LUPDATE;

                INOUT (LOGNUM_DATA, ESTATUS);

                This routine is performed only when there are no errors committed on screen, it updates appropriate variables pertaining to this screen program with new screen entries. Also, returns new value of ESTATUS.

ENDREPEAT
END LOCScreen
SOFTWARE DESCRIPTION FOR THE O'HARE RUNWAY
CONFIGURATION MANAGEMENT SYSTEM (U) MITRE CORP MCLEAN
VA METREK DIV S KAVOUSSI OCT 82 MTR-82W125-VOL-2
UNCLASSIFIED FAA-EM-82-28-VOL-2 DFSA01-81-C-10003 F/G 17/7 NL
PROCESS SET_UP SCREEN POINTERS (LOGS)
[This process sets up screen pointers for DMS use]

LOOP: [J = 1 to 4];

LOG_LOADLIST.CHOICE (J) = ADDR (LOGMEM_DATA.CHOICE(J));

ENDLOOP;

LOG_LOADLIST.MSG = ADDR (LOGMEM_DATA.MSG);

END SET_UP_SCREEN_POINTERS (LOGS);
**ROUTINE LCHECK**

**INOUT** (LOGNUM_DATA);

**OUT** (CURSOR);

[This routine checks for errors occurred on screen as a result of erroneous entry and returns value for cursor pointing to first data field where an error has occurred; and an appropriate screen message is issued advising user with corrections]

LOGNUM_DATA = 'DATA ENTERED';

**REPEAT WHILE** (LOGNUM_DATA.MSG EQ 'DATA ENTERED'); (J = 1 to 4)

CURSOR = J

**IF** X (LOGNUM_DATA.CHOICE(J)) NE 0

THEN LOGNUM_DATA.MSG = 'INPUT MUST BE X OR BLANK'

ENDREPEAT;

**IF** LOGNUM_DATA.MSG = 'DATA ENTERED'

THEN CURSOR = 1; (If no error is detected cursor is put on first data field on screen)

END LCHECK;
ROUTINE LVALID

INPUT (LOGNUM_DATA);

OUT (CURSOR);

[This routine performs data validation checks on screen entries and returns value for cursor pointing to first invalid data field. Also, an appropriate screen message is issued advising user with corrections]

LOGNUM_DATA.HSG = 'DATA ENTERED';

FLAG = 0;

REPEAT WHILe (FLAG LT 2) [J = 1 to 4]

CURSOR = J;

IF LOGNUM_DATA.CHOICE(J) NE ''

THEN FLAG = FLAG + 1;

END REPEAT;

IF FLAG EQ 2

THEN LOGNUM_DATA.HSG = 'SELECT ONLY ONE PLANNING LOG';

ELSE CURSOR = 1;

END LVALID;
ROUTINE UPDATE

INPUT (LOGNUM_DATA, ESTATUS)

[This routine is performed only when there are no errors committed on screen, it updates
appropriate variables pertaining to this screen program with new screen entries. Also, returns new
value of ESTATUS]

IF LOGNUM_DATA.CHOICE(1) NE ''
    THEN ESTATUS = FF13; [weather and wind planning log is selected]
ELSE LOGNUM_DATA.MSG = (80) ' ' [message line is blanked]

IF LOGNUM_DATA.CHOICE(2) NE ''
    THEN ESTATUS = FF14; [airport planning log is selected]

IF LOGNUM_DATA.CHOICE(3) NE ''
    THEN ESTATUS = FF15; [equipment planning log is selected]

IF LOGNUM_DATA.CHOICE(4) NE ''
    THEN ESTATUS = FF16; [demand planning log is selected]
END UPDATE;

2.5 Weather and Wind Planning Log Screen

The following pages, 2-187 to 2-206, describe the processing associated with the Weather and Wind Planning Log Screen.
**LOCAL VARIABLES**

STRUCTURE WX_DATA LIKE MELOG
   [This structure is similar to MELOG used as a working area within screen routine]
ENDSTRUCTURE;

STRUCTURE: CNVRT WX LIKE CNVTNX
   [This structure is similar to CNVTNX used as working area within screen routine]
ENDSTRUCTURE;

STRUCTURE WX_LOADLIST (a structure of pointers, one for each data field on screen used by panel manager for loading and unloading data to and from screen)

GROUP TABLE(13)
   PTR TIME [pointer for time data field]
   PTR CEIL [pointer for ceiling data field]
   PTR VIS [pointer for visibility data field]
   PTR DIR [pointer for direction of wind data field]
   PTR VRL [pointer for velocity of wind data field]
   PTR REMARKS [pointer for remarks data field]
   PTR MSG [pointer for screen message data field]

BITF FENCE [32 bit variable as prescribed by IMS manual, initialised to string of (32)'1'B]
ENDSTRUCTURE;
ROUTE WLOG

INPUT (WXLOG, CWFTWX, ESTATUS);
[This routine invokes weather and wind planning log screen]

REPEAT UNTIL (ESTATUS NE E12);
   WX_DATA = WXLOG;
   CWFTWX = CWFTWX;
   CALL WSCREEN;

   INPUT (WX_DATA, CWFTWX, ESTATUS);
   [This routine controls weather and wind planning log screen]

ENDREPEAT;

IF SUBSTR (WX_DATA, MSG, 1, 12) EQ 'DATA ENTERED'
  THEN
      WXLOG = WX_DATA;
      CWFTWX = CWFTWX;
  END WLOG;
ROUTINE WSSCREEN

IMPORT (MX_DATA,CONVEX MX,ESTATUS)

[This routine controls weather and wind planning log screen]

CHAR PHASE

[character variable of length 8 containing name of DMS panel initialised to 'APLOG1',
name of panel that controls weather and wind planning log screen]

INT CURSOR

[integer variable containing cursor's position on screen]

BITS DM(79)

[8 bit variable of data masks used in DMS]

structure AUX_DATA LIKE MX_DATA

ENDSTRUCTURE;

DM = FLODEF; [set data masks to default intensity (normal)]

DM(79) = FLHIGH; [set message data mask to high intensity]

CURSOR = 61; [set cursor to position 61; first data field used]

AUX_DATA = MX_DATA;

PERFORM SET_UP_SCREEN_POINTERS (WLOG);

REPEAT UNTIL (ESTATUS NE ENTER);

PERFORM DISPLAY_PANEL;

IF ESTATUS EQ PA1

THEN stop;

IF ESTATUS NE ENTER

THEN NEW_DATA = AUX_DATA;
ELSE

DM = FLDOF;  
DM(78) = FLDOF; 

CALL WCHECK

INOUT (UX_DATA,CWRT_XZ, CURSOR); 
[This routine checks for errors occurred on screen as a result of an erroneous entry and returns value for cursor pointing to first data field where an error has occurred; and an appropriate screen message is issued advising user with corrections] 

IF UX_DATA,MSG NE 'DATA ENTERED' 
THEN DM(CURSOR) = FLDOF; 
ELSE 

CALL WVALID; 

INOUT (UX_DATA,CWRT_XZ, CURSOR); 
[This routine performs data validation checks on screen entries and returns value for cursor pointing to first invalid data field. Also, an appropriate screen message is issued advising user with corrections]

IF UX_DATA,MSG NE 'DATA ENTERED' 
THEN DM(CURSOR) = FLDOF; 
ELSE 

CALL WUPDATE; 

INOUT (UX_DATA,CWRT_XZ); 
[This routine is performed only when there are no errors committed on screen; it sorts log entries on screen based on time] 

UX_DATA,MSG = 'DATA ENTERED AT' CONCATENATE GMT; 
AUX_DATA = UX_DATA; 

ENDREPEAT; 

END WSCREEN;
PROCESS SET_UP_SCREEN_POINTERS (LOG)
[This process sets up screen pointers for IMS use]

LOOP: [J = 1 TO 13]

MY_LOADLIST_TABLE(J).TIME=ADDR(MY_DATA_TABLE(J).TIME);  
MY_LOADLIST_TABLE(J).CHR=ADDR(MY_DATA_TABLE(J).CHR);  
MY_LOADLIST_TABLE(J).VIS=ADDR(MY_DATA_TABLE(J).VIS);  
MY_LOADLIST_TABLE(J).DIR=ADDR(MY_DATA_TABLE(J).DIR);  
MY_LOADLIST_TABLE(J).VEL=ADDR(MY_DATA_TABLE(J).VEL);  
MY_LOADLIST_TABLE(J).REMARKS=ADDR(MY_DATA_TABLE(J).REMARKS);  

ENDLOOP:

MY_LOADLIST_MSG=ADDR(MY_DATA_MSG);

END SET_UP_SCREEN_POINTERS (LOG);
Routine **WCHECK**

**Input** (WX_DATA, CVNT WX, CURSOR);

[This routine checks for errors occurred on screen as a result of an erroneous entry and returns value for cursor pointing to first data field where an error has occurred; and an appropriate screen message is issued advising user with corrections]

**ERR1 = 'NUMERIC INPUT REQUIRED';**
**ERR2 = 'NON-NEGATIVE INPUT REQUIRED';**
**ERR3 = 'NO DECIMAL POINTS ALLOWED';**
CURSOR = 60;
WX_DATA.MSG = 'DATA ENTERED';

**ON CONVERSION BEGIN:**

[ON CONVERSION is a PL/I feature, it is invoked if a character data is detected in a numerical data field]

WX_DATA.MSG = ERR1;

RETURN;

**Repeat While** (WX_DATA.MSG EQ 'DATA ENTERED'); [J = 11 TO 13]

CURSOR = CURSOR + 1;

PERFORM TIGN_DATA_FIELD_ERROR_CHECK;
EXIT IF [errors detected]
CURSOR = CURSOR + 1;

PERFORM CENL_DATA_FIELD_ERROR_CHECK;
EXIT IF [error detected]
CURSOR = CURSOR + 1;

PERFORM VIN_DATA_FIELD_ERROR_CHECK;
EXIT IF [error detected]
CURSOR = CURSOR + 1;
PERFORM DIR_DATA_FIELD_ERROR_CHECK;
EXITIF [error detected]
CURSOR = CURSOR + 1
PERFORM VEL_DATA_FIELD_ERROR_CHECK;
EXITIF [error detected]
CURSOR = CURSOR + 1;
ENDREPEAT;
END WCHECK;

PROCESS TIME_DATA_FIELD_ERROR_CHECK
[This process checks for errors on time data field]
Get String(WX_DATA.TABLE(J).TIME) EDIT(CHVRT_WX.TABLE(J).TIME); [conversion from character data to numerical data]

IF VERIFY('=', WX_DATA.TABLE(J).TIME) EQ 0
   THEN WX_DATA.MSG = ERR2;
   ELSEIF VERIFY('=', WX_DATA.TABLE(J).TIME) EQ 0
      THEN WX_DATA.MSG = ERR3;
END TIME_DATA_FIELD_ERROR_CHECK;

PROCESS CEIL_DATA_FIELD_ERROR_CHECK
[This process checks for errors on ceiling data field]
Get String(WX_DATA.TABLE(J).CEIL) EDIT(CHVRT_WX.TABLE(J).CEIL) [conversion from character data to numerical data]

IF VERIFY('=', WX_DATA.TABLE(J).CEIL) EQ 0
   THEN WX_DATA.MSG = ERR2;
   ELSEIF VERIFY('=', WX_DATA.TABLE(J).CEIL) EQ 0
      THEN WX_DATA.MSG = ERR3;
END CEIL_DATA_FIELD_ERROR_CHECK;
PROCESS VIS_DATA_FIELD_ERROR_CHECK
    [This process checks for errors on visibility data field]
    Get STRING (WX_DATA.TABLE(J).VIS) EDIT (CHVRT_WX.TABLE(J).VIS)
    [conversion from character data to numerical data]
    IF VERIFY('-', WX_DATA.TABLE(J).VIS) EQ 0
    THEN WX_DATA.MSG = ERR2;
    ELSEIF VERIFY('.', WX_DATA.TABLE(J).VIS) EQ 0
    THEN WX_DATA.MSG = ERR3;
    END VIS_DATA_FIELD_ERROR_CHECK;

PROCESS DIR_DATA_FIELD_ERROR_CHECK
    [This process checks for errors on wind direction data field]
    Get STRING (WX_DATA.TABLE(J).DIR) EDIT (CHVRT_WX.TABLE(J).DIR)
    [conversion from character data to numerical data]
    IF VERIFY('-', WX_DATA.TABLE(J).DIR) EQ 0
    THEN WX_DATA.MSG = ERR2;
    ELSEIF VERIFY('.', WX_DATA.TABLE(J).DIR) EQ 0
    THEN WX_DATA.MSG = ERR3;
    END DIR_DATA_FIELD_ERROR_CHECK;
PROCESS VEL_DATA_FIELD_ERROR_CHECK
[This process checks for errors on wind velocity data field]
Get STRING (WX_DATA_TABLE(J).VEL) EDIT (CONVERT WX_TABLE(J).VEL)
[conversion from character data to numerical data]

IF VERIFY ('-', WX_DATA_TABLE(J).VEL) EQ 0
THEN WX_DATA.MSG = ERR2;
ELSEIF VERIFY ('.', WX_DATA_TABLE(J).VEL) EQ 0
THEN WX_DATA.MSG = ERR3;
END VEL_DATA_FIELD_ERROR_CHECK;
ROUTINE WVALID

INOUT (WX_DATA,CWVET_WX,CURSOR);

This routine performs data validation checks on screen entries, and returns value for cursor pointing to first invalid data field. Also, an appropriate screen message is issued advising user with corrections.

$FOUR = 4;
$FIVE = 5;
WX_DATA = 'DATA ENTERED';

REPEAT WHILE (WX_DATA.MSG EQ 'DATA ENTERED');{J = 11 TO 13}

IF (WX_DATA.TABLE(J).TIME EQ (4) ' ' ) AND
(WX_DATA.TABLE(J).CRIL EQ (5) ' ' ) AND
(WX_DATA.TABLE(J).VIS EQ (5) ' ' ) AND
(WX_DATA.TABLE(J).DIR EQ (5) ' ' ) AND
(WX_DATA.TABLE(J).VEL EQ (5) ' ' ) AND
(WX_DATA.TABLE(J).REMARKS EQ (35 ' ' )

THEN [all entries blank]
CURSOR = CURSOR + 6;
CWVET_WX.TABLE(J).TIME = 9999;
ELSE

CURSOR = CURSOR + 1;

IF (WX_DATA.TABLE(J).TIME NE (4) ' ' ) AND
(WX_DATA.TABLE(J).CRIL EQ (5) ' ' ) AND
(WX_DATA.TABLE(J).VIS EQ (5) ' ' ) AND
(WX_DATA.TABLE(J).DIR EQ (5) ' ' ) AND
(WX_DATA.TABLE(J).VEL EQ (5) ' ' ) AND
(WX_DATA.TABLE(J).REMARKS EQ (35 ' ' )

THEN [entries other than time are missing]
WX_DATA.MSG = 'ADDITIONAL INFORMATION REQUIRED FOR THIS TIME ENTRY';
ELSE
IF (VI_DATA.TA3LE(J).TIME EQ (4)' ') AND
(VI_DATA.TA3LE(J).CZIL NE (5)' ') OR
(VI_DATA.TA3LE(J).VI3 NT (5)' ') OR
(VI_DATA.TA3LE(J).DIR NE (5)' ') OR
(VI_DATA.TA3LE(J).REMARES NE (39)' ''))
THEN [time missing]
   VI_DATA.MS3G = 'SPECIFY TIME ASSOCIATED WITH ENTRIES';
ELSE [check time]
   PERFORM TIME_CHECK;
   EXITIF [time entry is erroneous]
   PERFORM LEFT_ZERO_PADDING ON TIME_ENTRY;
   IF VI_DATA.MS3G EQ 'DATA ENTERED'
      THEN
         CURSOR = CURSOR + 1;
         IF VI_DATA.TA3LE(J).CZIL NE (5)' '
            THEN
               PERFORM RIGHT_JUSTIFY_CZIL_DATA_ENTRY;
               CURSOR = CURSOR + 1;
            IF VI_DATA.TA3LE(J).VI3 NE (5)' '
               THEN
                  PERFORM RIGHT_JUSTIFY_VI3_DATA_ENTRY;
                  CURSOR = CURSOR + 1;
               IF VI_DATA.TA3LE(J).DIR NE (5)' '
                  THEN
                     PERFORM RIGHT_JUSTIFY_DIR_DATA_ENTRY;
                     CURSOR = CURSOR + 1;
                  IF VI_DATA.TA3LE(J).REMARES NE (39)' ''
                     THEN
THEN IF CHNVRT_WX_TABLE(J).DIR GT 360.0
  THEN
    WX_DATA_MSG = 'WIND DIRECTION MUST NOT EXCEED 360 DEGREES'
  ELSE
    PERFORM LEFT_ZERO_PADDING_ON_WIND
      DIRECTION_ENTRY;
    IF WX_DATA_MSG EQ 'DATA ENTERED'
      THEN
        CURSOR = CURSOR + 1;
        IF WX_DATA_TABLE(J).VEL
          THEN PERFORM RIGHT
            JUSTIFY_VEL_DATA
              ENTRY;
            CURSOR = CURSOR + 1;
            PERFORM LEFT
              JUSTIFY
              REMARKS_DATA
              ENTRY
      ENDREPEAT;
    IF WX_DATA_MSG EQ 'DATA ENTERED'
      THEN CURSOR = 61;
  END IF;
PROCESS TIME_CHECK
[This process checks validity of time entry]
HOUR = FLOOR(FLOAT(CHVRV_WX_TABLE(J).TIME)/100.0);
IF HOUR GT 23 [if hour portion is greater than 23]
THEN WX_DATA.MSG = 'HOUR MUST NOT EXCEED 23';
ELSE
MIN = CHVRV_WX_TABLE(J).TIME - HOUR * 100
IF MIN GT 59 [if minute portion is greater than 59]
THEN WX_DATA.MSG = 'MINUTES MUST NOT EXCEED 59';
END TIME_CHECK;
PROCESS LEFT ZERO PADDING ON TIME ENTRY;

This process pads time entry with leading zeroes

C = F (FLOAT(CNVRT=./TABLE(J).TIME), $FOUR);
K = 0;
FILL = '0' B;
FLAG = '0' B;
REPEAT WHILE (FLAG EQ '0' B) REPEAT UNTIL (K EQ 4);
K = K + 1;
IF SUBSTR (C,K,1) EQ '

THEN FILL = FILL CONCATENATE '0';
ELSE FLAG = '1' B;
ENDREPEAT;
./DATA.TABLE(J).TIME = FILL CONCATENATE SUBSTR (C, K, 5-K);
END LEFT ZERO PADDING ON TIME_ENTRY;
PROCESS RIGHT_JUSTIFY_CEIL_DATA_ENTRY;
  [This process right-justifies ceiling entry]
  WX_DATA.TABLE(J).CEIL = F(CNVRT_WX.TABLE(J).CEIL, $FIVE);
END RIGHT_JUSTIFY_CEIL_DATA_ENTRY;

PROCESS RIGHT_JUSTIFY_VIS_DATA_ENTRY
  [This process right-justifies visibility entry]
  C = F(100.0 * CNVRT_WX.TABLE(J).VIS,$FOUR);
  WX_DATA.TABLE(J).VIS = SUBSTR(C,1,2)CONCATENATE " " CONCATENATE SUBSTR(C,3,2);
END RIGHT_JUSTIFY_CEIL_DATA_ENTRY;
PROCESS LEFT_ZERO_PADDING ON WIND_DIRECTION_ENTRY
[This process pads wind direction entry with leading zeroes]}

C = F(CHRT.WK.TABLE(j).DIR.$POUR);
K = 1;
FILL = '�';
FLAG = '0'B;

REPEAT WHILE (FLAG EQ '0'B) REPEAT UNTIL (K EQ 3);
  'X = X + 1;
  IF SUBSTR (C,X,1) = 'b'
    THEN FILL = FILL CONCATENATE '0'B;
  ELSE FLAG = 'l'B;
ENDREP;

IF FLAG EQ '0'B
  THEN X = 4;

WX_DATA.TABLE(j).DIR = FILL CONCATENATE SUBSTR(C,X,5-X);

END LEFT ZERO_PADDING ON WIND_DIRECTION_ENTRY;
PROCESS LEFT_JUSTIFY_REMARKS_DATA_ENTRY;

; [This process left-justifies remarks]

K = 0;
FLAG = 'O'B;

REPEAT WHILE (FLAG EQ 'O' B) REPEAT UNTIL (K EQ 35);

K = K + 1;

IF SUBSTR(MX_DATA_TABLE(J).REMARKS, K, 1) NE ' ' 

THEN LAG = '1'B;

ENDREPEAT;

C = SUBSTR (MX_DATA_TABLE(J).REMARKS, K, 36-K);

MX_DATA_TABLE(J).REMARKS = SUBSTR (C, 1, 35);

END LEFT_JUSTIFY_REMARKS_DATA_ENTRY;

PROCESS RIGHT_JUSTIFY_VEL_DATA_ENTRY

C = F(CONV_NK_TABLE(J).VEL, 6, FOURS)

MX_DATA_TABLE(J).VEL = SUBSTR(C,1,4) CONCATENATE 'B';

END RIGHT_JUSTIFY_VEL_DATA_ENTRY;
ROUTINE WUPDATA

INPUT (WX_DATA,CNVRT_WX);

(This routine is performed only when there are no errors committed on screen, it sort log entries on screen based on time)

LOOP: [J = 11 to 12]

L = J;

REPEAT WHILE (L GT 10);

IF (CNVRT_WX.TABLE(L+1).TIME LT CNVRT_WX.TABLE(L).TIME)

THEN

TEMP1 = WX_DATA.TABLE(L).TIME;
TEMP2 = WX_DATA.TABLE(L).CEIL;
TEMP3 = WX_DATA.TABLE(L).VIS;
TEMP4 = WX_DATA.TABLE(L).DIR;
TEMP5 = WX_DATA.TABLE(L).VEL;
TEMP6 = WX_DATA.TABLE(L).REMARKS;
CTEWP1 = CNVRT_WX.TABLE(L).TIME;
CTEWP2 = CNVRT_WX.TABLE(L).CEIL;
CTEWP3 = CNVRT_WX.TABLE(L).VIS;
CTEWP4 = CNVRT_WX.TABLE(L).DIR;
CTEWP5 = CNVRT_WX.TABLE(L).VEL;

WX_DATA.TABLE(L).TIME = WX_DATA.TABLE(L+1).TIME;
WX_DATA.TABLE(L).CEIL = WX_DATA.TABLE(L+1).CEIL;
WX_DATA.TABLE(L).VIS = WX_DATA.TABLE(L+1).VIS;
WX_DATA.TABLE(L).DIR = WX_DATA.TABLE(L+1).DIR;
WX_DATA.TABLE(L).VEL = WX_DATA.TABLE(L+1).VEL;
WX_DATA.TABLE(L).REMARKS = WX_DATA.TABLE(L+1).REMARKS;
CNVRT_WX.TABLE(L).TIME = CNVRT_WX.TABLE(L+1).TIME;
CNVRT_WX.TABLE(L).CEIL = CNVRT_WX.TABLE(L+1).CEIL;
CNVRT_WX.TABLE(L).VIS = CNVRT_WX.TABLE(L+1).VIS;
CNVRT_WX.TABLE(L).DIR = CNVRT_WX.TABLE(L+1).DIR;
CNVRT_WX.TABLE(L).VEL = CNVRT_WX.TABLE(L+1).VEL;
WX_DATA.TABLE(L+1).TIME = TEMP1;
WX_DATA.TABLE(L+1).CEIL = TEMP2;
WX_DATA.TABLE(L+1).VIS = TEMP3;
WX_DATA.TABLE(L+1).DIR = TEMP4;
WX_DATA.TABLE(L+1).VEL = TEMP5;
WX_DATA.TABLE(L+1).REMARKS = TEMP6;
CHVRTWX_TABLE(L+1).TIME = CTEDIT1;
CHVRTWX_TABLE(L+1).CEIL = CTEDIT2;
CHVRTWX_TABLE(L+1).VIS = CTEDIT3;
CHVRTWX_TABLE(L+1).DIR = CTEDIT4;
CHVRTWX_TABLE(L+1).VEL = CTEDIT5;
L = L - 1;
ELSE L = 10;
ENDREP:
ENDLOOP;
END UPDATE;
2.6 Airport Runway Surface Planning Log Screen

Pages 2-208 to 2-226 describe the Airport Runway Surface Planning Log Screen and its associated processing.
[**LOCAL VARIABLES**]

STRUCTURE SURF DATA LIKE SURFLOG
   [This structure is similar to SURFLOG used as a working area within screen routine]
ENDSTRUCTURE;

STRUCTURE CNVRTSRF LIKE CNVTSRF
   [This structure is similar to CNVTSRF used as a working area within screen routine]
ENDSTRUCTURE;

STRUCTURE SURF LOADLIST
   [A structure of pointers, one for each data field on screen used by panel manager for loading and unloading data to and from screen]

GROUP TABLE(13)
   PTR TIME [pointer for time data field]
   PTR RUNY [pointer for runway ID data field]
   PTR SURF [pointer for surface condition data field]
   PTR BRAK [pointer for braking condition data field]
   PTR CLOSED [pointer for runway closure data field]
   PTR OPEN [pointer for runway openings data field]
   PTR REMARKS [pointer for REMARKS data field]
   PTR MSG [pointer for screen message data field]

BITS FENCE [32 bit variable as prescribed by DNS manual, initialised to string of (32) '1'B]

ENDSTRUCTURE;
ROUTINE SLOG

INPUT (SURFLOG, CNVTSRF, RSTATUS)
[This routine invokes airport surface and runway planning log screen]

REPEAT UNTIL (RSTATUS NE FF12);
SURF DATA = SURFLOG;
CNVTSRF = CNVTSRF;
CALL SSSCREEN;

INPUT (SURF_DATA, CNVTSRF, RSTATUS)
[This routine controls airport surface and runway planning log screen]

ENDREPEAT;

IF SUBSTR (SURF_DATA.MSG, 1, 12) EQ 'DATA ENTERED'
THEN

SURFLOG = SURF_DATA;
CNVRTSRF = CNVTSRF;

END SLOG;

ROUTINE SSSCREEN

INPUT (SURF_DATA, CNVTSRF, RSTATUS);
[This routine controls airport surface and runway planning log screen]

CHAR PANEL [character variable of length 8 containing name of DNS panel initialized to 'APLOG2', name of panel that controls airport surface and runway planning log]

INT CURSOR [integer variable containing the cursor's position on the screen]

BITS DM(92) [8 bit variable of data mask used in DNS]

STRUCTURE AUX_DATA LIKE SURF_DATA

ENDSTRUCTURE;
DN = FLDEF; [set data masks to default intensity (normal)]
DN(92) = FLHIGH; [set message data mask to high intensity]
CURSOR = 71; [set cursor to position 71, first data field used by user]
AUX_DATA = SURF_DATA;
PERFORM SET_UP_SCREEN_POINTERS(SLOG);
REPEAT UNTIL (RSTATUS NE ENTER);
PERFORM DISPLAY_PANEL;
IF RSTATUS EQ PA1
THEN stop;
IF RSTATUS NE ENTER
THEN SURF_DATA = AUX_DATA;
ELSE
DM = FLDEF;
DM(92) = FLHIGH;
CALL SCHK;
INPUT (SURF_DATA, CNVRT_SRF, CURSOR);
[This routine checks for errors occurred on screen as a result of an erroneous entry and returns value for cursor pointing to first data field where an error has occurred; and an appropriate screen message is issued advising user with corrections]
IF SURF_DATA.MSG NE 'DATA ENTERED'
THEN DM(CURSOR) = FLHIGH;
ELSE
CALL SVALID;

INPUT (SURF_DATA, CHVST_SRF, CURSOR)

[This routine performs data validation checks on screen entries and returns value for cursor pointing to first invalid data field. Also, an appropriate screen message is issued advising user with corrections]

IF SURF_DATA.MSG NE 'DATA ENTERED'

THEN IN(CURSOR) = F ( neben;)

ELSE

CALL SUPDATA;

INPUT (SURF_DATA, CHVST_SRF)

[This routine is performed only when there are no errors committed on screen, it sorts log entries on screen based on time] SURF_DATA.MSG = 'DATA ENTERED AT ' CONCATENATE GMT;

AUX_DATA = SURF_DATA;

ENDREPEAT;

END SCREEN;
PROCESS SET_UP_SCREEN_POINTERS(SLOG)

[This process sets up screen pointers for DMS use]

LOOP: (J = 1 To 13)

SURF_LOADLIST.TABLE(J).TIME = ADDR (SURF_DATA.TABLE(J).TIME)
SURF_LOADLIST.TABLE(J).REY = ADDR (SURF_DATA.TABLE(J).REY);
SURF_LOADLIST.TABLE(J).SURF = ADDR (SURF_DATA.TABLE(J).SURF)
SURF_LOADLIST.TABLE(J).BRK = ADDR (SURF_DATA.TABLE(J).BRK)
SURF_LOADLIST.TABLE(J).CLOSED = ADDR (SURF_DATA.TABLE(J).CLOSED)
SURF_LOADLIST.TABLE(J).OPEN = ADDR (SURF_DATA.TABLE(J).OPEN)
SURF_LOADLIST.TABLE(J).REMARKS = ADDR (SURF_DATA.TABLE(J).REMARKS)

ENDLOOP;

SURF_LOADLIST.MSG = ADDR (SURF_DATA.MSG);

END SET_UP_SCREEN_POINTERS(SLOG);
ROUTINE SCHICK

INPUT (SURF_DATA, CVRT_SRF, CURSOR);

[This routine checks for errors occurred on screen as a result of an erroneous entry and returns
value for cursor pointing to first data field where an error has occurred, and an appropriate
screen message is issued advising user with corrections]

ERR1 = 'NUMERIC INPUT REQUIRED';
ERR2 = 'NON-NEGATIVE INPUT REQUIRED';
ERR3 = 'NO DECIMAL POINTS ALLOWED';
RUIDIR = 'VALID RUNWAY ID'S ARE: 4R 4L 9R 9L 14R 14L 22R 22L 27R 27L 32R 32L ALL';

RWYID(1) = '4R';
RWYID(2) = '4L';
RWYID(3) = '9R';
RWYID(4) = '9L';
RWYID(5) = '14R';
RWYID(6) = '14L';
RWYID(7) = '22R';
RWYID(8) = '22L';
RWYID(9) = '27R';
RWYID(10) = '27L';
RWYID(11) = '32R';
RWYID(12) = '32L';
RWYID(13) = 'ALL';
RWYID(14) = 'F';

SURF_DATA.MSG = 'DATA ENTERED';
CURSOR = 70;

ON CONVERSION BEGIN;

[ON CONVERSION is a PL/I feature. It is invoked if a character data is detected
in a numerical data field]

SURF_DATA.MSG = ERR1;
RETURN;

REPEAT WHILE (SURF_DATA.MSG EQ 'DATA ENTERED'); [J = 11 to 13]
CURSOR = CURSOR + 1;
PERFORM TIME_DATA_FIELD_ERROR_CHECK;
EXIT IF [error detected]
CURSOR = CURSOR + 1;
PERFORM RUNWAY_ID_DATA_FIELD_ERROR_CHECK;
EXIT IF [error detected]
CURSOR = CURSOR + 5;
ENDREPEAT;
END SCHECK;
PROCESS RUNWAY_ID_DATA_FIELD_ERROR_CHECK;
[This process checks for errors on runway ID data field]

K = INDEX (SURF_DATA.TABLE(J).RWY, 'b');

IF (K EQ 0) OR (K EQ 1)
  THEN C = SURF_DATA.TABLE(J).RWY;
ELSE IF K EQ 2
  THEN C = CONCATENATE SUBSTR (SURF_DATA.TABLE(J).RWY, 1, 1) CONCATENATE (SURF_DATA.
      TABLE(J).RWY, 3, 1);
ELSE C = CONCATENATE SUBSTR (SURF_DATA.TABLE(J).RWY, 1, 2);

FLAG = 'O'B;
REPEAT WHILE (FLAG EQ 'O'B); (K = 1 To 14)
  IF RUNID (K) EQ C
    THEN FLAG = '1'B;
ENDREPEAT;

IF FLAG = 'O'B;
  THEN SURF_DATA.MSG = RUNERR;
END RUNWAY_ID_DATA_FIELD_ERROR_CHECK;
PROCESS TIME_DATA_FIELD_ERROR_CHECK

[This process checks for errors on time data field]

GET STRING (SURF_DATA.TABLE(J).TIME) EDIT (CONVT_SRFTABLE(J).TIME)
[conversion from character data to numerical data]

IF VERIFY ('-', SURF_DATA.TABLE(J).TIME) EQ 0
THEN SURF_DATA.MSG = ERR2;
ELSEIF VERIFY ('.', SURF_DATA.TABLE(J).TIME) EQ 0
THEN SURF_DATA.MSG = ERR3;
END TIME_DATA_FIELD_ERROR_CHECK;
ROUTINE SVALID

INPUT (SURF_DATA, CVRT_SRF, CURSOR);
[This routine performs data validation checks on screen entries and returns value for cursor pointing to first invalid data field. Also, an appropriate screen message is issued advising user with corrections]

$POUR = 4;
CURSOR = 70;
SURF_DATA.NSG = 'DATA ENTERED';

REPEAT WHILE (SURF_DATA.NSG EQ 'DATA ENTERED'); [J = 11 To 13]

IF (SURF_DATA.TABLE(J).TIME NE (4) ' ')
AND (SURF_DATA.TABLE(J).RIF NE (3) ' ')
AND (SURF_DATA.TABLE(J).SURF EQ (5) ' ')
AND (SURF_DATA.TABLE(J).REAK EQ (5) ' ')
AND (SURF_DATA.TABLE(J).CLOSED EQ (6) ' ')
AND (SURF_DATA.TABLE(J).OPEN EQ (6) ' ')
AND (SURF_DATA.TABLE(J).RMAHA = (27) ' ')

THEN [All entries are blank]
CURSOR = CURSOR + 7
CVRT_SRF.TABLE(J).TIME = 9999;
ELSE
CURSOR = CURSOR + 1;

IF (SURF_DATA.TABLE(J).TIME HE (4) ' ')
AND (SURF_DATA.TABLE(J).RIF HE (3) ' ')
AND (SURF_DATA.TABLE(J).SURF EQ (5) ' ')
AND (SURF_DATA.TABLE(J).REAK EQ (5) ' ')
AND (SURF_DATA.TABLE(J).CLOSED EQ (6) ' ')
AND (SURF_DATA.TABLE(J).OPEN EQ (6) ' ')
AND (SURF_DATA.TABLE(J).REMARKS EQ (27) ' ')

THEN [Some entries are missing]
SURF_DATA.NSG = 'ADDITIONAL INFORMATION REQUIRED FOR THIS ENTRY';

END
ELSE
CURSOR = CURSOR + 1;
IF (SURF_DATA.TABLE(J).TIME EQ (4) ' ' ) AND
((SURF_DATA.TABLE(J).BUT NE (3) ' ' ) OR
(SURF_DATA.TABLE(J).SURF NE (5) ' ' ) OR
(SURF_DATA.TABLE(J).BREAK NE (5) ' ' ) OR
(SURF_DATA.TABLE(J).CLOSED NE (6) ' ' ) OR
(SURF_DATA.TABLE(J).OPEN NE (6) ' ' ) OR
(SURF_DATA.TABLE(J).REMARKS NE (27) ' ' ))
THEN [Time is missing]
SURF_DATA.MSG = 'SPECIFY TIME ASSOCIATED WITH ENTRIES';
ELSE
CURSOR = CURSOR + 1;
IF (SURF_DATA.TABLE(J).BUT EQ (3) ' ' ) AND
((SURF_DATA.TABLE(J).TIME NE (4) ' ' ) OR
(SURF_DATA.TABLE(J).SURF NE (5) ' ' ) OR
(SURF_DATA.TABLE(J).BREAK NE (5) ' ' ) OR
(SURF_DATA.TABLE(J).CLOSED NE (6) ' ' ) OR
(SURF_DATA.TABLE(J).OPEN NE (6) ' ' ) OR
(SURF_DATA.TABLE(J).REMARKS NE (27) ' ' ))
THEN [Runway ID is missing]
SURF_DATA.MSG = 'SPECIFY RUNWAY ASSOCIATED WITH ENTRIES';
ELSE [check time]
CURSOR = CURSOR - 1;
PERFORM TIME_CHECK;
ENDIF [time entry is erroneous]
PERFORM LEFT_ZERO_PADDING_ON_TIME_ENTRY;
IF SURF_DATA.MSG EQ 'DATA ENTERED'
ENDRPET;

IF SURF_DATA.MSG EQ 'DATA ENTERED'
  THEN CURSOR = 71;
END SVALID;
PROCESS TIME_CHECK

(This process checks validity of time entry)

HOUR = FLOOR (FLOAT(CNVRT_SRF.TABLE(J).TIME)/100.0);

IF HOUR GT 23

THEN SURF_DATA.MSG = 'HOUR MUST NOT EXCEED 23';

ELSE MIN = CVNRT_SRF.TABLR(J).TIME - HOUR *100;

IF MIN GT 59

THEN SURF_DATA.MSG = 'MINUTES MUST NOT EXCEED 59';

END TIME_CHECK;
PROCESS LEFT_ZERO_PADDING_ON_TIME_ENTRY;
  [This process pads time entry with leading zeroes]
  C = F(STRING(CHRT_SEP.TABLE(J).TIME), 8);  
  K = 0;
  FILL = ''; 
  FLAG = '0'F;
  REPEAT WHILE (FLAG EQ '0'F) REPEAT UNTIL (K EQ 4);
  K = K + 1;
  IF SUBSTR (C, K, 1) EQ ''
    THEN FILL = FILL CONCATENATE '0';
    ELSE FLAG = '1';
  ENDREPEAT;
  SURF_DATA_TABLE(J).TIME = FILL CONCATENATE SUBSTR (C, K, 5-K);
END LEFT_ZERO_PADDING_ON_TIME_ENTRY;
PROCESS RIGHT_JUSTIFY_RWY_DATA_ENTRY
[This process right-justifies runway ID entry]
    K = INDEX(SURF_DATA_TABLE(J).RWY, '');
    IF K EQ 2 THEN
        C = CONCATENATE SUBSTR (SURF_DATA_TABLE(J).RWY, 1, 1) CONCATENATE SUBSTR (SURF_DATA_TABLE(J).RWY, 3, 1); SURF_DATA_TABLE(J).RWY = SUBSTR (C, 1, 3);
    IF K EQ 3 THEN
        C = CONCATENATE SUBSTR(SURF_DATA_TABLE(J).RWY, 1, 2); SURF_DATA_TABLE(J).RWY = SUBSTR (C, 1, 3);
    END RIGHT_JUSTIFY_RWY_DATA_ENTRY;

PROCESS LEFT_JUSTIFY_SURF_DATA_ENTRY
[This process left-justifies surface conditions entry]
    K = 0;
    FLAG = '0'8;
    REPEAT WHILE (FLAG EQ '0'8) REPEAT UNTIL (K EQ 5);
        K = K + 1;
        IF SUBSTR(SURF_DATA_TABLE(J).SURF, K, 1) NE '' THEN FLAG = '1'8;
    ENDBREAK;
    C = SUBSTR(SURF_DATA_TABLE(J).SURF, K, 6-K); SURF_DATA_TABLE(J).SURF = SUBSTR(C, 1, 5);
    END LEFT_JUSTIFY_SURF_DATA_ENTRY;
PROCESS LEFT_JUSTIFY_BRAK_DATA_ENTRY
[This process left-justifies braking condition entry]

K = 0;
FLAG = '0';

REPEAT WHILE (FLAG EQ '0') REPEAT UNTIL (K EQ 5);

K = K + 1;

IF SUBSTR(SURF_DATA_TABLE(J).BRAK, K, 1) NE '

THEN FLAG = '1';

ENDREPEAT;
C = SUBSTR(SURF_DATA_TABLE(J).BRAK, K, 6-K); SURF_DATA_TABLE(J).BRAK = SUBSTR(C,1,5)

END LEFT_JUSTIFY_BRAK_DATA_ENTRY;

PROCESS LEFT_JUSTIFY_CLOSED_DATA_ENTRY
[This process left-justifiesummy closure entry]

K = 0;
FLAG = '0';

REPEAT WHILE (FLAG EQ '0') REPEAT UNTIL (K EQ 6);

K = K + 1;

IF SUBSTR(SURF_DATA_TABLE(J).CLOSED, K, 1) NE '

THEN FLAG = '1';

ENDREPEAT;
C = SUBSTR(SURF_DATA_TABLE(J).CLOSED, K, 7-K); SURF_DATA_TABLE(J).CLOSED = SUBSTR(C,1,6);

END LEFT_JUSTIFY_CLOSED_DATA_ENTRY;
PROCESS LEFTJUSTIFY-OPEN_DATA_ENTRY
[This process left-justifies runway opening entry]

\[
k = 0; \\
\text{FLAG} = '0'; \\
\text{REPEAT WHILE (FLAG EQ '0') REPEAT UNTIL (K EQ 6)}; \\
\quad K = K + 1; \\
\quad \text{IF SUBSTR(SURF_DATA.TABLE(J).OPEN,K,1) NE ' '}; \\
\quad \text{THEN} \quad \text{FLAG} = '1'; \\
\text{ENDREPEAT} \\
C = \text{SUBSTR(SURF_DATA.TABLE(J).OPEN,K,7-K)} \quad \text{SURF_DATA.TABLE(J).OPEN} = \text{SUBSTR(C,1,6)}; \\
\text{END LEFTJUSTIFY-OPEN_DATA_ENTRY};
\]

PROCESS LEFTJUSTIFY_REMARKS_DATA_ENTRY
[This process left-justifies remarks entry]

\[
k = 0; \\
\text{FLAG} = '0'; \\
\text{REPEAT WHILE (FLAG EQ '0') REPEAT UNTIL (K EQ 27)}; \\
\quad K = K + 1; \\
\quad \text{IF SUBSTR(SURF_DATA.TABLE(J).REMARKS,K,1) NE ' '}; \\
\quad \text{THEN} \quad \text{FLAG} = '1'; \\
\text{ENDREPEAT} \\
C = \text{SUBSTR(SURF_DATA.TABLE(J).REMARKS,K,28-K)} \quad \text{SURF_DATA.TABLE(J).REMARKS} = \text{SUBSTR(C,1,27)}; \\
\text{END LEFTJUSTIFY_REMARKS_DATA-ENTRY};
\]
ROUTINE UPDATE

IMUT ((SURF_DATA, CNVRT_SRF));

[This routine is performed only when there are no errors committed on screen, it sorts log entries on screen based on time]

LOOP; [J = 11 to 12]
L = J;

REPEAT WHILE (L GT 10);

IF CNVRT_SRF.TABLE(L).TIME LT CNVRT_SRF.TABLE(L).TIME
THEN

TEMP1 = SURF_DATA.TABLE(L).TIME;
TEMP2 = SURF_DATA.TABLE(L).RNY;
TEMP3 = SURF_DATA.TABLE(L).SURF;
TEMP4 = SURF_DATA.TABLE(L).BAK;
TEMP5 = SURF_DATA.TABLE(L).CLOSED;
TEMP6 = SURF_DATA.TABLE(L).OPEN;
TEMP7 = SURF_DATA.TABLE(L).REMARKS;
CTEMP1 = CNVRT_SRF.TABLE(L).TIME;
SURF_DATA.TABLE(L).TIME = SURF_DATA.TABLE(L+1).TIME;
SURF_DATA.TABLE(L).RNY = SURF_DATA.TABLE(L+1).RNY;
SURF_DATA.TABLE(L).SURF = SURF_DATA.TABLE(L+1).SURF;
SURF_DATA.TABLE(L).BAK = SURF_DATA.TABLE(L+1).BAK;
SURF_DATA.TABLE(L).CLOSED = SURF_DATA.TABLE(L+1).CLOSED;
SURF_DATA.TABLE(L).OPEN = SURF_DATA.TABLE(L+1).OPEN;
SURF_DATA.TABLE(L).REMARKS = SURF_DATA.TABLE(L+1).REMARKS;
CNVRT_SRF.TABLE(L).TIME = CNVRT_SRF.TABLE(L+1).TIME
SURF_DATA.TABLE(L+1).TIME = TEMP1;
SURF_DATA.TABLE(L+1).RNY = TEMP2;
SURF_DATA.TABLE(L+1).SURF = TEMP3;
SURF_DATA.TABLE(L+1).BAK = TEMP4;
SURF_DATA.TABLE(L+1).CLOSED = TEMP5;

SURF_DATA.TABLE(L+1).OPEN = TEMP6;
SURF_DATA.TABLE(L+1).REMARKS = TEMP7;
CHVT_SRF.TABLE(L+1).TIME = CTEMP1;
L = L - 1;
ELSE
   L = 10;
ENDREPEAT;
ENDLOOP;
END UPDATE;
2.7 Equipment Planning Log Screen

The Equipment Planning Log Screen is described on pages 2-228 to 2-244.
***LOCAL VARIABLES***

**STRUCTURE EQUIP DATA LIKE EQPLOG**

[This structure is similar to EQPLOG used as a working area within screen routine]

**ENDSTRUCTURE**;

**STRUCTURE CMVT_EQP LIKE CMVT_EQP**

[This structure is similar to CMVT_EQP used as a working area within screen routine]

**ENDSTRUCTURE**;

**STRUCTURE EQUIP_LOADLIST**

[A structure of pointers, one for each data field on screen used by panel manager for loading and unloading data to and from screen]

**GROUP TABLE(15)**

PTR RNY [pointer for runway ID data field]

PTR EQUIPMENT [pointer for equipment data field]

PTR OTS [pointer for 'OUT OF SERVICE' time data field]

PTR RTS [pointer for 'RETURN TO SERVICE' time data field]

PTR REMARKS [pointer for REMARKS data field]

PTR MSG [pointer for screen message data field]

PTR FENCE [32 bit variable as prescribed by DNS manual, initialized to string of (32) '1'B]

**ENDSTRUCTURE**;
ROUTINE ELOG

INOUT (EQPLOG, CHVTEQP, ESTATUS);
   [This routine invokes equipment planning log screen]
REPEAT UNTIL (ESTATUS NE FF12);
   EQUIP DATA = EQPLOG;
   CHVTEQP = CHVTEQP;
   CALL ESCREEN;
      INOUT (EQUIP_DATA, CHVTEQP, ESTATUS);
      [This routine controls equipment planning log screen]
ENDREPEAT;

IF SUBSTR (EQUIP_DATA, MSG, 1, 12) EQ 'DATA ENTERED'
   THEN
      EQPLOG = EQUIP_DATA;
      CHVTEQP = CHVTEQP;
END ELOG;
ROUTINE ESCREEN

INPUT (EQUIP_DATA, CNVRTEQP, RSTATUS);

[This routine controls equipment planning log screen]

CHR NAME [character variable of length 8 containing name of DNS panel initialized to 'BSYLOG',
name of panel that controls equipment planning log screen]

INT CURSOR [integer variable containing cursor's position on screen]

BITS DM(76) [8 bit variable of data mask used in DNS]

STRUCTURE AUX_DATA LIKE EQUIP_DATA

ENDSTRUCTURE;

DM = FLDDDF; [set data masks to default intensity (normal)]
DM(76) = FLHIGH; [set message data mask to high intensity]
CURSOR = 61; [set cursor to position 61; first data field used by user]
AUX_DATA = EQUIP_DATA;
PERFORM SET_UP_SCREEN_POINTERS (ELOG);
REPEAT UNTIL (RSTATUS NE ENTER);

PERFORM DISPLAY_PANEL;

IF RSTATUS EQ PAI
THEN STOP;

IF RSTATUS NE ENTER
THEN EQUIP_DATA = AUX_DATA;
ELSE
DM = FLDDDF;
DM(76) = FLHIGH;
CALL ECHECK;
INOUT EQUIP_DATA, CNVRT_EQP, CURSOR);

[This routine checks for errors occurred on screen as a result of an erroneous entry and returns value for cursor pointing to first data field when an error has occurred, and an appropriate screen message is issued advising user with corrections]

IF EQUIP_DATA.MSG NE 'DATA ENTERED'
THEN DM(CURSOR) = FLHMIGH;
ELSE
CALL EVALD;

INOUT (EQUIP_DATA, CNVRT_EQP, CURSOR);

[This routine performs data validation checks on screen entries and returns value for cursor pointing to first invalid data field. Also, an appropriate screen message is issued advising user with corrections]

IF EQUIP_DATA.MSG NE 'DATA ENTERED'
THEN DM(CURSOR) = FLHMIGH;
ELSE
CALL UPDATE;

INOUT (EQUIP_DATA, CNVRT_EQP);

[This routine is performed only when there are no errors committed on screen, it sorts log entries on screen based on primarily OTS and then on BTS times]

EQUIP_DATA.MSG = 'DATA ENTERED at 'CONCATENATE GMT;
AUX_DATA = EQUIP_DATA
ENDREPEAT
END SCREEn;
PROCESS SET_UP SCREEN_POINTERS (ELOG)
[This process sets up screen pointers for DNS use]

LOOP:  [J = 1 To 15]

EQUIP_LOADLIST.TABLE(J).RWY = ADDR (EQUIP_DATA.TABLE(J).RWY);
EQUIP_LOADLIST.TABLE(J).EQUIPMENT = ADDR (EQUIP_DATA.TABLE(J).EQUIPMENT);
EQUIP_LOADLIST.TABLE(J).OTS = ADDR (EQUIP_DATA.TABLE(J).OTS);
EQUIP_LOADLIST.TABLE(J).RTS = ADDR (EQUIP_DATA.TABLE(J).RTS);
EQUIP_LOADLIST.TABLE(J).REMARKS = ADDR (EQUIP_DATA.TABLE(J).REMARKS);

ENDLOOP;

EQUIP_LOADLIST.MSG = ADDR (EQUIP_DATA.MSG);

END SET_UP_SCREEN_POINTERS (ELOG);
ROUTINE ECHECK

INPUT (EQUIP_DATA, CNVRT_EQP, CURSOR)

[This routine checks for errors occurred on screen as a result of an erroneous entry and returns
value for cursor pointing to first data field when an error has occurred, and an appropriate screen
message is issued advising user with corrections]

ERR1 = 'NUMERIC INPUT REQUIRED';
ERR2 = 'NON-NEGATIVE INPUT REQUIRED';
ERR3 = 'NO DECIMAL POINTS ALLOWED';
RWYFR = 'VALID RUNWAY ID'S ARE: 4R 4L 9R 9L 14R 14L 22R 22L 27R 27L 32R 32L ALL';
RWYID(1) = '4R';
RWYID(2) = '4L';
RWYID(3) = '9R';
RWYID(4) = '9L';
RWYID(5) = '14R';
RWYID(6) = '14L';
RWYID(7) = '22R';
RWYID(8) = '22L';
RWYID(9) = '27R';
RWYID(10) = '27L';
RWYID(11) = '32R';
RWYID(12) = '32L';
RWYID(13) = 'ALL';
RWYID(14) = '

EQUIP_DATA.MSG = 'DATA ENTERED';
CURSOR = 60;

REPEAT WHILE (EQUIP_DATA.MSG EQ 'DATA ENTERED'); [J = 13 to 15]

CURSOR = CURSOR + 1;

PERFORM RUNWAY_ID_DATA_FIELD_ERROR_CHECK;

EXITIF [error detected]

CURSOR = CURSOR + 2;
PERFORM OTS_TIME_DATA_FIELD_ERROR_CHECK;
EXITIF [error detected]
CURSOR = CURSOR + 1;
PERFORM RTS_TIME_DATA_FIELD_ERROR_CHECK;
EXITIF [error detected]
ENDREPEAT;
END ECHECK;
PROCESS RUNWAY_ID_DATA_FIELD_ERROR_CHECK;

[This process checks for errors on runway ID data field]

\[ K = \text{INDEX (EQUIP\_TABLE(J).WBY, ' ')}; \]

\textbf{IF} (K EQ 0) OR (K EQ 1) \textbf{THEN} \( C = \text{EQUIP\_TABLE(J).WBY}; \)

\textbf{ELSEIF} K EQ 2

\textbf{THEN} \( C = \text{} \text{CONCATENATE} \text{} \text{SUBSTR (EQUIP\_TABLE(J).WBY, 1, 1)} \text{} \text{CONCATENATE} \text{} \text{EQUIP\_TABLE(J).WBY, 3, 1)}; \)

\textbf{ELSE} \( C = \text{} \text{CONCATENATE} \text{} \text{SUBSTR (EQUIP\_TABLE(J).WBY, 1, 2)}; \)

\textbf{FLAG} = '0'B;

\textbf{REPEAT} WHILE (\textbf{FLAG} EQ '0'B); \[ K \text{ EQ 1 To 14} \]

\textbf{IF} \text{EYWID(K)} EQ C \textbf{THEN} \text{FLAG} = '1'B

\textbf{ENDREPEAT};

\textbf{IF} \text{FLAG} = '0'B;

\textbf{THEN} \text{EQUIP\_MSG} = \text{EYWERS};

\textbf{END RUNWAY_ID_DATA_FIELD_ERROR_CHECK;}
PROCESS OTS_TIME_DATA_FIELD_ERROR_CHECK
[This process checks for errors on OTS time data field]
Get STRING (EQUIP_DATA.TABLE(J).OTS) EDIT (CINV_EQP.TABLE(J).OTS)
[conversion from character data to numerical data]
IF VERIFY ('-', EQUIP_DATA.TABLE(J).OTS) EQ 0
THEN EQUIP_DATA.MSG = ERR2;
ELSEIF VERIFY ('.', EQUIP_DATA.TABLE(J).OTS) EQ 0
THEN EQUIP_DATA.MSG = ERR3;
END OTS_TIME_DATA_FIELD_ERROR_CHECK;

PROCESS RTS_TIME_DATA_FIELD_ERROR_CHECK
[This process checks for errors on RTS time data field]
Get STRING (EQUIP_DATA.TABLE(J).RTS) EDIT (CINV_EQP.TABLE(J).RTS)
[conversion from character data to numerical data]
IF VERIFY ('-', EQUIP_DATA.TABLE(J).RTS) EQ 0
THEN EQUIP_DATA.MSG = ERR2;
ELSEIF VERIFY ('.', EQUIP_DATA.TABLE(J).RTS) EQ 0
THEN EQUIP_DATA.MSG = ERR3;
END RTS_TIME_DATA_FIELD_ERROR_CHECK;
ROUTINE EVALID

INPUT (EQUIP_DATA, CNVRT_EQP, CURSOR);

(This routine performs data validation checks on screen entries and returns value for cursor pointing to first invalid data field. Also, an appropriate screen message is issued advising user with corrections)

$POUR = 4;
CURSOR = 60;
EQUIP_DATA.MSG = 'DATA ENTERED';

REPEAT WHILE (EQUIP_DATA.MSG EQ 'DATA ENTERED' [J = 13 To 15])

IF (EQUIP_DATA.TABLE(J).RNY EQ (3) ' ') AND
(EQUIP_DATA.TABLE(J).EQUIPMENT EQ (11) ' ') AND
(EQUIP_DATA.TABLE(J).OTS EQ (4) ' ') AND
(EQUIP_DATA.TABLE(J).ReMKs EQ (39) ' ')
THEN [all entries are blank]
CURSOR = CURSOR + 5;
CNVRT_EQP.TABLE(J).OTS = 9999;
CNVRT_EQP.TABLE(J).RTS = 9999;
ELSE
CURSOR = CURSOR + 1;
PERFORM RIGHT_JUSTIFY_RNT_DATA_ENTRY;

IF (EQUIP_DATA.TABLE(J).RNY NE (3) ' ') AND
(EQUIP_DATA.TABLE(J).EQUIPMENT EQ (11) ' ') AND
(EQUIP_DATA.TABLE(J).OTS EQ (4) ' ') AND
(EQUIP_DATA.TABLE(J).RTS EQ (4) ' ') AND
(EQUIP_DATA.TABLE(J).REMARKS EQ (39) ' ')
THEN [entries are missing]
EQUIP_DATA.MSG = 'ADDITIONAL INFORMATION REQUIRED FOR THIS RUNWAY';
ELSEIF
(EQUIP_DATA.TABLE(J).RNY EQ (3) ' ') AND
((EQUIP_DATA.TABLE(J).EQUIPMENT NE (11) ' ') OR...
(EQUIP_DATA.TABLE(J).OTS NE (4) ' ') OR
(EQUIP_DATA.TABLE(J).RTS NE (4) ' ') OR
(EQUIP_DATA.TABLE (J).Remarks NE (39) ' ');

THEN [runway ID is missing]

EQUIP_DATA.MSG = 'SPECIFY RUNWAY ASSOCIATED WITH ENTRIES';

ELSEIF

(EQUIP_DATA.TABLE(J).RUN ME (3) ' ') AND
(EQUIP_DATA.TABLE(J).EQUIPMENT EQ (11))

THEN [equipment type is missing]

EQUIP_DATA.MSG = 'SPECIFY EQUIPMENT TYPE ASSOCIATED WITH ENTRIES';

ELSEIF

(EQUIP_DATA.TABLE(J).RUN ME (3) ' ') AND
(EQUIP_DATA.TABLE(J).EQUIPMENT EQ (11)) AND
(EQUIP_DATA.TABLE(J).RTS EQ (4) ' ')

THEN [OTS and RTS times are missing]

EQUIP_DATA.MSG = 'AN OTS AND/OR RTS TIME IS REQUIRED'

ELSE

CURSOR = CURSOR + 1;

PERFORM RIGHT_JUSTIFY_EQUIPMENT_DATA_ENTRY;

CURSOR = CURSOR + 1;

IF EQUIP_DATA.TABLE(J).OTS EQ (4) 'b'

THEN CWRT_EQP_TABLE(J).OTS = 2500;

ELSE

PERFORM OTS_TIME_CHECK;

EXITIF [error detected]

PERFORM LEFT_ZERO_PADDING_ON_TIME_ENTRY;
IF EQUIP_DATA.MSG EQ 'DATA ENTERED';
    THEN
        CURSOR = CURSOR + 1;
        IF EQUIP_DATA.TABLE(J).Rts Eq (4)''
            THEN CNVT.EQP.TABL(J).Rts = 2500;
        ELSE
            PERFORM RTS.TIME_CHECK
            EXIT IF [error detected]
        IF EQUIP_DATA.MSG EQ 'DATA ENTERED'
            THEN
                [Equation]
                THEN EQUIP_DATA.MSG = 'TIME FOR BTS MUST BE BLANK OR LATER THAN OTS';
            ELSE
                CURSOR = CURSOR + 1;
                PERFORM LEFT
                JUSTIFY REMARKS
                DATA_ENTRY;
            END REPEAT;
        IF EQUIP_DATA.MSG EQ 'DATA ENTERED'
            THEN
                CURSOR = 61;
        END EVALID;
PROCESS RIGHT JUSTIFY RUNWAY DATA ENTRY

This process right-justifies runway ID entry

K = INDEX (EQUIP_DATA.TABLE(J).RWY, ";");

IF K EQ 2 THEN

C = "" CONCATENATE SUBSTR (EQUIP_DATA.TABLE(J).RWY, 1, 1) CONCATENATE SUBSTR (EQUIP_DATA.

TABLE(J).RWY, 3, 1); EQUIP_DATA.TABLE(J).RWY = SUBSTR (C, 1, 3);

IF K EQ 3 THEN

C = "" CONCATENATE SUBSTR (EQUIP_DATA.TABLE(J).RWY, 1, 2), EQUIP_DATA.TABLE(J).RWY = SUBSTR

(C, 1, 3);

END RIGHT JUSTIFY RUNWAY_DATA_ENTRY;

PROCESS RIGHT JUSTIFY EQUIPMENT_DATA_ENTRY

This process right-justifies equipment type entry

K = 0;

FLAG = 'O'B;

REPEAT WHILE (FLAG EQ 'O'B) REPEAT UNTIL (K EQ 11);

K = K + 1;

IF SUBSTR (EQUIP_DATA.TABLE(J).EQUIPMENT, K, 1) NE " " THEN FLAG = 'I'B;

ENDREPEAT;

C = SUBSTR (EQUIP_DATA.TABLE(J).EQUIPMENT, K, 12 - K); EQUIP_DATA.TABLE(J).EQUIPMENT = SUBSTR (C, 1, 11);

END RIGHT JUSTIFY EQUIPMENT_DATA_ENTRY;
PROCESS OTS_TIME_CHECK
[This process checks for validity of OTS time entry]

HOUR = FLOOR (FLOAT(CNV_EQP.TABLE(J).OTS)/100.0);

IF HOUR GT 23
    THEN EQUIP_DATA.MSG = 'HOUR MUST NOT EXCEED 23';
ELSE
    MIN = CVRT_EQP.TABLE(J).OS - HOUR * 100;
    IF MIN GT 59
        THEN EQUIP_DATA.MSG = 'MINUTES MUST NOT EXCEED 59';
END;

END OTS_TIME_CHECK;

PROCESS LEFT_ZERO_PADDING_ON_OTS_TIME_ENTRY;
[This process pads OTS time entry with leading zeroes]

C = F(FLOAT(CNV_EQP.TABLE(J).OTS), &SOUR);
K = 0;
FILL = -;
FLAG = '0';

REPEAT WHILE (FLAG EQ '0') REPEAT UNTIL (K EQ 4);
    K = K + 1;
    IF SUBSTR (C, K, 1) EQ '0'
        THEN FILL = FILL CONCATENATE '0';
    ELSE FLAG = '1';
ENDREPEAT;

EQUIP_DATA.TABLE(J).OTS = FILL CONCATENATE SUBSTR (C, 5-K);

END LEFT_ZERO_PADDING_ON_OTS_TIME_ENTRY;
PROCESS RTS_TIME_CHECK
[This process checks validity of RTS time entry]

HOUR = FLOOR(FLOAT(CNVRT_EQP.TABLE(J).RTS)/100.0);

IF HOUR GT 23 THEN EQUIP_DATA.MSG = 'HOUR MUST NOT EXCEED 23';
ELSE
    MIN = CNVRT_EQP.TABLE(J).RTS - HOUR * 100;
    IF MIN GT 59 THEN EQUIP_DATA.MSG = 'MINUTES MUST NOT EXCEED 59';
END RTS_TIME_CHECK;

PROCESS LEFT_ZERO_PADDING_ON_RTS_TIME_ENTRY
[This process pads RTS time entry with leading zeroes]

C = $P(FLOAT(CNVRT_EQP.TABLE(J).RTS), $FOUR);
K = 0;
FILL = ';
FLAG = '0';

REPEAT WHILE (FLAG EQ '0') REPEAT UNTIL (K EQ 4);

K = K + 1;
IF SUBSTR (C, K, 1) EQ '' THEN FILL = FILL CONCATENATE '0'; ELSE FLAG = '1';
ENDREPEAT;

EQUIP_DATA.TABLE(J).RTS = FILL CONCATENATE SUBSTR (C, K, 5-K);

END LEFT_ZERO_PADDING_ON_RTS_TIME_ENTRY;
process left_justify_remarks_data_entry
  [This process left_justifies remarks entry]
  k = 0;
  flag = '0';
  repeat while (flag eq '0') repeat until (k eq 27);
  k = k + 1;
  if substr(surf_data_table(j).remarks, k, 1) ne '
     then flag = '1';
     endif;
  endrepeat;
  c = substr(surf_data_table(j).remarks, k, 28 - k); surf_data_table(j).remarks = substr(c, 1, 27);
end left_justify_remarks_data_entry;
ROUTINE UPDATE

INOUT (EQUIP_DATA, CONV_EQP)

[This routine is performed only when there are no errors committed on screen, it sorts log entries on screen based on time]

LOOP: [J = 13 To 14]
L = J;

REPEAT WHILE (L GT 13);
IF (CONV_EQP.TABL(L+1).OTS EQ CONV_EQP.TABL(L).OTS) OR (CONV_EQP.TABL(L+1).OTS EQ CONV_EQP.TABL(L).OTS) AND (CONV_EQP.TABL(L+1).ETS LT CONV_EQP.TABL(L).ETS)
THEN
TEMP1 = EQUIP_DATA.TABL(L).DT;
TEMP2 = EQUIP_DATA.TABL(L).EQUIPMENT;
TEMP3 = EQUIP_DATA.TABL(L).OTS;
TEMP4 = EQUIP_DATA.TABL(L).ETS;
CTEMP1 = CONV_EQP.TABL(L).OTS;
CTEMP2 = CONV_EQP.TABL(L).ETS;
EQUIP_DATA.TABL(L).DT = EQUIP_DATA.TABL(L+1).DT;
EQUIP_DATA.TABL(L).EQUIPMENT = EQUIP_DATA.TABL(L+1).EQUIPMENT;
EQUIP_DATA.TABL(L).OTS = EQUIP_DATA.TABL(L+1).OTS;
EQUIP_DATA.TABL(L).ETS = EQUIP_DATA.TABL(L+1).ETS;
EQUIP_DATA.TABL(L).REMREMS = EQUIP_DATA.TABL(L+1).REMREMS;
CONV_EQP.TABL(L).OTS = CONV_EQP.TABL(L+1).OTS;
CONV_EQP.TABL(L).ETS = CONV_EQP.TABL(L+1).ETS;
EQUIP_DATA.TABL(L+1).DT = TEMP1;
EQUIP_DATA.TABL(L+1).EQUIPMENT = TEMP2;
EQUIP_DATA.TABL(L+1).OTS = TEMP3;
EQUIP_DATA.TABL(L+1).ETS = TEMP4;
EQUIP_DATA.TABL(L+1).REMREMS = TEMP5;
CONV_EQP.TABL(L+1).OTS = CTEMP1;
CONV_EQP.TABL(L+1).ETS = CTEMP2;
L = L - 1;
ELSE L = 12;
ENDREPEAT;
END LOOP;
END UPDATE;
2.8 Demand Planning Log Screen

The processing for the Demand Planning Log Screen is described on pages 2-246 to 2-263.
STRUCTURE OAG DATA LIKE OAGLOG
[This structure is similar to OAGLOG used as a working area within screen routine]
ENDSTRUCTURE;

STRUCTURE CVRT OAG LIKE CVTOAG
[This structure is similar to CVTOAG used as a working area within screen routine]
ENDSTRUCTURE;

STRUCTURE OAG_LOADLIST [a structure of pointers, one for each data field on screen used by panel manager for loading and unloading data to and from screen]
  PTR INITIAL [pointer for initial data field]
  PTR SCROLL [pointer for scroll data field]
  GROUP TABLE(4)
    PTR GMT [pointer for GMT data field]
    PTR TTLLRD [pointer for total arrival demand data field]
    PTR TTDLPF [pointer for total departure demand data field]
    PTR KUBBS [pointer for KUBBS arrival demand data field]
    PTR CGT [pointer for CGT arrival demand data field]
    PTR VAINS [pointer for VAINS arrival demand data field]
    PTR FARM [pointer for FARM arrival demand data field]
PTR NORTH [pointer for NORTH departure demand data field]
PTR EAST [pointer for EAST departure demand data field]
PTR SOUTH [pointer for SOUTH departure demand data field]
PTR WEST [pointer for WEST departure demand data field]
PTR MSG [pointer for the screen message data field]
BITE FENCE [32 bit variable as prescribed for DMS manual, initialized to string of (32) '1'8]
ENDSTRUCTURE;
ROUTINE CLOG
IN (OAGDEM);
INOUT (OAGLOG, CVTOAG, RSTATUS);
[This routine invokes demand planning log screen]
REPEAT UNTIL (RSTATUS NE PF12);
OAG_DATA = OAGLOG;
CVTVT_OAG = CVTOAG;
CALL CSSCREEN;
IN (OAGDEM);
INOUT (OAG_DATA, CVTVT_OAG, RSTATUS);
[This routine controls demand planning log screen]
ENDREPEAT;
IF SUBSTR (OAG_DATA.MSG, 1, 12) EQ 'DATA ENTERED'
THEN
OAGLOG = OAG_DATA;
CVTVT_OAG = CVTVT_OAG;
END CLOG;
ROUTINE GSCREEN

IN (OAGDATA);
INPUT (OAGDATA, CNVRT_OAG, ESTATUS);  
(This routine controls demand planning log screen)

CHR PANEL [character variable of length 8 containing the name of DNS panel initialized to 'DMNDLOG', the name of the panel that controls demand planning log screen]

INT CURSOR [integer variable containing the cursor's position on the screen]

BITS DM(47): [8 bit variable of data mask used in DNS]

STRUCTURE AUX_DATA LIKE OAG_DATA

ENDSTRUCTURE;

DM = FLDOFF; [set data masks to default intensity (normal)]

DM(47) = FLDOFF; [set the message data mask to high intensity]

CURSOR = 2; [set cursor to position 2; second data field used by user]

AUX_DATA = OAG_DATA;
OAG_DATA.INITIAL = '2';
OAG_DATA.SCROLL = '2';
CNVRT_OAG.SCROLL = 0;

OAG_LOADLIST.INITIAL = ADDR(OAG_DATA.INITIAL); [set up pointer for initial data field]
OAG_LOADLIST.SCROLL = ADDR(OAG_DATA.SCROLL); [set up pointer for scroll data field]
OAG_LOADLIST.MSG = ADDR(OAG_DATA.MSG); [set up pointer for message data field]

Get STRING (GMT) EDIT (G) [convert the current time from character to numeric]

REPEAT UNTIL (ESTATUS NE ENTER);

INDEX = FLOOR (MOD(FLOAT(CNVRT_OAG.SCROLL + INDEX), 24.0));
[compute current hour]

LOOP: [L = 1 To 4]
[set up screen pointers for four hours starting with current hour]
HR = FLOOR(MOD(FLOAT(INDEX + L - 1), 24.0))

OAG_LOADLIST_TABLE(L).CNT = ADDR(OAG_DATA_TABLE(HR).CNT);
OAG_LOADLIST_TABLE(L).TAILER = ADDR(OAG_DATA_TABLE(HR).TAILER);
OAG_LOADLIST_TABLE(L).TYPDEF = ADDR(OAG_DATA_TABLE(HR).TYPDEF);
OAG_LOADLIST_TABLE(L).CUSBS = ADDR(OAG_DATA_TABLE(HR).CUSBS);
OAG_LOADLIST_TABLE(L).CUT = ADDR(OAG_DATA_TABLE(HR).CUT);
OAG_LOADLIST_TABLE(L).VAINS = ADDR(OAG_DATA_TABLE(HR).VAINS);
OAG_LOADLIST_TABLE(L).FARM = ADDR(OAG_DATA_TABLE(HR).FARM);
OAG_LOADLIST_TABLE(L).NORTH = ADDR(OAG_DATA_TABLE(HR).NORTH);
OAG_LOADLIST_TABLE(L).EAST = ADDR(OAG_DATA_TABLE(HR).EAST);
OAG_LOADLIST_TABLE(L).SOUTH = ADDR(OAG_DATA_TABLE(HR).SOUTH);
OAG_LOADLIST_TABLE(L).WEST = ADDR(OAG_DATA_TABLE(HR).WEST);

ENDLOOP;

PERFORM DISPLAY_PANEL;

IF RSTATUS EQ PA1

THEN stop;

IF RSTATUS NE ENTER

THEN OAG_DATA = AUX_DATA;

ELSE

DM = FLDDEF;
DM(47) = FLDHIGH;
CALL GCHECK;
IN (OAGDMN, INDEX)

INPUT (OAG_DATA, CNVRT.OAG_CURSOR);

[This Routine checks for errors occurred on screen as a result of an erroneous entry
and returns value for the cursor pointing to first data field where an error has
occurred, and an appropriate screen message is issued advising user with corrections]

IF OAG_DATA.MSG NE 'DATA ENTERED'

THEN DM(CURSOR) = FLDHIGH;
CNVRT.OAG.SCROLL = 0;
ELSE
    CALL OVALID;
    IN (INDEX);

    INOUT (OAG_DATA, CVRT_OAG, CURSOR);
    [This routine performs data validation checks on screen entries and
    returns value for cursor pointing to first invalid data field. Also,
    an appropriate screen message is issued advising user with
    corrections]

    IF OAG_DATA_MSG NE 'DATA ENTERED'
        THEN
            DM(CURSOR) = FLDHIGH;
            CVRT_OAG.SCROLL = 0;
        ELSE
            OAG_DATA.SCROLL = (2) ' ';
            OAG_DATA.MSG = 'DATA ENTERED AT' CONCATENATE GMT;
            AUX_DATA = OAG_DATA;

    ENDREPEAT;

END SCREEN;
ROUTINE GCHECK

IN (OAGDATA, INDEX)

INPUT (OAGDATA, CNVRT OAG, CURSOR);

[This routine checks for errors occurred on screen as a result of an erroneous entry and returns value for cursor pointing to first data field where an error has occurred, and an appropriate screen message is issued advising user with corrections]

ERR1 = 'NUMERIC INPUT REQUIRED';
ERR2 = 'NON-NEGATIVE INPUT REQUIRED';
ERR3 = 'NO DECIMAL POINTS ALLOWED';
ERR5 = 'INPUT MUST BE X OR BLANK';

OAGDATA.MSG = 'DATA ENTERED';
CURSOR = 1;

ON CONVERSION BEGIN; [ON CONVERSION is a PL/I feature that is invoked if a character data is detected in a numerical data field]

OAGDATA.MSG = ERR1;
RETURN;

IF X(OAGDATA.INITIAL) NE 0
THEN OAGDATA.MSG = ERR5;
ELSE

IF OAGDATA.INITIAL NE (2)'
THEN

OAGDATA = OAG DATA; BY NAME;
OAGDATA.INITIAL = (2)'
OAGDATA.SCROLL = (4)'
CURSOR = 2;

PERFORM SCROLL_DATA_FIELD_ERROR_CHECK;
EXITIF [error detected]

REPEAT WHILE (OAG_DATA.MSG EQ 'DATA ENTERED'); [L = 0 to 3]

IF MR GT 23

THEN MR = MR - 24;

CURSOR = CURSOR + 2;

PERFORM TTLARE_DATA_FIELD_ERROR_CHECK;

EXITIF [error detected]

CURSOR = CURSOR + 1;

PERFORM TTLDEP_DATA_FIELD_ERROR_CHECK;

EXITIF [error detected]

PERFORM KIBBS_DATA_FIELD_ERROR_CHECK;

EXITIF [error detected]

CURSOR = CURSOR + 1;

PERFORM CGT_DATA_FIELD_ERROR_CHECK;

EXITIF [error detected]

PERFORM VAINE_DATA_FIELD_ERROR_CHECK;

EXITIF [error detected]

CURSOR = CURSOR + 1;

PERFORM PARDT_DATA_FIELD_ERROR_CHECK;

CURSOR = CURSOR + 1;

EXITIF [error detected]

CURSOR = CURSOR + 1;

EXITIF [error detected]

CURSOR = CURSOR + 1;

EXITIF [error detected]

CURSOR = CURSOR + 1;
PERFORM NORTH_DATA_FIELD_ERROR_CHECK;
EXITIF [error detected]
CURSOR = CURSOR + 1;
PERFORM EAST_DATA_FIELD_ERROR_CHECK;
EXITIF [error detected]
CURSOR = CURSOR + 1;
PERFORM SOUTH_DATA_FIELD_ERROR_CHECK;
EXITIF [error detected]
CURSOR = CURSOR + 1;
PERFORM WEST_DATA_FIELD_ERROR_CHECK;
EXITIF [error detected]

ENDREPEAT;
END GCHECK;

PROCESS SCROLL_DATA_FIELD_ERROR_CHECK
[This process checks for errors on scroll data field]
Get STRING (OAG_DATA.SCROLL) EDIT (CMVRTOAC.SCROLL);
IF VERIFY ('.', OAG_DATA.SCROLL) EQ 0
THEN OAG_DATA.MSG = ERR3;
END SCROLL_DATA_FIELD_ERROR_CHECK;
PROCESS TTLDEP_DATA_FIELD_ERROR_CHECK
            [This process checks for errors on total departure demand data field]
            Get STRING (OAG_DATA.TABLE(HR).TTLDEF) EDIT (CNVRT_OAG.TABLE(HR).TTLDEF);
            IF VERIFY ('-"', OAG_DATA.TABLE(HR).TTLDEF) EQ 0
                THEN OAG_DATA.MSG = ERR2;
            ELSEIF VERIFY ('\', OAG_DATA.TABLE(HR).TTLDEF) EQ 0
                THEN OAG_DATA.MSG = ERR3;
            END TTLDEP_DATA_FIELD_ERROR_CHECK;

PROCESS TTLARR_DATA_FIELD_ERROR_CHECK
            [This process checks for errors on total arrival demand data field]
            Get STRING (OAG_DATA.TABLE(HR).TTLARR) EDIT (CNVRT_OAG.TABLE(HR).TTLARR);
            IF VERIFY ('-"', OAG_DATA.TABLE(HR).TTLARR) EQ 0
                THEN OAG_DATA.MSG = ERR2;
            ELSEIF VERIFY ('\', OAG_DATA.TABLE(HR).TTLARR) EQ 0
                THEN OAG_DATA.MSG = ERR3;
            END TTLARR_DATA_FIELD_ERROR_CHECK;
PROCESS KUBBS_DATA_FIELD_ERROR_CHECK
[This process checks for errors on KUBBS data field]

Get STRING (OAG_DATA.TABLE(HR).KUBBS) EDIT (CMVT_OAG.TABLE(HR).KUBBS);

IF VERIFY ('-', OAG_DATA.TABLE(HR).KUBBS) EQ 0
    THEN OAG_DATA.MSG = ERR2;
ELSEIF VERIFY ('.', OAG_DATA.TABLE(HR).KUBBS) EQ 0
    THEN OAG_DATA.MSG = ERR3;
END KUBBS_DATA_FIELD_ERROR_CHECK;

PROCESS CGT_DATA_FIELD_ERROR_CHECK
[This process checks for errors on CGT data field]

Get STRING (OAG_DATA.TABLE(HR).CGT) EDIT (CMVT_OAG.TABLE(HR).CGT);

IF VERIFY ('-', OAG_DATA.TABLE(HR).CGT) EQ 0
    THEN OAG_DATA.MSG = ERR2;
ELSEIF VERIFY ('.', OAG_DATA.TABLE(HR).CGT) EQ 0
    THEN OAG_DATA.MSG = ERR3;
END CGT_DATA_FIELD_ERROR_CHECK;
PROCESS VAINS_DATA_FIELD_ERROR_CHECK  
[This process checks for errors on VAINS data field]
Get STRING (OAAG_DATA.TABLE(HR).VAINS) EDIT (CONVERT_OAAG_TABLE(ER).VAINS)
  IF VERIFY ('.', OAAG_DATA.TABLE(HR).VAINS) EQ 0
    THEN OAAG_DATA.MSG = ERR2;
  ELSEIF VERIFY ('-', OAAG_DATA.TABLE(HR).VAINS) EQ 0
    THEN OAAG_DATA.MSG = ERR3;
END VAINS_DATA_FIELD_ERROR_CHECK;

PROCESS FARM_DATA_FIELD_ERROR_CHECK  
[This process checks for errors on FARM data field]
Get STRING (OAAG_DATA.TABLE(HR).FARM) EDIT (CONVERT_OAAG_TABLE(ER).FARM)
  IF VERIFY ('.', OAAG_DATA.TABLE(HR).FARM) EQ 0
    THEN OAAG_DATA.MSG = ERR2;
  ELSEIF VERIFY ('-', OAAG_DATA.TABLE(HR).FARM) EQ 0
    THEN OAAG_DATA.MSG = ERR3;
END FARM_DATA_FIELD_ERROR_CHECK;
PROCESS NORTH_DATA_FIELD_ERROR_CHECK
[This process checks for errors on NORTH data field]
Get STRING (OAC_DATA.TABLE(HR).NORTH) EDIT (CNVRT_OAC_TABLE(HR).NORTH)
IF VERIFY ('.', OAC_DATA.TABLE(HR).NORTH) EQ 0
THEN OAC_DATA.MSG = ERR2;
ELSEIF VERIFY ('-', OAC_DATA.TABLE(HR).NORTH) EQ 0
THEN OAC_DATA.MSG = ERR3;
END NORTH_DATA_FIELD_ERROR_CHECK;

PROCESS EAST_DATA_FIELD_ERROR_CHECK
[This process checks for errors on EAST data field]
Get STRING (OAC_DATA.TABLE(HR).EAST) EDIT (CNVRT_OAC_TABLE(HR).EAST)
IF VERIFY ('.', OAC_DATA.TABLE(HR).EAST) EQ 0
THEN OAC_DATA.MSG = ERR2;
ELSEIF VERIFY ('-', OAC_DATA.TABLE(HR).EAST) EQ 0
THEN OAC_DATA.MSG = ERR3;
END EAST_DATA_FIELD_ERROR_CHECK;
PROCESS SOUTH DATA FIELD ERROR_CHECK
[This process checks for errors on SOUTH data field]
Get STRING (OAG_DATA.TABLE(HR).SOUTH) EDIT (CONV_OAG_DATA.TABLE(HR).SOUTH);
IF VERIFY ("." , OAG_DATA.TABLE(HR).SOUTH) EQ 0
THEN OAG_DATA.MSG = ERR2;
ELSEIF VERIFY ("--", OAG_DATA.TABLE(HR).SOUTH) EQ 0
THEN OAG_DATA.MSG = ERR3;
END SOUTH DATA FIELD_ERROR_CHECK;

PROCESS WEST DATA_FIELD_ERROR_CHECK
[This process checks for errors on WEST data field]
Get STRING (OAG_DATA.TABLE(HR).WEST) EDIT (CONV_OAG_DATA.TABLE(HR).WEST);
IF VERIFY ("." , OAG_DATA.TABLE(HR).WEST) EQ 0
THEN OAG_DATA.MSG = ERR2;
ELSEIF VERIFY ("--", OAG_DATA.TABLE(HR).WEST) EQ 0
THEN OAG_DATA.MSG = ERR3;
END WEST DATA FIELD_ERROR_CHECK;
ROUTINE CVALID

IN (INDEX);

INOUT (OAG_DATA, CHVRT_OAG, CURSOR);

[This routine performs data validation checks on screen entries and returns value for cursor pointing to first invalid data field. Also, an appropriate screen message is issued advising user with corrections]

$THREE = 3;
CURSOR = 2;
OAG_DATA.MSG = 'DATA ENTERED';
DOBERR = 'NUMBER OF AIRCRAFT MUST NOT EXCEED 99';
TITLE = 'TOTAL DOES NOT EQUAL SUM OF INDIVIDUAL ENTRIES';

REPEAT WHILE (OAG_DATA.MSG EQ 'DATA ENTERED'; [L = 0 To 3]
[check for demand values greater than 99]

HR = INDEX + L

IF HR GT 23
THEN HR = HR - 24;
SUN = 0.0;
CURSOR = CURSOR + 4;

IF CHVRT_OAG.TABLE(HR).KURSS GT 99.0

THEN OAG_DATA.MSG = DOBERR;
ELSE

OAG_DATA.TABLE(HR).KURSS = SUBSTR(P(CHVRT_OAG.TABLE(HR).KURSS,$THREE),1,3);
SUN = SUN + CHVRT_OAG.TABLE(HR).KURSS;
CURSOR = CURSOR + 1;

IF CHVRT_OAG.TABLE(HR).OCT GT 99.0
Then \( \text{OAG\_DATA\_MSG} = \text{ERROR} \);

Else

\[
\begin{align*}
\text{OAG\_DATA\_TABLE(\_HR).CCT} &= \text{SUBSTR}(\text{CNVRT\_OAG\_TABLE(\_HR).CCT}, \_STRIKE), 1, 3); \\
\text{SUM} &= \text{SUM} + \text{CNVRT\_OAG\_TABLE(\_HR).CCT}; \\
\text{CURSOR} &= \text{CURSOR} + 1;
\end{align*}
\]

\textbf{IF} \text{CNVRT\_OAG\_TABLE(\_HR).VAINS} \gt 99.0

\textbf{THEN}

\[ \text{OAG\_DATA\_MSG} = \text{ERROR} \]

\textbf{ELSE}

\[ \begin{align*}
\text{OAG\_DATA\_TABLE(\_HR).VAINS} &= \text{SUBSTR}(\text{CNVRT\_OAG\_TABLE(\_HR).VAINS}, \_STRIKE), 1, 3); \\
\text{SUM} &= \text{SUM} + \text{CNVRT\_OAG\_TABLE(\_HR).VAINS}; \\
\text{CURSOR} &= \text{CURSOR} + 1;
\end{align*} \]

\textbf{IF} \text{CNVRT\_OAG\_TABLE(\_HR).PABIOI} \gt 99.0

\textbf{THEN}

\[ \text{OAG\_DATA\_MSG} = \text{ERROR} \]

\textbf{ELSE}

\[ \begin{align*}
\text{OAG\_DATA\_TABLE(\_HR).PABIOI} &= \text{SUBSTR}(\text{CNVRT\_OAG\_TABLE(\_HR).PABIOI}, \_STRIKE), 1, 3); \\
\text{SUM} &= \text{SUM} + \text{CNVRT\_OAG\_TABLE(\_HR).PABIOI}; \\
\text{CURSOR} &= \text{CURSOR} + 1;
\end{align*} \]

\textbf{IF} \text{FLOOR(CNVRT\_OAG\_TABLE(\_HR).TTLARR)} \gt \text{SUM} [\text{check total against sum of individual demand values}]

\textbf{THEN}

\[ \text{OAG\_DATA\_MSG} = \text{ERROR} \]

\textbf{ELSE}

\[ \begin{align*}
\text{OAG\_DEM\_TABLE(\_HR).TTLARR} &= \text{SUBSTR}(\text{CNVRT\_OAG\_TABLE(\_HR).TTLARR}, \_STRIKE), 1, 3); \\
\text{SUM} &= 0.0; \\
\text{CURSOR} &= \text{CURSOR} + 6;
\end{align*} \]

\textbf{IF} \text{CNVRT\_OAG\_TABLE(\_HR).NORTH} \gt 99.0
THEN OAG_DATA(MSG = DORDER;
ELSE
OAG_DATA_TABLE(HR).NORTH = SUBSTR(F(CHRVT_OAG_TABLE(HR).NORTH, 'THREE'), 1, 3);
SUM = SUM + CHRVT_OAG_TABLE(HR).NORTH;
CURSOR = CURSOR + 1;
IF CHRVT_OAG_TABLE(HR).EAST GT 99.0 THEN OAG_DATA(MSG = DORDER;
ELSE
OAG_DATA_TABLE(HR).EAST
= SUBSTR(F(CHRVT_OAG_TABLE(HR).EAST, 'THREE'), 1, 3);
SUM = SUM + CHRVT_OAG_TABLE(HR).EAST;
CURSOR = CURSOR + 1;
IF CHRVT_OAG_TABLE(HR).SOUTH GT 99.0 THEN OAG_DATA(MSG = DORDER;
ELSE
OAG_DATA_TABLE(HR).SOUTH
= SUBSTR(F(CHRVT_OAG_TABLE(HR).SOUTH, 'THREE'), 1, 3);
SUM = SUM + CHRVT_OAG_TABLE(HR).SOUTH;
CURSOR = CURSOR + 1;
IF CHRVT_OAG_TABLE(HR).WEST GT 99.0 THEN OAG_DATA(MSG = DORDER;
 ELSE
ELSE
OAG_DATA_TABLE
(HR).WEST =
SUBSTR(P(CHVRT
OAG_TABLE
(HR).WEST,
$THREE),(1,3));
CURSOR =
CURSOR - 5;
IF
FLOOR(CHVRT_
OAG_TABLE(HR).TT
LDEP) NE SUM
THEN OAG
DATA_MSG
= TITLE2;
ELSE
CURSOR=
CURSOR+8;
OAG
DATA_TABLE(HR).TTLDEP
= SUBSTR(P(CHVRT
OAG_TABLE(HR).TTLDEP,
$THREE),(1,3));
ENDREPEAT;
IF OAG_DATA_MSG EQ 'DATA ENTERED'
THEN CURSOR = 2;
END GVALID;
2.9 Airport Status Screen

The following pages, 2-265 to 2-278, describe the processing for
the Airport Status Screen.
***LOCAL VARIABLES***

STRUCTURE ARPT DATA(2) LIKE APTSTAT
   [This structure is similar to APTSTAT used as a working area within the screen routine]
ENDSTRUCTURE;

STRUCTURE CNVRT APT(2) LIKE CNVTAPT
   [This structure is similar to CNVTAPT used as a working area within the screen routine]
ENDSTRUCTURE;

CHAR MIDDATA(2)   [this variable is similar to MIDFLAG used as a working area within the screen routine]

INT SWITCH(2)     [this variable is used for switching between current and forecast screens, initialized to (2, 1)]

STRUCTURE AIRPORT LOADLIST [a structure of pointers, one for each data field on the screen used by panel manager for loading and unloading to and from screen]

PTR TIME [pointer for environment data field]

GROUP WX
   PTR CEIL [pointer for ceiling data field]
   PTR VIS [pointer for visibility data field]

GROUP WIND
   PTR DIR [pointer for wind direction data field]
   PTR VEL [pointer for wind velocity data field]

PTR MIDWAY [pointer for MIDWAY data field]
GROUP RUNWAY(12)

GROUP TOWER

PTR ARR [pointer for tower imposed arrival runway closures data field]

PTR DEP [pointer for tower imposed departure runway closures data field]

PTR SURF [pointer for surface conditions data field]

PTR BAK [pointer for braking condition data field]

PTR BVR [pointer for BVR reading data field]

PTR DIR [pointer for runway wind direction data field]

PTR VEL [pointer for runway wind velocity data field]

PTR CRSS [pointer for crosswind component data field]

PTR TAIL [pointer for tailwind component data field]

PTR CHIL [pointer for ceiling minima data field]

PTR VIS [pointer for visibility minima data field]

GROUP CLOSED

PTR ARR [pointer for arrival runway closure data field]

PTR DEP [pointer for departure runway closure data field]

PTR MSG [pointer for screen message data field]

BITS FENCE [32 bit variable as prescribed by DNS manual, initialised to string of (32) '1' B]

ENDSTRUCTURE;
ROUTINE ARPT

IN (CNVTPIN);

INOUT (APTSTAT, MIDFLAG, CNVTAPT, ESTATUS, 1);

(This routine invokes airport status screen for both current and forecast environment)

ARPT_DATA = APTSTAT;
MIDDATA = MIDFLAG;
CNVRT_APT = CNVTAPT;

REPEAT UNTIL (ESTATUS NE PP12);

ARPT_DATA(1) = APTSTAT(1);
CNVRT_APT(1) = CNVTAPT(1);
MIDDATA(1) = MIDFLAG(1);

REPEAT UNTIL (ESTATUS NE PP3);

I = SWITCH(I);

CALL ASCREEN;

IN (CNVTPIN);

INOUT (ARPT_DATA(I), MIDDATA(I), CNVRT_APT(I), ESTATUS);

[this routine controls airport status screen]

ENDDO;

[This routine controls airport status screen]

ENDREPEAT;

END LOOP;

LOOP; [J = 1 To 2]

IF SUBST(AZPTDATA(J), MSG, 1, 12) EQ 'DATA ENTERED'

THEN

APTSTAT(J) = ARPT_DATA(J);
MIDFLAG(J) = MIDDATA(J);
CNVRT_APT(J) = CNVRT_APT(J);

END LOOP;

END ARPT;
ROUTINE ASCREEM

IN (CNVTPM); 

INPUT (ARPT_DATA(I), MIDDATA(I), CHRT_APT(I), RSTATUS);
[This routine controls airport status screen]

CHAR FRAME [character variable of length 8 containing the name of DNS panel initialized to 'AIRPORT', the name of the panel that controls airport status screen]

INT CURSOR [integer variable containing the cursor's position on the screen]

BITS DM(163) [8 bit variable of data mask used in DNS]

STRUCTURE AUX_DATA LIKE ARPT_DATA(I)
ENDSTRUCTURE;

STRUCTURE AUX_CHK LIKE CHRT_APT(I)
ENDSTRUCTURE;

CHAR AUX_CID;

CURSOR = 2; [set cursor to position 2; first data field used by the user]

DM = FLDDEF; [set data masks to default intensity (normal)]

DM(1) = FLMHIGH; [set first data field to high intensity]

DM(163) = FLMHIGH; [set last data field to high intensity]

AUX_DATA = ARPT_DATA(I);

AUX_CID = MIDFLAG(I);

AUX_CHK = CHRT_APT(I);

PERFORM SET_UP_SCREEN_POINTERS,(ARPT);
REPEAT UNTIL (RSTATUS NE ENTER);

PERFORM DISPLAY_PANEL;

IF RSTATUS EQ FAIL

THEN stop;

IF RSTATUS NE ENTER

THEN

ARPT_DATA(I) = AUX_DATA;
MIDDATA(I) = AUX Mid;
CNVRT_APT(I) = AUX CNVT;

ELSE

LOOP: [J = 2 To 162]

DM(J) = FLDDEF;

ENDLOOP;

CALL ACHECK;

INOUT (ARPT_DATA(I), MIDDATA(I), CNVRT_APT, CURSOR);

(This routine checks for errors occurred on screen as a result of an erroneous
entry and returns value for cursor pointing to first data field where an error
has occurred and an appropriate screen message is issued advising user with
corrections)

IF ARPT_DATA(I).NSG NE 'DATA ENTERED'

THEN

DM(CURSOR) = FLDMID;

ELSE

CALL AVVALID;
INOUT (ARPT_DATA(I), MIDDATA(I), CNVRT_APT(I), CURSOR);  
[This routine performs data validation checks on screen entries and returns value for cursor pointing to first invalid data field. Also, an appropriate screen message is issued advising user with corrections]

IF ARPT_DATA(I).MSG NE 'DATA ENTERED'
THEN
   DM(CURSOR) = FLDHIGH;
ELSE
   CALL AUPDATE;
   IN (CNVTFRM);
   INOUT (ARPT_DATA(I), MIDDATA(I));  
   [This routine performs local updates on screen]
   ARPT_DATA(I).MSG = 'DATA ENTERED AT 'COMCATEMAAY CMT; 
   AUX_DATA = ARPT_DATA(I); 
   AUX_MID = MIDDATA(I); 
   CNVT_AUX = CNVRT_APT(I);
ENDREPEAT;
END SCREEN;
PROCESS SET_UP_SCREEN_POINTERS (ARPT)

This process sets up screen pointers for SMS use

  AIRPORT_LOADLIST.TIME = ADDR(ARPT_DATA(I).TIME);
  AIRPORT_LOADLIST.WX.CEIL = ADDR(ARPT_DATA(I).WX.CEIL);
  AIRPORT_LOADLIST.WX.VIS = ADDR(ARPT_DATA(I).WX.VIS);
  AIRPORT_LOADLIST.WIND.DIR = ADDR(ARPT_DATA(I).WIND.DIR);
  AIRPORT_LOADLIST.WIND.VEL = ADDR(ARPT_DATA(I).WIND.VEL);
  AIRPORT_LOADLIST.MIDWAY = ADDR(MIDDATA(I));

  LOOP: [J = 1 To 12]
  AIRPORT_LOADLIST.RUNWAY(J).TOWER.ARR = ADDR(ARPT_DATA(I).RUNWAY(J).TOWER.ARR);
  AIRPORT_LOADLIST.RUNWAY(J).TOWER.DEF = ADDR(ARPT_DATA(I).RUNWAY(J).TOWER.DEF);
  AIRPORT_LOADLIST.RUNWAY(J).SURF = ADDR(ARPT_DATA(I).RUNWAY(J).SURF);
  AIRPORT_LOADLIST.RUNWAY(J).BREAK = ADDR(ARPT_DATA(I).RUNWAY(J).BREAK);
  AIRPORT_LOADLIST.RUNWAY(J).RVR = ADDR(ARPT_DATA(I).RUNWAY(J).RVR);
  AIRPORT_LOADLIST.RUNWAY(J).DIR = ADDR(ARPT_DATA(I).RUNWAY(J).DIR);
  AIRPORT_LOADLIST.RUNWAY(J).VEL = ADDR(ARPT_DATA(I).RUNWAY(J).VEL);
  AIRPORT_LOADLIST.RUNWAY(J).CSS = ADDR(ARPT_DATA(I).RUNWAY(J).CSS);
  AIRPORT_LOADLIST.RUNWAY(J).TAIL = ADDR(ARPT_DATA(I).RUNWAY(J).TAIL);
  AIRPORT_LoadLIST.RUNWAY(J).CLOSED.ARR = ADDR(ARPT_DATA(I).RUNWAY(J).CLOSED.ARR);
  AIRPORT_LOADLIST.RUNWAY(J).CLOSED.DEF = ADDR(ARPT_DATA(I).RUNWAY(J).CLOSED.DEF);

  ENDLOOP:

  AIRPORT_LOADLIST.MSG = ADDR(ARPT_DATA(I).MSG);

END SET_UP_SCREEN_POINTERS (ARPT);
ROUTINE ACHECK

`INPUT (ARPT_DATA(I),MIDDATA(I),CONVRT_APT(I),CURSOR);`

This routine checks for errors occurred on screen as a result of an erroneous entry and returns
value for cursor pointing to first data field where an error has occurred; and an appropriate
screen message is issued advising user with corrections.

ERR1 = 'NUMERIC INPUT REQUIRED';
ERR2 = 'NON-NEGATIVE INPUT REQUIRED';
ERR3 = 'NO DECIMAL POINTS ALLOWED';
ERR5 = 'INPUT MUST BE X OR BLANK';

ARPT_DATA(I).MSG = 'DATA ENTERED';

ON CONVERSION BEGIN: [ON CONVERSION is a PL/I feature that is invoked if a character data is
detected in a numerical data field]

ARPT_DATA(I).MSG = ERR1;

RETURN;
CURSOR = 2;
PERFORM CEIL_DATA_FIELD_ERROR_CHECK;

EXITIF [error detected]
CURSOR = 3;
PERFORM VIL_DATA_FIELD_ERROR_CHECK;

EXITIF [error detected]
CURSOR = 4;
PERFORM WIND_DIR_DATA_FIELD_ERROR_CHECK;

EXITIF [error detected]
CURSOR = 5;
PERFORM WIND_VEL_DATA_FIEID_ERROR_CHECK;
EXITIF [error detected]
CURSOR = 6;
IF X (MIDDATA(I)) NE 0
  THEN ARPT_DATA(I).NSG = ERR5;
  ELSE
    REPEAT WHILE (ARPT_DATA(I).NSG = 'DATA ENTERED'); [J = 1 to 12]
      CURSOR = CURSOR + 1;
      IF X(ARPT_DATA(I).RUNWAY(J).TOMER.ARR) NE 0
        THEN ARPT_DATA(I).NSG = ERR5;
        ELSE CURSOR = CURSOR + 1;
      IF X(ARPT_DATA(I).RUNWAY(J).TOMER.DEP) NE 0
        THEN ARPT_DATA(I).NSG = ERR5;
        ELSE CURSOR = CURSOR + 1;
      IF X(ARPT_DATA(I).RUNWAY(J).SURF) NE 0
        THEN ARPT_DATA(I).NSG = ERR5;
        ELSE CURSOR = CURSOR + 1;
      IF X(ARPT_DATA(I).RUNWAY(J).REL) NE 0
        THEN ARPT_DATA(I).NSG = ERR5;
        ELSE CURSOR = CURSOR + 9;
      ENDREPEAT;
    IF ARPT_DATA(I).NSG = 'DATA ENTERED'
      THEN CURSOR = 2;
  ENDIF;
END ACHECK;
PROCESS CEIL_DATA_FIELD_ERROR_CHECK
[This process checks for errors on ceiling data field]
Get STRING (ARPT_DATA(I).WX.CEIL) EDIT (CHVWT_DATA(I).WX.CEIL);
IF CHVWT_DATA(I).WX.CEIL LT 0.0
    THEN ARPT_DATA(I).MSG = ERR3;
    ELSEIF VERIFY(' ', ARPT_DATA(I).WX.CEIL) EQ 0
        THEN ARPT_DATA(I).MSG = ERR3;
END CEIL_DATA_FIELD_ERROR_CHECK;

PROCESS VIS_DATA_FIELD_ERROR_CHECK
[This process checks for errors on visibility data field]
Get STRING (ARPT_DATA(I).WX.VIS) EDIT (CHVWT_DATA(I).WX.VIS);
IF CHVWT_DATA(I).WX.VIS LT 0.0
    THEN ARPT_DATA(I).MSG = ERR3;
END VIS_DATA_FIELD_ERROR_CHECK;

PROCESS WIND_DIR_DATA_FIELD_ERROR_CHECK
[This process checks for errors on wind direction data field]
Get STRING (ARPT_DATA(I).WX.DIR) EDIT (CHVWT_DATA(I).WX.DIR);
IF CHVWT_DATA(I).WX.DIR LT 0.0
    THEN ARPT_DATA(I).MSG = ERR3;
    ELSEIF VERIFY(' ', ARPT_DATA(I).WX.DIR) EQ 0
        THEN ARPT_DATA(I).MSG = ERR3;
END WIND_DIR_DATA_FIELD_ERROR_CHECK;
PROCESS WIND_VEL_DATA_FIELD_ERROR_CHECK
   [This process checks for errors on wind velocity data field]
   GET STRING (APRT_DATA(I), WIND_VEL) EDIT (CNVRT_DATA(I), WIND_VEL);
   IF CNVRT_DATA(I), WIND_VEL LT 0.0
      THEN APRT_DATA(I), MSG = ERR2;
   ELSEIF VERIFY ('.', APRT_DATA(I), WIND_VEL) EQ 0
      THEN APRT_DATA(I), MSG = ERR3;
   END WIND_VEL_DATA_FIELD_ERROR_CHECK;
ROUTINE AVALID

INPUT (ARPT_DATA(I), MIDDATA(I), CNVRT_APT(I), CURSOR);

(This routine performs data validation checks on the screen entries and returns the value for cursor pointing to the first invalid data field. Also, an appropriate screen message is issued advising the user with corrections)

$TWO = 2;
$THREE = 3;
$FOUR = 4;
CURSOR = 2;

ARPT_DATA(I).WX.CEIL = SUBSTR(F(CNVRT_APT(I).WX.CEIL,$FOUR), 1,4);
CURSOR = 3;

IF CNVRT_APT(I).WX.VIS GE 100.0
THEN ARPT_DATA(I).MSG = 'VISIBILITY MUST BE LESS THAN 100 MILES'
ELSE C = F(100.0 = CNVRT_APT(I).WX.VIS,$FOUR);
    IF CNVRT_APT(I).WX.VIS LT 10.0
THEN ARPT_DATA(I).WX.VIS = SUBSTR(C,2,1) CONCATENATE ' ' CONCATENATE SUBSTR (C,3,2);
ELSE ARPT_DATA(I).WX.VIS = SUBSTR(C,1,2) CONCATENATE ' ' CONCATENATE SUBSTR (C,3,1);
CURSOR = 4;

IF CNVRT_APT(I).WIND.DIR GE 360.0
THEN ARPT_DATA(I).MSG = 'WIND DIRECTION MUST BE LESS THAN 360 DEGREES';
ELSE C = TRANSLATE (F(CNVRT_APT(I).WIND.DIR,$THREE), '0',' '); ARPT_DATA(I).WIND.DIR = SUBSTR (C,1,3);
CURSOR = 5;
ARPT_DATA(I).WIND.VEL = SUBSTR(F(CNVRT_APT(I).WIND.VEL,$TWO),1,3);
REPEAT WHILE (ARPT_DATA(I).MSG EQ 'DATA ENTERED'); [J = 1 To 12]
CURSOR = 13 * J - 4;
IF (ARPT_DATA(I).RUNWAY(J).SURF EQ (2) 'a') AND (ARPT_DATA(I).RUNWAY(J).BRE Eq (2) 'b')
THEN ARPT_DATA(I).MSG = 'SURFACE AND BRAKING CONDITIONS ARE NOT CONSISTENT';
ENDREPEAT;
IF ARPT_DATA(I).MSG EQ 'DATA ENTERED'
THEN CURSOR = 2;
END VALID;
ROUTINE  AUPDATE

IN (CNYTPRN);

INOUT (ARPT_DATA(1), KIDDATA(1), CVRT_APT(1));

[This routine performs local updates on screen]

CALL  WIND;

INOUT (ARPT_DATA(1), CVRT_APT(1));

[This routine computes crosswind and tailwind components of prevailing wind and sets up corresponding screen data fields]

CALL  CLOSING;

IN (CNYTPRN);

INOUT (ARPT_DATA(1), CVRT_APT(1));

[This routine closes runways based on wind conditions and weather minima]

END  AUPDATE;
2.10 Runway Equipment Status Screen

Pages 2-280 to 2-289 describe the processing for the Runway Equipment Status Screen.
LOCAL VARIABLES

STRUCTURE RAW_DATA(2) LIKE RAW_EQP
    [This structure is similar to RAW_EQP used as a working area within the screen routine]
ENDSTRUCTURE;

INT SWITCH(2)
    [This variable is used for switching between current and forecast screens, initialized to (2,1)]
ENDSTRUCTURE;

STRUCTURE RAW_LOADLIST
    [A structure of pointers, one for each data field on the screen used by panel manager for loading and unloading to and from screen]

PTR TIME
    [pointer for environment data field]

GROUP RUNWAY(12)

    PTR CATII [pointer for CATII data field]
    PTR LOC [pointer for localizer data field]
    PTR GS [pointer for glide slope data field]
    PTR OM [pointer for outer marker data field]
    PTR NM [pointer for middle marker data field]
    PTR IM [pointer for inner marker data field]
    PTR RAIL [pointer for RAIL data field]
    PTR ALS [pointer for ALS data field]
    PTR BVR [pointer for BVR data field]
    PTR HIRL [pointer for HIRL data field]
    PTR CL [pointer for centerline lights data field]
    PTR TDZ [pointer for TDZ data field]
    PTR NDB_VOR [pointer for NDB_VOR data field]

PTR MSG
    [pointer for screen message data field]

BITS FENCE
    [32 bit variable as prescribed by RMS manual, initialized to string of (32) '1's]
ENDSTRUCTURE;
ROUTINE RKY

INOUT (RKYEQP, RSTATUS, I);
[This routine invokes runway equipment status screen for both current and forecast environment]

RKY_DATA = RKYEQP;

REPEAT UNTIL (RSTATUS NE PF12);
RKY_DATA(I) = RKYEQP(I);
I = SWITCH(I);
[switch between two screens]

REPEAT UNTIL (RSTATUS NE PF4);
I = SWITCH(I);

CALL SCREEN;

INOUT (RKY_DATA(I), RSTATUS);
[This routine controls runway equipment status screen]

ENDREPEAT;

ENDREPEAT;

LOOP: [I = 1 to 2]

IF SUBSTR(RKY_DATA(J), MSG, 1, 12) EQ 'DATA ENTERED'

THEN RKYEQP(J) = RKY_DATA(J);

ENDLOOP;

END RKY;
ROUTINE RSSCREEN

INOUT (RWY_DATA(1), RSTATUS);
[This routine controls runway equipment status screen]

CHR FNNAME [character variable of length 8 containing name of RMS panel initialized to 'RUNWAY', name of panel that controls runway equipment status screen]

INT CURSOR [integer variable containing cursor's position on screen]

BITS DM(156) [8 bit variable of data mask used in RMS]

STRUCTURE AUX_DATA LIKE RWY_DATA(1)
END STRUCTURE;

CURSOR = 3; [set cursor to position 3; first data field used by user]

DM = FLDDDEF; [set data fields to default intensity (normal)]

DM(1) = FLDMHIGH; [set first data field to high intensity]

DM(156) = FLDMHIGH; [set last data field to high intensity]

AUX_DATA = RWY_DATA(1);

PERFORM SET_UP_SCREEN_POINTERS(RENT);

REPEAT UNTIL (RSTATUS NE ENTER);

PERFORM DISPLAY_PANEL;

IF RSTATUS EQ PAL
THEN stop;

IF RSTATUS NE ENTER
THEN RWY_DATA(1) = AUX_DATA;
ELSE

2-282
LOOP; [J = 2 To 157]
DM(J) = FLDDEF;
ENDLOOP;
CALL RCHECK;

INOUT (RNY_DATA(1), CURSOR);

[This routine checks for errors occurred on screen as a result of erroneous entry and returns value for cursor pointing to first data field where an error has occurred, and an appropriate screen message is issued advising user with corrections]

IF RNY_DATA(1).MSG NE 'DATA ENTERED'
THEN DM(CURSOR) = FLDHIGH;
ELSE

CALL RUPDATE;

INOUT (RNY_DATA(1));

[This routine performs local updates on screen]

RNY_DATA(1).MSG = 'DATA ENTERED AT ' CONCATENATE GMT;
AUX_DATA = RNY_DATA(1);

ENDREPEAT;

END BSSCREEN;
PROCESS SET_UP_SCREEN_POINTERS (RUNY)
[This process sets up screen pointers for RMS use]

RUNY_LOADLIST.TIME = ADDR(RUNY_DATA(1).TIME);

LOOP: [J = 1 To 12]

RUNY_LOADLIST.RUNWAY(J).CATII = ADDR(RUNY_DATA(1).RUNWAY(J).CATII);

RUNY_LOADLIST.RUNWAY(J).LOC = ADDR(RUNY_DATA(1).RUNWAY(J).LOC);
RUNY_LOADLIST.RUNWAY(J).GS = ADDR(RUNY_DATA(1).RUNWAY(J).GS);
RUNY_LOADLIST.RUNWAY(J).HN = ADDR(RUNY_DATA(1).RUNWAY(J).HN);
RUNY_LOADLIST.RUNWAY(J).IM = ADDR(RUNY_DATA(1).RUNWAY(J).IM);
RUNY_LOADLIST.RUNWAY(J).RAIL = ADDR(RUNY_DATA(1).RUNWAY(J).RAIL);
RUNY_LOADLIST.RUNWAY(J).ALS = ADDR(RUNY_DATA(1).RUNWAY(J).ALS);
RUNY_LOADLIST.RUNWAY(J).RVR = ADDR(RUNY_DATA(1).RUNWAY(J).RVR);
RUNY_LOADLIST.RUNWAY(J).HRL = ADDR(RUNY_DATA(1).RUNWAY(J).HRL);
RUNY_LOADLIST.RUNWAY(J).CL = ADDR(RUNY_DATA(1).RUNWAY(J).CL);
RUNY_LOADLIST.RUNWAY(J).TWZ = ADDR(RUNY_DATA(1).RUNWAY(J).TWZ);
RUNY_LOADLIST.RUNWAY(J).NDB_VOR = ADDR(RUNY_DATA(1).RUNWAY(J).NDB_VOR);

ENDLOOP;

RUNY_LOADLIST.MSG = ADDR(RUNY_DATA(1).MSG);

END SET_UP_SCREEN_POINTERS (RUNY);
ROUTINE RCHECK

INOUT (RWY_DATA(I), CURSOR);

[this routine checks for errors occurred on screen as a result of an erroneous entry and returns
value for cursor pointing to first data field where an error has occurred, and an appropriate
screen message is issued advising user with corrections]

ERR5 = 'INPUT MUST BE X OR BLANK';
RWY_DATA(I).MSG = 'DATA ENTERED';

REPEAT WHILE (RWY_DATA(I).MSG EQ 'DATA ENTERED'); [J = 1 to 12]
CURSOR = 13 * J - 11;

IF (J EQ 5 OR J EQ 6) AND X(RWY_DATA(I).RUNWAY(J).CATII) NE 0
  THEN RWY_DATA(I).MSG = ERR5;
ELSE
  CURSOR = CURSOR + 1;

IF X(RWY_DATA(I).RUNWAY(J).LOC) NE 0
  THEN RWY_DATA(I).MSG = ERR5;
ELSE
  CURSOR = CURSOR + 1;

IF (J NE 2) AND X(RWY_DATA(I).RUNWAY(J).GS) NE 0
  THEN RWY_DATA(I).MSG = ERR5;
ELSE
  CURSOR = CURSOR + 1;

IF (J NE 2) AND X(RWY_DATA(I).RUNWAY(J).NH) NE 0
  THEN RWY_DATA(I).MSG = ERR5;
ELSE
  CURSOR = CURSOR + 1;
IF (J EQ 5 OR J EQ 6) AND X(RWy_DATA(I).RUNWAY(J).IN) NE 0 THEN RWy_DATA(I).MSG = ERR5;
ELSE
CURSOR = CURSOR + 1;
IF (J NE 2 AND X(RWy_DATA(I).RUNWAY(J).RAIL) NE 0 THEN RWy_DATA(I).MSG = ERR5;
ELSE
CURSOR = CURSOR + 1;
IF (J NE 1 AND J NE 2 AND J NE 7 AND J NE 8) AND X(RWy_DATA(I).RUNWAY(J).EVR) NE 0 THEN RWy_DATA(I).MSG = ERR5;
ELSE
CURSOR = CURSOR + 1;
IF (J NE 1) AND (J NE 2) AND (J NE 4) AND (J NE 7) AND (J NE 8) AND (J NE 9) AND X(RWy_DATA(I).RUNWAY(J).CL) NE 0 THEN RWy_DATA(I).MSG = ERR5;
ELSE
CURSOR = CURSOR + 1;
IF (J EQ 5) OR (J EQ 6) OR (J EQ 11) OR (J EQ 12) AND X(RWy_DATA(I).RUNWAY(J).TBE) NE 0 THEN RWy_DATA(I).MSG = ERR5;
ELSE
CIRSOR +1* ;
ip
-4)

(JNME1) An(JN
BbD (JME 8) AND

(I 1F10) WDbX(Ew
DATxrT).
RUHH&Y(J) .NDbVOR)
HE 0

TaEN

IvY D&T(I).MSG - S

I')IIA

IF

AllYDATA(I.NSC THEN

END

CURSOI.

3;

'DATAENTEIED*
(if

no errors detected return Cursor to top!

&CHEMZ;

to


ROUTINE UPDATE

INPUT (RWY_DATA(I));

[This Routine performs local updates on screen]

LOOP: [J = 5 TO 6]

IF RWY_DATA(I).RUNWAY(J).LOC NE ( )
THEN RWY_DATA(I).RUNWAY(J).CATII = 'X';

IF RWY_DATA(I).RUNWAY(J).GS NE (2)
THEN RWY_DATA(I).RUNWAY(J).CATII = 'X';

IF RWY_DATA(I).RUNWAY(J).ON NE (2)
THEN RWY_DATA(I).RUNWAY(J).CATII = 'X';

IF RWY_DATA(I).RUNWAY(J).IN NE (2)
THEN RWY_DATA(I).RUNWAY(J).CATII = 'X';

IF RWY_DATA(I).RUNWAY(J).ALS NE (2)
THEN RWY_DATA(I).RUNWAY(J).CATII = 'X';

IF RWY_DATA(I).RUNWAY(J).NIEL NE (2)
THEN RWY_DATA(I).RUNWAY(J).CATII = 'X';
IF RWY_DATA(I).RUNWAY(J).CL NE (2) THEN
    IF RWY_DATA(I).RUNWAY(J).TDZ NE (2) THEN
        RWY_DATA(I).RUNWAY(J).CLII = 'X';
    ENDloop;
END RUPDATE;
2.11 Demand Profile Screen

The Demand Profile Screen is described in pages 2-291 to 2-309.
**LOCAL VARIABLES**

```plaintext
STRUCTURE DEMND_DATA(2) LIKE DEMAND
  [This structure is similar to DEMAND used as a working area within screen routine]
ENDSTRUCTURE;

STRUCTURE CNVRT_DYN(2) LIKE CNVTDYN
  [This structure is similar to CNVTDYN used as a working area within screen routine]
ENDSTRUCTURE;

INT SWITCH(2) [This variable is used for switching between current and forecast screens, initialized to (2,1)]

STRUCTURE DEMND_LOADLIST [A structure of pointers, one for each data field on screen used by panel manager for loading and unloading to and from screen]
  PTR TIME [pointer for environment data field]
  PTR FROM [pointer for beginning time data field]
  PTR TO [pointer for ending time data field]
  PTR RETRIEVE [pointer for retrieve function data field]
  GROUP ARR
    PTR TOTAL [pointer for total arrival demand data field]
    PTR KURBS [pointer for KURBS arrival demand data field]
    PTR COO [pointer for COO arrival demand data field]
    PTR VAINS [pointer for VAINS arrival demand data field]
    PTR FARMM [pointer for FARMM arrival demand data field]
```
GROUP DEF

PTR TOTAL [pointer for total departure demand data field on demand profile screen]

PTR NORTH [pointer for NORTH departure demand data field on demand profile screen]

PTR EAST [pointer for EAST departure demand data field on demand profile screen]

PTR SOUTH [pointer for SOUTH departure demand data field on demand profile screen]

PTR WEST [pointer for WEST departure demand data field on demand profile screen]

PTR MEG [pointer for screen message data field]

BIT3 FRACK [32 bit variable as prescribed by DMS manual, initialized to string of (32) '1'B]

ENDSTRUCTURE

2-32
ROUTINE DMND

IN (CHYTOAG);

INOUT (DEMAND, CNVTDEM, ESTATUS, I);

This routine invokes demand profile screen for both current and forecast environments.

DMND_DATA = DEMAND;

CNVTDEM = CNVTDEM;

REPEAT UNTIL (STATUS NE PF12);

DMND_DATA(I) = DEMAND(I);

CNVTDEM(I) = CNVTDEN(I);

I = SWITCH(I);

REPEAT UNTIL (STATUS NE PF5);

I = SWITCH(I); [switch between two screens]

CALL DSCREEN;

IN (CHYTOAG);

INOUT (DMND_DATA(I), CNVTDEN(I), ESTATUS, I);

This routine controls demand profile status screen.

ENDREPEAT;

ENDREPEAT;

LOOP: [I = 1 to 2]

IF SUBSTR(DMND_DATA(J), MSG, 1, 12) EQ 'DATA ENTERED'

THEN

DEMAND(J) = DMND_DATA(J);

CNVTDEN(J) = CNVTDEN(J);

ENDLOOP;

END DMND;
ROUTINE DSSCREEN
IN (CONTOAG);

CALL (DMND_DATA(I), CNVRT_DNM(I), RSTATUS, I);

*This routine controls demand profile screen*

CHAR PNAME [character variable of length 8 containing name of DMS panel initialized to 'DEMAND',
name of panel that controls demand profile screen]

INT CURSOR [integer variable containing cursor's position on screen]

BIT DM(15) [8 bit variable of data mask used in DMS]

FLT OFFSET(2)

STRUCTURE AUX_DATA LIKE DMND_DATA(I)
ENDSSTRUCTURE;

STRUCTURE AUX_CNTY LIKE CNVRT_DNM(I)
ENDSSTRUCTURE;

CURSOR = 4; [set cursor to position 4; first data field used by user]

DM = FLDOFF; [set data fields to default intensity (normal)]

DM(1) = FLDOFF; [set first data field to high intensity]

DM(15) = [set last data field to high intensity]

RETRIEVE = (2) ' '; AUX_DATA = DMND_DATA(I); AUX_CNTY = CNVRT_DNM(I);

Get STRING(GMT) EDIT (CPROM); [get current time]

CPROM = MOD(CPROM + OFFSET(1), 2400.0);
CFO = MOD (CPROM + 100.0, 240.0);
CPROM = TRANSLATE (SUBSTR(P(CPROM,CPROM),1,4),'0','');
PERFORM SET_UP_SCREEN_POINTERS (DMOD);  
REPEAT UNTIL (ESTATUS NE ENTER);  
PERFORM DISPLAY_PANEL;  
IF ESTATUS EQ PAL  
THEN stop;  
ELSE  
IF ESTATUS NE ENTER  
THEN DMOD_DATA(I) = AUX_DATA;  
CONVRT_DATA(I) = AUX_CONV;  
ELSE  
LOOP; (j = 4 To 12)  
DM(I) = FLDWP;  
ENDLOOP;  
CALL DCHECK;  
IN (CHYTOAG);  

INPUT (DMOD_DATA(I), RETRIEVE, CONVRT_DATA(I), CFROM, CTO, CURSOR);  
(This routine checks for errors occurred on screen as a result of erroneous entry and returns value for cursor pointing to first data field where an error has occurred; and an appropriate screen message is issued advising user with corrections)  
IF DMOD_DATA(I).MSG NE 'DATA ENTERED'  
THEN  
DM(CURSOR) = FLDMAP;  
ELSE  
CALL DVALID;  
IN (CHYTOAG);
This routine performs data validation checks on screen entries and returns value for cursor pointing to first invalid data field. Also, an appropriate screen message is issued advising user with corrections.

IF DMD_DATA(I).MSG NE 'DATA ENTERED'
THEN DM(CURSOR) = PDEMIC;
ELSE
DMD_DATA(I).MSG = 'DATA ENTERED AT CONCATENATE GMX';
AUX_DATA = DMD_DATA(I);
AUX_DATA = CVNT_DEM(I);
ENDREPEAT;
END DSCEREE;
PROCESS SET_UP_SCREEN_POINTERS(DMND);

This process sets up screen pointers for DNS use

DMND_LOADLIST.TIME = ADDR(DMND_DATA(I).TIME);
DMND_LOADLIST.FROM = ADDR(FROM);
DMND_LOADLIST.TO = ADDR(TO);
DMND_LOADLIST.RETRIEVE = ADDR(RETRIEVE);
DMND_LOADLIST.ARR.TOTAL = ADDR(DMND_DATA(I).ARR.TOTAL);
DMND_LOADLIST.ARR.KURBS = ADDR(DMND_DATA(I).ARR.KURBS);
DMND_LOADLIST.ARR.CUT = ADDR(DMND_DATA(I).ARR.CUT);
DMND_LOADLIST.MR.VAINS = ADDR(DMND_DATA(I).MR.VAINS);
DMND_LOADLIST.MR.PARM = ADDR(DMND_DATA(I).MR.PARM);
DMND_LOADLIST.MR.TOTAL = ADDR(DMND_DATA(I).MR.TOTAL);
DMND_LOADLIST.MR.NORTH = ADDR(DMND_DATA(I).MR.NORTH);
DMND_LOADLIST.MR.EAST = ADDR(DMND_DATA(I).MR.EAST);
DMND_LOADLIST.MR.SOUTH = ADDR(DMND_DATA(I).MR.SOUTH);
DMND_LOADLIST.MR.WEST = ADDR(DMND_DATA(I).MR.WEST);
DMND_LOADLIST.MR.HLG = ADDR(DMND_DATA(I).MR.HLG);

END SET_UP_SCREEN_POINTERS(DMND);
ROUTINE DCHECK

IN (CNVTOAG);

INPUT (DNDDATA(I), RETRIEVE, CNVDT DEM(I), CPROM, CTID, CURSOR);

This routine checks for errors occurred on screen as a result of erroneous entry and returns value
for cursor pointing to first data field where an error has occurred; and an appropriate screen
message is issued advising user with corrections.

ERR1 = 'NUMERIC INPUT REQUIRED';
ERR2 = 'NON-NEGATIVE INPUT REQUIRED';
ERR3 = 'NO DECIMAL POINTS ALLOWED';
ERR5 = 'INPUT MUST BE X OR BLANK';

ON CONVERSION BEGIN; [ON CONVERSION is a PL/I feature that is invoked when a character data is
detected in a numerical data field]

DNDDATA(I).MSG = ERR1;

RETURN;

DNDDATA(I).MSG = 'DATA ENTERED';
CURSOR = 4;

IF X (RETRIEVE) NE 0

THEN DNDDATA(I).MSG = ERR5;
ELSEIF RETRIEVE NE (2) ' ' 

THEN;
ELSE
CURSOR = 5;
PERFORM ARR TOTAL DATA FIELD ERROR CHECK;
EXITIF [error detected]
CURSOR = 6;
PERFORM ARR KUBBS DATA FIELD ERROR CHECK;
EXITIF [error detected]

DATA(1).ARR_PLANT = (4) ' ';
CVRST(1).ARR_PLANT = 0;
CURSOR = 7;
PERFORM ARR_OUG_DATA_FIELD_ERROR_CHECK;
EXITIF [error detected]
CURSOR = 8;
PERFORM ARR_VAINS_DATA_FIELD_ERROR_CHECK;
EXITIF [error detected]
CURSOR = 9;
PERFORM ARR_FAKMI_DATA_FIELD_ERROR_CHECK;
EXITIF [error detected]
DATA(1).ARR_MEK_A = (4) ' ';
CVRST(1).ARR_MEK_A = 0;
CURSOR = 10;
PERFORM DFG_TOTAL_DATA_FIELD_ERROR_CHECK;
EXITIF [error detected]
CURSOR = 11;
PERFORM DFG_MONTH_DATA_FIELD_ERROR_CHECK;
EXITIF [error detected]
CURSOR = 12;
PERFORM DEF_EAST_DATAFIELD_ERROR_CHECK;
EXITIF [error detected]
CURSOR = 13;
PERFORM DEF_SOUTH_DATAFIELD_ERROR_CHECK;
EXITIF [error detected]
CURSOR = 14;
PERFORM DEF_WEST_DATAFIELD_ERROR_CHECK;
EXITIF [error detected]
DMRD_DATA(1).DEF.MEL_D = (4) ' ';
DMVD_BEM(1).DEF.MEL_D = 0.;
IF DMRD_DATA(1).MSG = 'DATA ENTERED'
THEN CURSOR = 4;
END DCHECK;
PROCESS ARR_TOTAL_DATA_FIELD_ERROR_CHECK
   [This process checks for errors on total arrival demand data field]
   Get STRING (DMND_DATA(1).ARR_TOTAL) EDIT (CONVT_DENM(1).ARR_TOTAL);
   IF CONVT_DENM(1).ARR_TOTAL LT 0.
      THEN DMND_DATA(1).MSG = ERR2;
   ELSEIF VERIFY ('.', DMND_DATA(1).ARR_TOTAL) NE 0
      THEN DMND_DATA(1).MSG = ERR3;
   END ARR_TOTAL_DATA_FIELD_ERROR_CHECK;

PROCESS ARR_KURBS_DATA_FIELD_ERROR_CHECK
   [This process checks for errors on KURBS data field]
   Get STRING (DMND_DATA(1).ARR_KURBS) EDIT (CONVT_DENM(1).ARR_KURBS);
   IF CONVT_DENM(1).ARR_KURBS LT 0.
      THEN DMND_DATA(1).MSG = ERR2;
   ELSEIF VERIFY ('.', DMND_DATA(1).ARR_KURBS) NE 0
      THEN DMND_DATA(1).MSG = ERR3;
   END ARR_KURBS_DATA_FIELD_ERROR_CHECK;
PROCESS ARR_COT_DATA_FIELD_ERROR_CHECK
    (This process checks for errors on COT data field)
    Get STRING (DMND_DATA(I).ARR.COT) EDIT (CONVR_DDEM(I).ARR.COT);
    IF CONVR_DDEM(I).ARR.COT LT 0.
        THEN DMND_DATA(I).MSG = ERR2;
    ELSEIF VERIFY ("."., DMND_DATA(I).ARR.COT) EQ 0
        THEN DMND_DATA(I).MSG = ERR3;
    END ARR_COT_DATA_FIELD_ERROR_CHECK;

PROCESS ARR_VAINS_DATA_FIELD_ERROR_CHECK
    (This process checks for errors on VAINS data field)
    Get STRING (DMND_DATA(I).ARR.VAINS) EDIT (CONVR_DDEM(I).ARR.VAINS);
    IF CONVR_DDEM(I).ARR.VAINS LT 0.
        THEN DMND_DATA(I).MSG = ERR2;
    ELSEIF VERIFY ("."., DMND_DATA(I).ARR.VAINS) EQ 0
        THEN DMND_DATA(I).MSG = ERR3;
    END ARR_VAINS_DATA_FIELD_ERROR_CHECK;
PROCESS ARR_FARM_DATA_FIELD_ERROR_CHECK
    This process checks for errors on FARM data field.
    Get STRING (DMOD_DATA(I).ARR_FARM) EDIT (CONVRT_DEM(I).ARR_FARM);
    IF CONVRT_DEM(I).ARR_FARM LT 0.
        THEN DMOD_DATA(I).MSG = ERR2;
    ELSEIF VERIFY ('.', DMOD_DATA(I).ARR_FARM) EQ 0
        THEN DMOD_DATA(I).MSG = ERR3;
    END ARR_FARM_DATA_FIELD_ERROR_CHECK;

PROCESS DEP_TOTAL_DATA_FIELD_ERROR_CHECK
    This process checks for errors on total departure demand data field.
    Get STRING (DMOD_DATA(I).DEP_TOTAL) EDIT (CONVRT_DEM(I).DEP_TOTAL);
    IF CONVRT_DEM(I).DEP_TOTAL LT 0.
        THEN DMOD_DATA(I).MSG = ERR2;
    ELSEIF VERIFY ('.', DMOD_DATA(I).DEP_TOTAL) EQ 0
        THEN DMOD_DATA(I).MSG = ERR3;
    END DEP_TOTAL_DATA_FIELD_ERROR_CHECK;
PROCESS DEP_NORTH_DATA_FIELD_ERROR_CHECK
   [This process checks for errors on NORTH data field]
   Get STRING (DWIND_DATA(1).DEP.NORTH) EDIT (CONVRT_DEM(1).DEP.NORTH);
   IF CONVRT_DEM(1).DEP.NORTH LT 0.
      THEN DWIND_DATA(1).MSG = ERR2;
      ELSEIF VERLST ('.', DWIND_DATA(1).DEP.NORTH) NE 0
         THEN DWIND_DATA(1).MSG = ERR3;
   END DEP_NORTH_DATA_FIELD_ERROR_CHECK;

PROCESS DEP_EAST_DATA_FIELD_ERROR_CHECK
   [This process checks for errors on EAST data field]
   Get STRING (DWIND_DATA(1).DEP.EAST) EDIT (CONVRT_DEM(1).DEP.EAST);
   IF CONVRT_DEM(1).DEP.EAST LT 0.
      THEN DWIND_DATA(1).MSG = ERR2;
      ELSEIF VERLST ('.', DWIND_DATA(1).DEP.EAST) NE 0
         THEN DWIND_DATA(1).MSG = ERR3;
   END DEP_EAST_DATA_FIELD_ERROR_CHECK;
PROCESS DEP_SOUTH_DATA_FIELD_ERROR_CHECK
   [This process checks for errors on SOUTH data field]
   Get STRING (DOMD_DATA(I).DEP.SOUTH) EDIT (CONV1T_DEM(I).DEP.SOUTH);
   IF CONV1T_DEM(I).DEP.SOUTH LT 0.
      THEN DOMD_DATA(I).MSG = ERR2;
      ELSEIF VERIFY ('.', DOMD_DATA(I).DEP.SOUTH) EQ 0
         THEN DOMD_DATA(I).MSG = ERR3;
   END DEP_SOUTH_DATA_FIELD_ERROR_CHECK;

PROCESS DEP_WEST_DATA_FIELD_ERROR_CHECK
   [This process checks for errors on WEST data field]
   Get STRING (DOMD_DATA(I).DEP.WEST) EDIT (CONV1T_DEM(I).DEP.WEST);
   IF CONV1T_DEM(I).DEP.WEST LT 0.
      THEN DOMD_DATA(I).MSG = ERR2;
      ELSEIF VERIFY ('.', DOMD_DATA(I).DEP.WEST) EQ 0
         THEN DOMD_DATA(I).MSG = ERR3;
   END DEP_WEST_DATA_FIELD_ERROR_CHECK;
ROUTINE DVALID
IN (CNVTOAG);
INOUT (DMRD_DATA(I).RETRIEVE,CNVRT_DEM(I),CPROM,CSTO,CURSOR);

[This routine performs data validation checks on screen entries and returns value for cursor
pointing to first invalid data field. Also, an appropriate screen message is issued advising user
with corrections]

$THREE = 3;
DMSGERR = 'NUMBER OF AIRCRAFT MUST NOT EXCEED 99';
TIMEER = 'TOTAL DOES NOT EQUAL SUM OF INDIVIDUAL ENTRIES';
IF RETRIEVE ME (2) ' ';
THEN PERFORM RETRIEVE_DEM_DATA_FROM_DEM_LOG;
CURSOR = 6;
IF CNVRT_DEM(I).ARR.KURBS GT 99.0
THEN DMRD_DATA(I).MSG = DMSGERR;
ELSE DMRD_DATA(I).ARR.KURBS = SUBSTR(F(CNVRT_DEM(I).ARR.KURBS,$THREE),1,3);
CURSOR = 7;
IF CNVRT_DEM(I).ARR.COT GT 99.0
THEN DMRD_DATA(I).MSG = DMSGERR;
ELSE DMRD_DATA(I).ARR.COT = SUBSTR(F(CNVRT_DEM(I).ARR.
COT,$THREE),1,3)
CURSOR = 8;
IF CNVRT_DEM(I).ARR.VAINS GT 99.0
THEN DMRD_DATA(I).MSG = DMSGERR;
ELSE DMRD_DATA(I).ARR.VAINS = SUBSTR(F(CNVRT_DEM(I).ARR.VAINS,$THREE),1,3);
CURSOR = 9;
IF CVRT_DEN(I).ARR.FARM GT 99.0
    THEN DMND_DATA(I).MSG = DMNDERR;
ELSE
    DMND_DATA(I).ARR.FARM = SUBSTR(F(CVRT_DEN(I).ARR.
        FARM,$three),1,3);
    CURSOR = 11;
    IF CVRT_DEN(I).DEP.NORTH GT 99.0
        THEN DMND_DATA(I).MSG = DMNDERR;
    ELSE
        DMND_DATA(I).DEP.NORTH = SUBSTR(F(CVRT_DEN(I).DEP.
            NORTH,$three),1,3);
        CURSOR = 12;
        IF CVRT_DEN(I).DEP.EAST GT 99.0
            THEN DMND_DATA(I).MSG = DMNDERR;
    ELSE
        DMND_DATA(I).DEP.NORTH = SUBSTR(F(CVRT
            DEN(I).ARR.FARM,$three),1,3);
        CURSOR = 13;
        IF CVRT_DEN(I).DEP.SOUTH GT 99.0
            THEN DMND_DATA(I).MSG = DMNDERR;
        ELSE
            DMND_DATA(I).DEP.SOUTH = SUBSTR(F(CVRT
                DEN(I).DEP.SOUTH,$three),1,3);
            CURSOR = 14;
        IF CVRT_DEN(I).DEP.WEST GT 99.0
THEN  DMD_DATA(1).MSG = "DMDINHER;"
ELSE
DMD_DATA(1).DEP.WEST = SUBSTR(F(CHVRT
    DEN(1).DEP.WEST,$THREE),1,3); 
CURSOR = 5;
 IF 
FLOOR(CHVRT DEN(1).ARR.TOTAL) NE
FLOOR(CHVRT DEN(1).ARR.KUBBS +
    CHVRT DEN(1).ARR.OUT + CHVRT
DEN(1).ARR.VAINGS + CHVRT
DEN(1).ARR.FARM)
THEN  DMD_DATA(1).MSG = "TTLER;"
ELSE
DMD_DATA(1).ARR.TOTAL = 
SUBSTR(F(CHVRT
    DEN(1).ARR.TOTAL,$THREE),1,3);
CURSOR = 10;
 IF 
FLOOR(CHVRT
    DEN(1).DEP.TOTAL) NE
FLOOR(CHVRT
    DEN(1).DEP.NORTH + CHVRT
DEN(1).DEP.EAST + CHVRT
DEN(1).DEP.SOUTH + CHVRT
DEN(1).DEP.WEST)
THEN  DMD_DATA(1).MSG = "TTLER;"
ELSE
DMD_DATA(1).DEP.TOTAL = 
SUBSTR(F(CHVRT
    DEN(1).DEP.TOTAL,$THREE),1,3);
 IF DMD_DATA(1).MSG EQ "DATA ENTERED"
THEN CURSOR = 4;
END DVALID;
PROCESS RETRIEVE_DEMAND_DATA_FROM_DEMAND_LOG

[This process retrieves data from demand log]

INDEX = FLOOR (CFROM/100.0);

ALPHA = (CFROM - 100.*FLOAT(INDEX))/60.0;

NEXT = FLOOR(CTO/100.0);  
   [prorate hourly demand]

CNVRT_DEM(I).ARR.KUBSS = FLOAT(FLOOR((1.0 - ALPHA)*CNVTOG.TABLE(INDEX).KUBSS + ALPHA * CNVTOG.TABLE(NEXT).KUBSS + 0.5));

CNVRT_DEM(I).ARR.COT = FLOAT(FLOOR((1.0 - ALPHA)*CNVTOG.TABLE(INDEX).COT + ALPHA * CNVTOG.TABLE(NEXT).COT + 0.5));

CNVRT_DEM(I).ARR.VAINS = FLOAT(FLOOR((1.0 - ALPHA)*CNVTOG.TABLE(INDEX).VAINS + ALPHA * CNVTOG.TABLE(NEXT).VAINS + 0.5));

CNVRT_DEM(I).ARR.PARMN = FLOAT(FLOOR((1.0 - ALPHA)*CNVTOG.TABLE(INDEX).PARMN + ALPHA * CNVTOG.TABLE(NEXT).PARMN + 0.5));

CNVRT_DEM(I).DEF.NORTH = FLOAT(FLOOR((1.0 - ALPHA)*CNVTOG.TABLE(INDEX).NORTH + ALPHA * CNVTOG.TABLE(NEXT).NORTH + 0.5));

CNVRT_DEM(I).DEF.EAST = FLOAT(FLOOR((1.0 - ALPHA)*CNVTOG.TABLE(INDEX).EAST + ALPHA * CNVTOG.TABLE(NEXT).EAST + 0.5));

CNVRT_DEM(I).DEF.SOUTH = FLOAT(FLOOR((1.0 - ALPHA)*CNVTOG.TABLE(INDEX).SOUTH + ALPHA * CNVTOG.TABLE(NEXT).SOUTH + 0.5));

CNVRT_DEM(I).DEF.WEST = FLOAT(FLOOR((1.0 - ALPHA)*CNVTOG.TABLE(INDEX).WEST + ALPHA * CNVTOG.TABLE(NEXT).WEST + 0.5));


RETRIVE = (2) ' ';

END RETRIEVE_DEMAND_DATA_FROM_DEMAND_LOG;
2.12 Ordered List of Configurations Screen

The processing for the Ordered List of Configurations Screen is presented on pages 2-311 to 2-331.
**LOCAL VARIABLES**

INT CDATA(2) [integer variable containing index of operating configuration for both current and forecast conditions]

CHR MSG_DATA(2) [character variable of length 80 containing screen message for both current and forecast ordered list of configurations screens]

INT COUNT(2) [integer variable containing number of eligible configurations for both current and forecast conditions]

INT SWITCH(2) [this variable is used for switching between current and forecast screens, initialized to (2,1)]

STRUCTURE LIST(2)

GROUP CONF(73) [up to 73 configurations]

PLT CAPACITY [capacity of each configuration]

INT INDEX [index associated for each configuration used for table look up]

ENDSTRUCTURE;

BITS MIDIND [24 bit variable with 1 indicating runways that require coordination with MIDWAY airport]
STRUCTURE ORDER LOADLIST
[A structure of pointers, one for each data field on screen used by panel manager for loading and unloading to and from screen]

PTR TIME [pointer for environment data field]
PTR TOT_ARR [pointer for percentage of arrivals data field]
PTR NUMBER [pointer for number of configurations data field]
PTR SCROLL [pointer for scroll data field]

GROUP CONFIG(73)

PTR SELECT [pointer for configuration selection data field]
PTR RANK [pointer for rank data field]
PTR ARR(3) [pointer for arrival runways data field]
PTR DEP(4) [pointer for departure runways data field]
PTR CAPACITY [pointer for capacity data field]
PTR REMARKS [pointer for remarks data field]

PTR MSG [pointer for screen message data field]

BITS FENCE [32 bit variable as prescribed by IMS manual, initialized to string of (32) '1'B]

ENDSTRUCTURE;
ROUTINE ORDER

IN (PRCAEI, INFORM, CNFRG, RNYG, MIDFLAG);

INOUT (CONFILST, CONFIND, RSTATUS, I);
   [This routine invokes ordered list of configurations screen for both current and forecast environments]

CALL OSETUP;

IN (PRCAEI, INFORM, CNFRG, RNYG, MIDFLAG)

INOUT (CONFILST, LIST, COUNT);
   [This routine sets up information on ordered list of configurations screen]

CDATA = CONFIND;
MSG_DATA(1) = CONFILST(1).MSG;
MSG_DATA(2) = CONFILST(2).MSG;

REPEAT UNTIL (RSTATUS NE PP12);
   CDATA(I) = CONFIND(I);
   CONFILST(I).MSG = MSG_DATA(1);
   I = SWICTH(I); [switch between two screens]

REPEAT UNTIL (RSTATUS NE PP6);
   CALL OSCREEN;
IN (CONFLIST(1), LIST(1), COUNT(1));
INOUT (CDATA(1), RSTATUS);

[This routine controls ordered list of configurations across]
ENDREPEAT;
ENDREPEAT;
LOOP: [J = 1 To 2]
IF SUBSTR(CONFLIST(J), MSG, 1, 12) EQ 'DATA ENTERED'
THEN CONFINFO(J) = CDATA(J);
ROUTINE OS SETUP

IN (PRCARR, INFORM, CNFORQ, MWYEQP, MIDFLAG, COUNT);

INOUT (CONFLST, LIST, COUNT);  [This routine sets up information on ordered list of configurations screen]

LIST = INFORM, BY NAME;

LOOP;  [K = 1 to 2]  [compute percentage of arrivals]

IF PRCARR(K).TOTAR + PRCARR(K).TOTDEP = 0.0

THEN ATUPREC = .5;  [default value of arrival percentage is .5 if no demand is specified]

ELSE ATUPREC = PRCARR(K).TOTAR/(PRCARR(K).TOTARR + PRCARR(K).TOTDEP);

CONFLST(K).TOTARR = SUBSTR(F(ATUPREC *100.0, @THREE),1,3);

CALL OSORT;

INOUT (LIST(K));  [This routine sorts list of configurations based on capacity]

COUNT(K) = 0;

LOOP;  [N = 1 to 73]

P = LIST(K).CONF(N).INDEX;

IF P LT 999

THEN
PERFORM SCREEN_PARAMETERS_SET_UP;
   [set up parameters on screen]

PERFORM FLAG_SETTING;
   [set up appropriate flags]
ENDLOOP;

CONFLICT(K).NUMBER = SUBSTR(P(FLOAT(COUNT(K)),"THREE"),1,3));
ENDLOOP;
END OSETUP;
PROCESS SCREEN_PARAMETERS_SET_UP
[This process sets up parameters on ordered list of configurations screen]

COUNT(K) = COUNT(K) + 1;  [Increment configuration counter]

CONFLST(K).CONFIG(COUNT(K)).ARR(1) = CNFGCQ(P).ARR_ENVY(1);
CONFLST(K).CONFIG(COUNT(K)).ARR(2) = CNFGCQ(P).ARR_ENVY(2);
CONFLST(K).CONFIG(COUNT(K)).ARR(3) = CNFGCQ(P).ARR_ENVY(3);
CONFLST(K).CONFIG(COUNT(K)).ARR(4) = CNFGCQ(P).ARR_ENVY(4);

CONFLST(K).CONFIG(COUNT(K)).DEP(1) = CNFGCQ(P).DEP_ENVY(1);
CONFLST(K).CONFIG(COUNT(K)).DEP(2) = CNFGCQ(P).DEP_ENVY(2);
CONFLST(K).CONFIG(COUNT(K)).DEP(3) = CNFGCQ(P).DEP_ENVY(3);
CONFLST(K).CONFIG(COUNT(K)).DEP(4) = CNFGCQ(P).DEP_ENVY(4);

CONFLST(K).CONFIG(COUNT(K)).CAPACITY = SUBSTR(P'LIST(K).CONF(K).CAPACITY, $FOUR), 1, 4);

END SCREEN_PARAMETERS_SET_UP;
PROCESS FLAG SETTING
[This process determines warning flags for ordered list of configurations screen]

FLAG = '0'B;
HIRLIND = 'B';

LOOP: [1: 1 to 12] [determine runways on which HIRL is out]

IF RWYEQP(K).RUNWAY(L).HIRL NE (2) 'B'
THEN HIRLIND = HIRLIND CONCATENATE '1' B;
ELSE HIRLIND = HIRLIND CONCATENATE '0' B;

ENDLOOP;

HIRLIND = HIRLIND CONCATENATE HIRLIND;
MESSAGE = (27) ' '; IF PRCARR(K).CONF(P).BMPRCH AT 0. [if airport is saturated]
THEN MESSAGE = 'SATURATED';
FLAG = '1'B;
IF MIDLFLAG(K) NE (2) 'B' AND ((CNFGQ(P).ID AND HIRLIND) GT 0)
THEN IF FLAG EQ '1'B
THEN MESSAGE = MESSAGE CONCATENATE 'MIDWAY';
ELSE MESSAGE = 'MIDWAY';
FLAG = '1'B;
IF (CNFGQ(P).ID AND HIRLIND) NE (24) '0'B
THEN IF FLAG EQ '1'B
THEN MESSAGE = MESSAGE CONCATENATE 'DAY ONLY';
ELSE MESSAGE = 'DAY ONLY';
ELSE MESSAGE = 'DAY ONLY';

END FLAG SETTING;
ROUTINE OSORT

INPUT (LIST(K));
[This routine sorts configurations list on capacity; Shell's method is used]

M(1) = 36;
M(2) = 18;
M(3) = 9;
M(4) = 5;
M(5) = 3;
M(6) = 1;

LOOP: \( M = 1 \) to 6

LOOP: \( J = (M+1) \) to 73

1 = J - H(M);

IF LIST(K).CONF(J).CAPACITY GT LIST(K).CONF(1).CAPACITY

THEN

TEMP1 = LIST(K).CONF(1).CAPACITY;
TEMP2 = LIST(K).CONF(1).INDEX;
LIST(K).CONF(1).CAPACITY = LIST(K).CONF(1+H(M)).CAPACITY;
LIST(K).CONF(1).INDEX = LIST(K).CONF(1+H(M)).INDEX;
LIST(K).CONF(1+H(M)).CAPACITY = TEMP1;
LIST(K).CONF(1+H(M)).INDEX = TEMP2;
I = I - H(M);

IF I GT 0

THEN
REPEAT WHILE ((I GT 0) AND (LIST(K).COMP(I+H(N)).CAPACITY GT LIST(K).COMP(I).CAPACITY));

TEMP1 = LIST(K).COMP(I).CAPACITY;
TEMP2 = LIST(K).COMP(I).INDEX;
LIST(K).COMP(I).CAPACITY = LIST(K).COMP(I+H(N)).CAPACITY;
LIST(K).COMP(I).INDEX = LIST(K).COMP(I+H(N)).INDEX;
LIST(K).COMP(I+H(N)).CAPACITY = TEMP1;
LIST(K).COMP(I+H(N)).INDEX = TEMP2;
I = I - H(N);
ENDREPEAT;

END LOOP

END OSORT;
ROUTINE OSCREEN

IN (COMGLST(I), LIST(I), COUNT(I));

INOUT (CDATA(I), ESTATUS);  
(This routine controls ordered list of configurations screen)

CHR PNNAME [character variable of length 8 containing name of DNS panel initialized to 'OLIST', name of panel that controls ordered list of configurations screen]

INT CURSOR [integer variable containing cursor's position on screen]

BITS DM(115) [8 bit variable of data masks used in DNS]

INT CNVT_SCROLL [integer value of scroll data field]

STRUCTURE BLANK [This structure is used in conjunction with scrolling function it contains data fields similar to COMGLST structure that are blank]

CHR SELECT [length 1]

CHR RANK [length 2]

CHR ARR(3) [length 3]

CHR DEF(4) [length 3]

CHR CAPACITY [length 5]

CHR REMARKS [length 27]

ENDSTRUCTURE;

PERFORM SET_UP_SCREEN_PERMANENT_POINTERS(Order);

PERFORM SCREEN_PROGRAM_INITIALIZATION;

REPEAT UNTIL (ESTATUS NE ENTER);

PERFORM SCREEN_SCROLL;
PERFORM DISPLAY_PANEL;
IF RSTATUS EQ PA1
  THEN STOP;
IF RSTATUS NE ENTER
  THEN;
ELSE
  DM = FLDDEF;
  DM(1) = FLMHIGH;
  DM(115) = FLMHIGH;
  CALL OCHECK;
  IN (CONV_SCROLL, L, N, COUNT(I));
  INPUT (CONFLIST(I), CURSOR);
  [This routine checks for errors occurred on screen as a result of an erroneous
   entry and returns value for cursor pointing to first data field where an error
   has occurred; and an appropriate screen message is issued advising user with
   corrections]
  IF CONFLIST(I).MSG NE 'DATA ENTERED'
    THEN
      DM(CURSOR) = FLMHIGH;
      CONV_SCROLL = 0;
    ELSE
      CALL OVALID;
      IN (L, N, COUNT(I));
INOUT (CONFLST(I), CURSOR);

This routine performs data validation checks on screen entries and returns value for cursor pointing to first invalid data field. Also, an appropriate screen message is issued advising user with corrections.

IF CONFLST(I).MSG NE 'DATA ENTERED'
THEN

SM(CURSOR) = FLDMIGH;
CNT_SCROLL = 0;
CALL OVALID;

IN (LIST(I), L, N, CONFLST(I));
INOUT (CDATA(I));

This routine locally updates configuration index parameter.

CONFLST(I).MSG = 'DATA ENTERED AT 'CONCATENATE GM;
CONFLST(I).SCROLL = (4) ' ';

LOOP: [P = 1 to 73]

CONFLST(I).CONF(P).SELECT = ' ';

ENDLOOP;

ENDREPEAT

END SCREEN;
PROCESS SET_UP_SCREEN_PERMANENT_POINTERS (ORDER)
[This process sets up screen pointers for permanent variables for DNS use]
ORDER_LOADLIST.TIME = ADDR(CONFLST(I).TIME);
ORDER_LOADLIST.UTER = ADDR(CONFLST(I).UTER);
ORDER_LOADLIST.NUMBER = ADDR(CONFLST(I).NUMBER);
ORDER_LOADLIST.SCROLL = ADDR(CONFLST(I).SCROLL);
ORDER_LOADLIST.MSG = ADDR(CONFLST(I).MSG);
END SET_UP_SCREEN_PERMANENT_SCREEN_ORDER;

PROCESS SCREEN_PROGRAM_INITIALIZATION
[This process performs a number of variable initializations for screen routine]
IF COUNT(I) EQ 0 [if no configuration is eligible generate message]
THEN CONFLST(I).MSG = SUBSTR(CONFLST(I).MSG, 1, 20) CONCATENATE ' *** NO ELIGIBLE CONFIGURATION S ***';
DELTA = MIN(10, COUNT(I)); [up to 10 configurations appear on screen at a time]
NEXT = DELTA; [set up scrolling function parameters]
INDEX = 1;
CONF_SCROLL = 0;
DM = FLDDEF;
DM(115) = FLDHIGH;
[set up other parameters]
CONFLST(I).SCROLL = (4) '1';
LOOP; [P = 1 to 73]
CONFLST(I).CONF(P).SELECT = '1';
ENDLOOP;
CURSOR = 4;
END SCREEN_PROGRAM_INITIALIZATION;
PROCESS SCREEN_SCROLL
[This process performs scrolling function for ordered list of configurations screen]
INDEX = INDEX + CNVT_SCROLL;
NEXT = NEXT + CNVT_SCROLL;
IF NEXT LT DELTA THEN
    L = 1
    M = MAX(1, NEXT);
    NEXT = M;
    INDEX = M - DELTA + 1;
ELSEIF INDEX GT COUNT(I) - DELTA + 1 THEN
    M = COUNT(I);
    L = MIN(COUNT(I), INDEX);
    NEXT = L + DELTA - 1;
ELSE
    M = NEXT;
    L = INDEX;
K = 0;
IF (M LT 10) AND COUNT GT 9 THEN
    K = X + 1;
    DM(1) = 0;
ORDER_LOADLIST.CONF(K).RANK = ADDR(0, RANK);
ORDER_LOADLIST.CONF(K).SELECT = ADDR(0, SELECT);
ORDER_LOADLIST.CONF(K).ARR(1) = ADDR(0, ARR(1));
ORDER_LOADLIST.CONF(K).ARR(2) = ADDR(0, ARR(2));
ORDER_LOADLIST_CONF(K).ARR(3)  = ADDR(BLANK.ARR(3));
ORDER_LOADLIST_CONF(K).DEP(1)  = ADDR(BLANK.DEP(1));
ORDER_LOADLIST_CONF(K).DEP(2)  = ADDR(BLANK.DEP(2));
ORDER_LOADLIST_CONF(K).DEP(3)  = ADDR(BLANK.DEP(3));
ORDER_LOADLIST_CONF(K).DEP(4)  = ADDR(BLANK.DEP(4));
ORDER_LOADLIST_CONF(K).CAPACITY = ADDR(BLANK.CAPACITY);
ORDER_LOADLIST_CONF(K).REMARKS  = ADDR(BLANK.REMARKS);

ENDREPEAT;

REPEAT WHILE (COUNT(I) GT 0); [J = L to N]
  K = K + 1;
  IF CDATA(I) EQ LIST(I).CONF(J).INDEX
  THEN
    LOOP: [P = 1 to 11]
      DM(LINK + P - 1) = FLMHIGH; [highlight operating configuration]
    ENDL0OP;
  ORDER_LOADLIST_CONF(K).RANK  = ADDR(COMPLST(I).CONF(J).RANK);
ORDER_LOADLIST_CONF(K).SELECT = ADDR(COMPLST(I).CONF(J).SELECT);
ORDER_LOADLIST_CONF(K).ASS(1) = ADDR(COMPLST(I).CONF(J).ASS(1));
ORDER_LOADLIST_CONF(K).ASS(2) = ADDR(COMPLST(I).CONF(J).ASS(2));
ORDER_LOADLIST_CONF(K).ASS(3) = ADDR(COMPLST(I).CONF(J).ASS(3));
ORDER_LOADLIST_CONF(K).DEP(1) = ADDR(COMPLST(I).CONF(J).DEP(1));
ORDER_LOADLIST_CONF(K).DEP(2) = ADDR(COMPLST(I).CONF(J).DEP(2));
ORDER_LOADLIST_CONF(K).DEP(3) = ADDR(COMPLST(I).CONF(J).DEP(3));
ORDER_LOADLIST_CONF(K).DEP(4) = ADDR(COMPLST(I).CONF(J).DEP(4));
ORDER_LOADLIST_CONF(K).CAPACITY = ADDR(COMPLST(I).CONF(J).CAPACITY);
ORDER_LOADLIST_CONF(K).REMARKS = ADDR(COMPLST(I).CONF(J).REMARKS);

ENDREPEAT;
REPEAT WHILE (K LT 10);

K = K + 1;

DN(11*K - 6) = FIDMARK;

ORDER_LOADLIST_CONF(K).RANK = ADDR(BLANK.RANK);
ORDER_LOADLIST_CONF(K).SELECT = ADDR(BLANK.SELECT);
ORDER_LOADLIST_CONF(K).ARR(1) = ADDR(BLANK.ARR(1));
ORDER_LOADLIST_CONF(K).ARR(2) = ADDR(BLANK.ARR(2));
ORDER_LOADLIST_CONF(K).ARR(3) = ADDR(BLANK.ARR(3));
ORDER_LOADLIST_CONF(K).ARR(4) = ADDR(BLANK.ARR(4));
ORDER_LOADLIST_CONF(K).DEP(1) = ADDR(BLANK.DEP(1));
ORDER_LOADLIST_CONF(K).DEP(2) = ADDR(BLANK.DEP(2));
ORDER_LOADLIST_CONF(K).DEP(3) = ADDR(BLANK.DEP(3));
ORDER_LOADLIST_CONF(K).DEP(4) = ADDR(BLANK.DEP(4));
ORDER_LOADLIST_CONF(K).CAPACITY = ADDR(BLANK.CAPACITY);
ORDER_LOADLIST_CONF(K).REMARKS = ADDR(BLANK.REMARKS);

ENDREPEAT;

END SCREEN_SCROLL;
ROUTINE OCHECK

IN (CNVT_SCROLL, L, M, COUNT(M));

INOUT (CONFLST(I), CURSOR);

This routine checks for errors occurred on screen as a result of an erroneous entry and returns
value for cursor pointing to first data field where an error has occurred; and an appropriate
screen message is issued advising user with corrections.

X OR BLANK = 'X'
ERR1 = 'NUMERIC INPUT REQUIRED';
ERR3 = 'NO DECIMAL POINTS ALLOWED';
ERR5 = 'INPUT MUST BE X OR BLANK';

ON CONVERSION BEGIN;

CONFLST(I).MSG = ERR1;
RETURN;

CONFLST(I).MSG = 'DATA ENTERED';
CURSOR = 4;

Get STRING (CONFLST(I).SCROLL) EDIT (CNVT_SCROLL);

IF VERIFY (*., CONFLST(I).SCROLL) EQ 0

THEN CONFLST(I).MSG = ERR3;
ELSEIF (L EQ 1) AND (COUNT OR 10)

THEN K = 10 - M;
ELSE K = 0;

REPEAT WHILE (CONFLST(I).MSG EQ 'DATA ENTERED'); [J = L to M]

CURSOR = 11*K + 5;
K = K + 1;
IF VERI(F(CONPLST(I)).CONF(I).SELECT,X OR BLANK)NE 0
THEN CONPLST(I).MSG = ERR3;
ENDREPEAT;

IF CONPLST(I).MSG EQ 'DATA ENTERED'
THEN CURSOR = 4;
END OCHECK;
ROUTINE OVALID

IN  (L, M, COUNT(M));

INOUT (CONFLIST(I), CURSOR);
[This routine performs data validation checks on screen entries and returns value for cursor
pointing to first invalid data field. Also, an appropriate screen message is issued advising user
with corrections]

ERROR = 'SELECT ONLY ONE CONFIGURATION';

CONFIST(I).MSG = 'DATA ENTERED';

FLAG = 0;

IF  (L NOT 1) AND  (COUNT IS 10)
    THEN  K = 10 - M;
    ELSE  K = 0;

REPEAT WHILE (FLAG LT 2);  [J = L TO M]

    CURSOR = 11*K + 5;
    K = K + 1;
    IF  (CONFIST(I).CONF(J).SELECT NB ' ')
        THEN  FLAG = FLAG + 1;

ENDREPEAT

IF  FLAG LT 2
    THEN  CURSOR = 4;
    ELSE  CONFIST(I).MSG = ERROR;

END OVALID;
ROUTINE UPDATE

IN (LIST(I), L, M, COMFLIST(I));
INO (CDATA(I));

[This routine updates configuration index parameter locally]

FLAG = '0'8; 

REPEAT WHILE (FLAG = '0'8); [J = L to M]

IF COMFLIST(I).CONF(J).SELECT NE ' ' THEN

CDATA(I) = LIST(I).CONF(J).INDEX;
COMFLIST(I).CONF(J).SELECT = ' '; 
FLAG = '1'B;

ENDREPEAT;

END UPDATE;
2.13 Departure Queue Screen

The processing for the Departure Queue Screen is presented on pages 2-333 to 2-339.
**LOCAL VARIABLES**

STRUCTURE QUELEN DATA LIKE QUELEN;
   [This structure is similar to QUELEN used as a working area within screen routine]
ENDSTRUCTURE;

INT COUNT_QUELEN(4) [This variable is similar to COUNTQM used as a working area within screen routine]

STRUCTURE QUE_JOBBLIST [A structure of pointers, one for each data field on screen used by panel manager for loading and unloading to and from screen]
   GROUP LINK(4)
      PTR DFMUN [pointer for departure runway data field]
      PTR QL [pointer for queue length data field]
   RITE FENCE [32 bit variable as prescribed by EMS manual, initialized to string of (32) '1'B]
ENDSTRUCTURE;
ROUTINE QUEUE

INPUT (QUELEN, CNVQLN, RSTATUS);
    [This routine invokes current departure queue screen]
REPEAT UNTIL (RSTATUS NE FF12);
    QUELEN_DATA = QUELEN;
    CNVRLT_QUELEN = CNVQLN;
    REPEAT UNTIL (RSTATUS NE FF7);
        CALL QSCREEN;
        INPUT (QUELEN_DATA, CNVRLT_QUELEN, RSTATUS);
            [This routine controls current departure queue screen]
    ENDFREPEAT;
ENDREPEAT;

IF SUBSTR(QUELEN_DATA, MSG, 1, 12) EQ 'DATA ENTERED'
    THEN
        QUELEN = QUELEN_DATA;
        CNVQLN = CNVRLT_QUELEN;
END QUEUE;
ROUTINE QSCREEN

**INPUT** (QUELEN_DATA, CURLT_QUELEN, RSTATUS);

[This routine controls current departure queue screen]

**CUR_** FNAME [character variable of length 8 containing name of DNS panel initialized to 'QUELEN',

name of panel that controls current departure queue]

**INT** CURSOR [integer variable containing cursor's position on screen]

**BITS** DM(9) [8 bit variable of data mask used in DNS]

**STRUCTURE** AUX_DATA LIKE QUELEN_DATA

ENDSTRUCTURE;

CURSOR = 2;

AUX_DATA = QUELEN_DATA;

DM = FLDDW; [set data fields to default intensity (normal)]

DM(9) = FLDMAX; [set last data field to high intensity]

PERFORM SET_UP_SCREEN_POINTERS(Queue);

DEPCOUNT = 0;

LOOP; [J = 1 to 4]

IF QUELEN_DATA.DEPRUN(J) NE 3 " [check number of departure runways in current configuration]

THEN

DM(2*J) = FLDDF;

DEPCOUNT = DEPCOUNT + 1;

ELSE DM(2*J) = FLDMAX; [darken screen if there are no departure runways]

ENDLOOP;

REPEAT UNTIL (RSTATUS NE ENTER);

PERFORM DISPLAY_PANEL;
IF RSTATUS EQ 'PAL'
  THEN stop;
IF RSTATUS NE 'ENTER'
  THEN QUELEM_DATA = AUX_DATA;
LOOP: [J = 1 to 2 * DEPCOUNT]
  DM(J) = FLOORE;
ENDLOOP;
CALL QCHECK;
IN (DEPCOUNT);
INPUT (QUELEM_DATA, CURRT_QUELEM, CURSOR);
  [This routine checks for errors occurred on screen as a result of an erroneous entry and returns value for cursor pointing to first data field where an error has occurred; and an appropriate screen message is issued advising user with corrections]
IF QUELEM_DATA.MSG NE 'DATA ENTERED'
  THEN DM(CURSOR) = FLDHIGH;
END;
CALL QVALID;
IN (DEPCOUNT);
INPUT (QUELEM_DATA, CURRT_QUELEM)
  [This routine right-justifies data on screen]
QUELEM_DATA.MSG = 'DATA ENTERED AT ' CONCATENATE DM;
AUX_DATA = QUELEM_DATA;
ENDREPEAT;
END QSCREEN;
PROCESS SET_UP_SCREEN_POINTERS(Queue)
   [This process sets up screen pointers for MSG use]
   LOOP; (J = 1 to 4)
      QUE_LOADLIST.LINE(J).DEPRUN = ADDR(QUELEN_DATA.DEPRUN(J));
      QUE_LOADLIST.LINE(J).QL = ADDR(QUELEN_DATA.QL(J));
   ENDLOOP;
   QUE_LOADLIST.MSG = ADDR(QUELEN_DATA.MSG);
   END SET_UP_SCREEN_POINTERS(Queue);
ROUTINE QCHECK

IN (DEPCOUNT);

INPUT (QUELEN_DATA, CVRT_QULEN, CURSOR);

(This routine checks for errors occurred on screen as a result of an erroneous entry and returns value for cursor pointing to first data field where an error has occurred; and an appropriate screen message is issued advising user with corrections)

ERR1 = 'NUMERIC INPUT REQUIRED';
ERR2 = 'NON-NEGATIVE INPUT REQUIRED';
ERR3 = 'NO DECIMAL POINTS ALLOWED';
QUELEN_DATA.MSG = 'DATA ENTERED';

ON CONVERSION BEGIN;

QUELEN_DATA.MSG = ERR1;

RETURN;

REPEAT WHILE (QUELEN_DATA.MSG EQ 'DATA ENTERED') [J = 1 to DEPCOUNT]

CURSOR = 2*J;

Get STRING (QUELEN_DATA.QL(J)) EDIT (CVRT_QULEN(J));

IF VERIFY ('.', QUELEN_DATA.QL(J)) EQ 0

THEN QUELEN_DATA.MSG = ERR2;

ELSEIF VERIFY('.', QUELEN_DATA.QL(J)) EQ 0

THEN QUELEN_DATA.MSG = ERR3;

ENDREPEAT;

IF QUELEN_DATA.MSG = 'DATA ENTERED'

THEN CURSOR = 2;

END QCHECK;
ROUTINE QVALID

IN (DEPCOUNT);

INOUT (QUELEN_DATA, CONVT_QULEN);

[This routine right-justifies on data on screen]

$TWO = 2;

LOOP: (J = 1 TO DEPCOUNT)

QUELEM_DATA.QJ(J) = SUBSTR(FLOAT(CONVT_QULEN(J)), $TWO), 1, 2);

ENDLOOP;

END QVALID;
2.14 Ordered List of Transitions Screen

Pages 2-341 to 2-396 present the processing of the Ordered List of Transitions Screen.
STRUCTURE TRANS_LOADLIST
[A structure of pointers, one for each data field on screen used by panel manager for loading and unloading to and from screen]

PTR PCT_ARR [pointer for arrival percentage data field]
PTR NUM_ELIG [pointer for number of eligible configuration data field]
PTR SCROLL [pointer for scroll data field]
PTR ARR(3) [pointer for arrival runways data field]
PTR DEP(4) [pointer for departure runways data field]
PTR CMRANHR [pointer for current configuration's transition hour capacity data field]
PTR CFINCAP [pointer for current configuration's final capacity data field]
GROUP CONFIG(IO)
    PTR RANK [pointer for rank data field]
    PTR ARR(3) [pointer for arrival runways data field]
    PTR DEP(4) [pointer for departure runways data field]
    PTR MINUTES [pointer for transition duration data field]
    PTR HOURLY [pointer for hourly transition capacity data field]
    PTR FINCAP [pointer for final capacity data field]
PTR MSG [pointer for screen message data field]

BITE FENCE [32 bit variable as prescribed by DNS manual, initialized to string of (32) '1' B]
ENDSTRUCTURE;
INT CONFIGDATA (2, 7) [an integer variable containing numerical codes for arrival and departure runways in current and final configurations involved in transition]

ROUTINE TSETUP

IN (PREARR, CNVIRCQ, CNVIRCHN, DEPMAT, FIXTRAV, TRANLIST, INFORM, CONFIND, CNVTHE, ELGRLY, CNVTOLN, QUELEN);

INPUT (RSTATUS);
[This routine invokes ordered list of transitions screen]

CALL TRAN;

IN (PREARR, CNVIRCQ, CNVIRCHN, DEPMAT, FIXTRAV, INFORM, CONFIND, CNVTHE, ELGRLY, CNVTOLN, QUELEN);

INPUT (TRANLIST)
[This routine performs transition computations and transition screen parameter set up]

TEMP1 = TRANLIST.NSC;
TEMP2 = (4) ' ';

REPEAT UNTIL (RSTATUS NE FP12);

TRANLIST.NSC = TEMP1;
TRANLIST.SCROLL = TEMP2;

REPEAT UNTIL (RSTATUS NE FP9);

CALL YCRENIN;

IN (ELGRLY(3));

INPUT (TRANLIST, RSTATUS);
[This routine controls ordered list of transitions screen]

ENDEPEAT;

ENDEPEAT;

ENDEPEAT;

END TSETUP;
ROUTINE TSCREEN

IN (ELGBLT(2));

INOUT (TRANLST, ESTATUS);

[This routine controls ordered list of transitions screen]

CHR PHNAME [Character variable of length 8 containing name of DMS panel initialized to 'TRANLIST', name of panel that controls ordered list of transitions screen]

INT CURSOR [Integer variable containing cursor's position on screen]

BITS DM(123) [8 bit variable of data mask used in DMS]

INT CNVT_SCROLL [Integer value of scroll data field]

PERFORM SET_UP_SCREEN_PERMANT_POINTERS (TSETUP);

PERFORM SCREEN_PROGRAM_INITIALIZATION;

IF KEGRLTY(2).NUM EQ 0

THEN TRANLST.MSG = 'NO ELIGIBLE CONFIGURATIONS';

IF TRANLST.MSG EQ 'CURRENT CONFIGURATION IS UNELIGIBLE'

THEN

COUNT = 0;
DELTA = 0;
NEXT = 0;

REPEAT UNTIL (ESTATUS NE ENTER);

PERFORM SCREEN_SCROLL;

PERFORM DISPLAY_PANEL;

IF ESTATUS EQ FAIL

THEN stop;
IF RSTATUS EQ ENTER
THEN
   DM = FLDOEP;
   DM(123) = FLDOHIGH;

   LOOP: {J = 4 to 12}
   DM(J) = FLDOHIGH;
   ENDDO;
   CALL TCHECK;

   INOUT (TRANLIST, CNVT_SCROLL);
   [This routine checks for errors occurred on screen as a result of an erroneous entry
    and returns value for cursor pointing to first data field where an error has
    occurred; and an appropriate screen message is issued advising user with corrections]

IF TRANLIST.NSC NE 'DATA ENTERED'
THEN
   DM(CURSOR) = FLDOHIGH;
   CNVT_SCROLL = 0;
ELSE
   TRANLIST_SCROLL = (4) ' ';
   TRANLIST.NSC = 'DATA ENTERED AT ' CONCATENATE CNVT;
ENDDO;
END SCREEN;
PROCESS SET_UP_SCREEN_PERMANENT PointERS(TSETUP)

(This process sets up screen pointers of permanent variables for DMS use)

TRANS_LOADLIST.ARR(1) = ADDR(TRANLIST.ARR(1));
TRANS_LOADLIST.ARR(2) = ADDR(TRANLIST.ARR(2));
TRANS_LOADLIST.ARR(3) = ADDR(TRANLIST.ARR(3));
TRANS_LOADLIST.DEF(1) = ADDR(TRANLIST.DEF(1));
TRANS_LOADLIST.DEF(2) = ADDR(TRANLIST.DEF(2));
TRANS_LOADLIST.DEF(3) = ADDR(TRANLIST.DEF(3));
TRANS_LOADLIST.DEF(4) = ADDR(TRANLIST.DEF(4));
TRANS_LOADLIST.PCT_ARR = ADDR(TRANLIST.PCT_ARR);
TRANS_LOADLIST.MINUTES = ADDR(TRANLIST.MINUTES);
TRANS_LOADLIST.MSG = ADDR(TRANLIST.MSG);

LOOP: [J = 1 to 10]

TRANS_LOADLIST.CONFIG(J).BANK = ADDR(TRANLIST.CONFIG(J).BANK);
TRANS_LOADLIST.CONFIG(J).ARR(1) = ADDR(TRANLIST.CONFIG(J).ARR(1));
TRANS_LOADLIST.CONFIG(J).ARR(2) = ADDR(TRANLIST.CONFIG(J).ARR(2));
TRANS_LOADLIST.CONFIG(J).ARR(3) = ADDR(TRANLIST.CONFIG(J).ARR(3));
TRANS_LOADLIST.CONFIG(J).DEF(1) = ADDR(TRANLIST.CONFIG(J).DEF(1));
TRANS_LOADLIST.CONFIG(J).DEF(2) = ADDR(TRANLIST.CONFIG(J).DEF(2));
TRANS_LOADLIST.CONFIG(J).DEF(3) = ADDR(TRANLIST.CONFIG(J).DEF(3));
TRANS_LOADLIST.CONFIG(J).DEF(4) = ADDR(TRANLIST.CONFIG(J).DEF(4));
TRANS_LOADLIST.CONFIG(J).HOURLY = ADDR(TRANLIST.CONFIG(J).HOURLY);
TRANS_LOADLIST.CONFIG(J).PINCAP = ADDR(TRANLIST.CONFIG(J).PINCAP);
TRANS_LOADLIST.CONFIG(J).FINCAP = ADDR(TRANLIST.CONFIG(J).FINCAP);
TRANS_LOADLIST.CONFIG(J).FINCAP = ADDR(TRANLIST.CONFIG(J).FINCAP);

END LOOP:

END SET_UP_SCREEN_PERMANENT_POINTERS(TSETUP);
PROCESS SCREEN PROGRAM INITIALIZATION

[This process performs a number of variable initializations for screen routine]

CURSOR = 3;
DM = FLDDEF;
DM(123) = FLDHIGH;

LOOP: [j = 4 to 12]
| DM(j) = FLDHIGH;
ENDLOOP;

CHVT SCROLL = 0;
TRANSLF.MSG = (4) \n';
COUNT = ELGBLT.NUM;
DELTA = MIN(COUNT, 10);
NEXT = DELTA;
INDEX = 1;

END SCREEN_PROGRAM_INITIALIZATION;
PROCESS SCREEN_SCROLL
[This process performs scrolling function for ordered list of transitions screen]

INDEX = INDEX + CNVT_SCROLL;
NEXT = NEXT + CNVT_SCROLL;

IF NEXT LT DELTA

THEN
L = 1;
M = MAX(1, NEXT);
NEXT = M;
INDEX = M - DELTA + 1;

ELSE IF INDEX GT COUNT - DELTA + 1;

THEN
M = COUNT;
L = MIN(COUNT, INDEX);
INDEX = L;
NEXT = L + DELTA - 1;

ELSE
M = NEXT;
L = INDEX;

K = 0;

IP (N LT 10) AND (COUNT GT 9)

THEN
REPEAT WHILE (K LT 10 - N)
LOOP; [J = 1 to 11]
DM(11*K + J + 12) = FLDABX;
ENDLOOP;
K = K + 1;
ENDREPEAT:

REPEAT WHILE \((N \geq 0)\); \([J = L \text{ to } N]\)

\(K = K + 1;\)

TRANS_LOADLIST.CONFIG(K).RANK = ADDR(TRANSLIST.CONFIG(J).RANK);
TRANS_LOADLIST.CONFIG(K).ARR(1) = ADDR(TRANSLIST.CONFIG(J).ARR(1));
TRANS_LOADLIST.CONFIG(K).ARR(2) = ADDR(TRANSLIST.CONFIG(J).ARR(2));
TRANS_LOADLIST.CONFIG(K).ARR(3) = ADDR(TRANSLIST.CONFIG(J).ARR(3));
TRANS_LOADLIST.CONFIG(K).DEP(1) = ADDR(TRANSLIST.CONFIG(J).DEP(1));
TRANS_LOADLIST.CONFIG(K).DEP(2) = ADDR(TRANSLIST.CONFIG(J).DEP(2));
TRANS_LOADLIST.CONFIG(K).DEP(3) = ADDR(TRANSLIST.CONFIG(J).DEP(3));
TRANS_LOADLIST.CONFIG(K).DEP(4) = ADDR(TRANSLIST.CONFIG(J).DEP(4));
TRANS_LOADLIST.CONFIG(K).HOURS = ADDR(TRANSLIST.CONFIG(J).HOURS);
TRANS_LOADLIST.CONFIG(K).FIRCAP = ADDR(TRANSLIST.CONFIG(J).FIRCAP);
TRANS_LOADLIST.CONFIG(K).MINUTES = ADDR(TRANSLIST.CONFIG(J).MINUTES);

ENDLOOP:

REPEAT WHILE \((K \ LT 10)\);

LOOP; \([J = 1 \text{ to } 11]\)

DM(11*K + J + 12) = FIELD;

ENDLOOP;

K = K + 1;

ENDREPEAT:

END SCREEN_SCROLL;
ROUTINE TCHECK

INPUT (TRANSLST, CVNT.SCROLL);

[This routine checks for errors occurred on screen as a result of an erroneous entry and returns
value for cursor pointing to first data field where an error has occurred; and appropriate screen
message is issued advising user with corrections]

ERR1 = 'NUMERIC INPUT REQUIRED';
ERR3 = 'NO DECIMAL POINTS ALLOWED';

ON CONVERSION BEGIN;

TRANSLST.MSG = ERR1;

RETURN;

TRANSLST.MSG = 'DATA ENTERED';

GET STRING (TRANSLST.SCROLL) EDIT (CVNT.SCROLL);

IF VERIFY ('.', TRANSLST.SCROLL) EQ 0

THEN TRANSLST.MSG = ERR3;

END TCHECK;
ROUTINE TRAN

VAR (PBCARR, CHCFGQ, CHYTNQM, DEPMAK, FIXTRAN, INFORM, CONFIND, CHOTH, ELIGBLYT, CHYTNLH, QUELEN);

INPUT (TRANSLT);
[This routine performs transition computations and screen parameter set up]

Y = 'O'S;
REM = 0;
$TMTH = 2;
$TRHSE = 3;
$FOUR = 4;

IF (CHOTH(1) EQ 2) AND (CHOTH(2) EQ 1)
[set up variable X based on weather conditions before and after transition]
THEN X = 2;
ELSE X = 1;
[compute forecast percentage of arrivals]

IF PBCARR(2).TOTARK + PBCARR(2).TODDF EQ 0.
THEN ATOYPEC = 0.5;
ELSE ATOYPEC = PBCARR(2).TOTARK/(PBCARR(2).TOTARK + PBCARR(2).TODDF);
ATOTYPEC = FLOAT(FLOOR(ATOTYPEC100. + .5));
COMPLET.PCT_ARR = SUBSTR(F(ATOTYPEC,STHREE),1,3);
CURCONF = CONFIND(1); [current configuration index]
FIX(1,*,*) = FIXTRAN(CURCONF,*,*);
TFLAG = 'O'S;
IF SUBSTR(ELIGBLYT(1).ID, CURCONF,1) EQ 'O'S
THEN [If current configuration is currently eligible, then continues]
CC = 0;
CALL CONSERT;

INPUT (CHQGRD(CURCONF));

OUTPUT (CC);

[This routine sets variable CONFIGDATA which signifies runways in a configuration]

CONFIGDATA(1,"*") = CC;

IF (X EQ 2) AND (SUBSTR(ELSGTY(2),ID,CURCONF,1) EQ '1'B)

THEN [If transition is from IFR to WR and current configuration will be ineligible in

   WR conditions]

   X = 1;

   Y = '1'B;

   TFLAG = '1'B;

   XP = 1;

END IF;

CALL DESERT;

INPUT (X, XP, CURCONF, IMPFIRE, PFCARE, CVTXRM, CONFIGDATA, CHQGRD);

OUTPUT (DNE);

[This routine computes demand values for each fix pertaining to current

configuration]

IF Y EQ '1'B

THEN X = 2;

[ determine number of LP variables pertaining to current configuration]

IF CONFIGDATA(1,7) NE 0

THEN VABNUM = 10;

ELSEIF CONFIGDATA(1,6) NE 0

THEN VABNUM = 9;
ELSE \text{ \texttt{VARMIN}} = 8;

\text{INT} = 0;

\text{LOOP:} \{I = 7 \text{ to } 10\} \text{ \{set up variable INT time required for queue flush out for each departure runway based on departure queue length\}}

\text{IF} \text{ \texttt{DM(1,I)} NE 0}

\text{THEN} \text{ \texttt{INT}(I-6) = (3600./\text{DIR}(1,I) \ast \text{CHTL}(R,R))};

\text{ENDIFLOOP;}

\text{NUMAX} = 0;

\text{LOOP:} \{G = 1 \text{ to } 7\} \text{ \{set up number of runways in current configuration\}}

\text{IF} \text{ \texttt{CONFIGDATA}(1,G) NE 0}

\text{THEN} \text{ \texttt{NUMAX} = NUMAX + 1};

\text{ENDIFLOOP;}

\text{LOOP:} \{Q = 1 \text{ to } 73\}

\text{IF} \text{ \texttt{(SUBSTR(HEW(2),1D,Q,1) EQ '0') AND \texttt{(Q NE CURCONF)}}}}

\text{THEN} \text{ \{if configuration Q is eligible in forecast environment and it is not same as current configuration\}}

\text{FINCONF} = \texttt{Q}; \text{ \{final configuration index\}}

\text{CC} = 0;

\text{CALL \texttt{CONSET};}

\text{IN \texttt{(FINCONF)}};

\text{OUT \texttt{(CC)}};

\text{[This routine sets variable CONFIGDATA which signifies runways in a configuration]}
CONFIGDATA(2,*) = CC;
[determine number of LP variables pertaining to final configuration]

IF CONFIGDATA(2,7) NE 0
THEN VARNUN1 = 10;
ELSEIF CONFIGDATA(2,6) NE 0
THEN VARNUN1 = 9;
ELSE VARNUN1 = 8;
COLORMAX = 0;
LOOP: [G = 1 to 7] [set up numbers of runways in final configuration]

IF CONFIGDATA(2,G) NE 0
THEN COLORMAX = COLORMAX + 1;
ENDLOOP;

IF TFLAG = '1'B [if transition is from LPF to VFR and current configuration
will be ineligible in forecast conditions]
THEN

XF = 2;
CALL DENSITY:
IN (X, XP, FINCONF, IMPORM, FRCAARR, CVTDEMN, CONFIGDATA, COMPGRO);
OUT (DEN);  
[This routine computes demand values for each fix
pertaining to final configuration]

FIX(2,*,*) = FIXTRAY(FINCONF,*,*);
CALL SPTRAN;
IN (DO.
VAINH.,
INT. FIX);
OUT (TAITU,. TRANCA);
(special routine to compute
transitioa capacity,
by passing
LPS algorithm)
1! ''
TM (if
tranaitiom
Is
taking place
Io
some
onviroment:
that
is
VFI
to
WI or
II
and
configuration Q
eigible]
IP
-2;
CALL 34 tW;
IN (I. l, 11COIW, IUIO11,
PICARI, CIITDIM,
CONFIGDATA,
OUT (OM)
(This
routine computes
demand
values
for
each
fix
pertaining to
final configuration
)
CALL TDE?;
IN (C=OITI(X(X).
CONFICDTA,
IMT.
DEPI
);
OUT (ETlDIP, TRAVTIN)
(This
routine prepares
dependence
matrix)
LOOP;
(I
- 4
to
7
[if departure
runuays
are
me take
average
IF
CONFIGOATA(1,I)
I COWFIGDATA(2,I)
- I(1,1+3) -
(3M3(1,1+3) +
DIM(2.I+3))/$WO;
3M(2.1+3)
- [DINM (1,1+3);
CALL DISERT;
CALL TERROR;
CALL (TRANSMIT, TRAMAP);
 Effort (special routine to complete transition capacity, by passing
LPS algorithm)
IF (COM, VARIABLES, VARIABLES, INT. PTT):
ELSE (COMING, COMING, INT. PTT):
IN (ROMMAX, COLUMNMAX, VARNUM1, VARNUM2, CONFIGDATA, MATDEF, DEM, TRAVTIM, FIX);
OUT (TRANSSTIME, TRANCAP);
\[This \ routine \ performs \ LP \ algorithm \ and \ determines \ transition \ duration \ and \ capacity\]
ELSE
IF (CONDITION(1) EQ 1) AND (CONDITION(2) EQ 2)
THEN [If transition is from IPR to VPR]
    XP = 2;
    XX = 2;
    CALL INNSRST;
    IN (XX, XP, FINCONF, INFORM, FECARR, CNVTDEA,
        CHIGN, CONFIGDATA, CNFR);  
    OUT (DEM);
    [This routine computes demand values for each fix pertaining to final configuration]
    FIX(2,X,*,*) = FIXTRAV(FINCONF,*,*);
    CALL SPTREN;
    IN (DEM, VARNUM1, VARNUM2, INT, FIX);
    OUT (TRANSSTIME, TRANCAP);
    [This is a special routine to compute transition capacity, bypassing LP algorithm]
ELSEIF SUBSTR(ELGCBLY(2), ID, CURCONF,1) EQ 'O'B
THEN
    XP = 2;
    XX = 1;
CALL DENSEST;
IN (XX, XP, CURCONF, INFORM, PRECAK, CWTEND, CONFIGDATA, CHFURl)
OUT (DDBD);
[This routine computes demand values for each fix pertaining to current configuration]
XX = 2;
XP = 2;
CALL DENSEST;
IN (XX, XP, FINCONF, INFORM, PRECAK, CWTEND, CONFIGDATA);
OUT (DDBD);
[This routine computes demand values for each fix pertaining to final configuration]
FIX(2,*,*) = FIXTRAY(FINCONF,*,*);
CALL TDEP;
IN (CONDITION(X), CONFIGDATA, INT, DEPMT);
OUT (MATDEP, TRAYIN);
[This routine prepares dependence matrix]
LOOP [1 to 4 to 7] (if departure runways are same take average demand)
IF CONFIGDATA (1,1) EQ CONFIGDATA(2,1)
THEN
- \( D_{\text{DEN}}(1,1+3) = (D_{\text{DEN}}(1,1+3) + D_{\text{DEN}}(2,1+3))/2 \)
- \( D_{\text{DEN}}(2,1+3) = D_{\text{DEN}}(1,1+3) \)
ENDLOOP

CALL CALC;

IN (РУММАЕ, СОЛОЕНСЕ, САМОВУР, САМОВРИ, РУММАЕ, РУММЕН, ТАТСУ, ТАТСУ);
OUT (ТРАССТ, ТРАССТ);
(This routine performs LP computation and determines transition duration and capacity)

ELSE
- \( X = 1; \)
- \( XX = 1; \)
CALL ДВИГЕТ;

IN (XX, XP, FINCONF, ИНФОРМ, ПИКСЕР, КВТК, КВТК, КВТК);
OUT (ДВИГЕТ);
(This routine demands values for each fix pertaining to the final configuration)
- \( \text{FIX}(2, *, *) = \text{ФИЛХБАД}(\text{FINCONF}, *, *); \)
CALL СПЦР;

IN (DEN, ВАСКУЛ, ВАСКУЛ, INT, FIX);
OUT (ТРАССТ, ТРАССТ);
(This is a special routine to compute transition capacity, bypassing LP algorithm)
**CLI**

**LIST(Q).INDEX** = Q;
**List_LIST(Q).MINUTES** = TRANTIME;
**List_LIST(Q).FINC** = INFORM(2).COMP(FINCOMP).CAPACITY;
**TOTCAP** = INFORM(2).COMP(FINCOMP).CAPACITY;
**TOTCAP** = TOTCAP * (60 - TRANTIME)/60;
**List(Q).CAP** = TOTCAP + TRANCAP;
**LIST(Q).INDEX** = Q;

**ELSE**

**List(Q).CAP** = -1.0;
**List(Q).INDEX** = 999;
**CMVT_LIST(Q).INDEX** = 0;
**CMVT_LIST(Q).MINUTES** = 0;
**CMVT_LIST(Q).FINC** = 0;

**ENDDO;**

**CALL** OBJRET;

**INOUT** (LIST);

[This routine sorts list of transitions on transition hour capacity]

**COUNT** = 0;

**LOOP;** [N = 1 To 73]

**P** = LIST(N).INDEX;

**IF** P LT 999

**THEN**

**COUNT** = COUNT + 1;
TRANSLST.COMFIG(COUNT).RANK = SUBSTR(FLOAT(COUNT), $THREE), 1, 3)
TRANSLST.COMFIG(COUNT).ARR(1) = CNFGIQ(P).ARR_EHY(1);
TRANSLST.COMFIG(COUNT).ARR(2) = CNFGIQ(P).ARR_EHY(2);
TRANSLST.COMFIG(COUNT).ARR(3) = CNFGIQ(P).ARR_EHY(3);
TRANSLST.COMFIG(COUNT).ARR(4) = CNFGIQ(P).ARR_EHY(4);
TRANSLST.COMFIG(COUNT).ARR(5) = CNFGIQ(P).ARR_EHY(5);
TRANSLST.COMFIG(COUNT).ARR(6) = CNFGIQ(P).ARR_EHY(6);
TRANSLST.COMFIG(COUNT).ARR(7) = CNFGIQ(P).ARR_EHY(7);
TRANSLST.COMFIG(COUNT).ARR(8) = CNFGIQ(P).ARR_EHY(8);
TRANSLST.COMFIG(COUNT).ARR(9) = CNFGIQ(P).ARR_EHY(9);
TRANSLST.COMFIG(COUNT).ARR(10) = CNFGIQ(P).ARR_EHY(10);
TRANSLST.COMFIG(COUNT).ARR(11) = CNFGIQ(P).ARR_EHY(11);
TRANSLST.COMFIG(COUNT).ARR(12) = CNFGIQ(P).ARR_EHY(12);
TRANSLST.COMFIG(COUNT).ARR(13) = CNFGIQ(P).ARR_EHY(13);
TRANSLST.COMFIG(COUNT).ARR(14) = CNFGIQ(P).ARR_EHY(14);
TRANSLST.COMFIG(COUNT).ARR(15) = CNFGIQ(P).ARR_EHY(15);
TRANSLST.COMFIG(COUNT).ARR(16) = CNFGIQ(P).ARR_EHY(16);
TRANSLST.COMFIG(COUNT).ARR(17) = CNFGIQ(P).ARR_EHY(17);
TRANSLST.COMFIG(COUNT).ARR(18) = CNFGIQ(P).ARR_EHY(18);
TRANSLST.COMFIG(COUNT).ARR(19) = CNFGIQ(P).ARR_EHY(19);
TRANSLST.COMFIG(COUNT).ARR(20) = CNFGIQ(P).ARR_EHY(20);

IF SUBSTR(ELGBLTY(2).ID.CURCONF, 1) EQ '0's

THEN COUNT = COUNT + 1;
TRANSLST.NUM_ELIG = SUBSTR(FLOAT(COUNT), $THREE), 1, 3);
E = ELGBLTY(2).NUM;
ELGBLTY(2).NUM = COUNT;
ELSE [current configuration is ineligible]
TRANSLST.MSG = SUBSTR(TRANSLST.MSG, 1, 20) CONCATenate '***CURRENT CONFIGURATION IS INELIGIBLE***';
TRANSLST.NUM_ELIG = SUBSTR(FLOAT(ELGBLTY(2).NUM), $THREE), 1, 3);

TRANSLST.AR(1) = CNFGIQ(CURCONF).ARR_EHY(1);
TRANSLST.AR(2) = CNFGIQ(CURCONF).ARR_EHY(2);
TRANSLST.AR(3) = CNFGIQ(CURCONF).ARR_EHY(3);
TRANSLST.DEP(1) = CNFGIQ(CURCONF).DEP_EHY(1);
TRANSLST.DEP(2) = CNFGIQ(CURCONF).DEP_EHY(2);
TRANSLST.DEP(3) = CNFGIQ(CURCONF).DEP_EHY(3);
TRANSLST.DEP(4) = CNFGIQ(CURCONF).DEP_EHY(4);

IF SUBSTR(ELGBLTY(1).ID.CURCONF, 1) EQ '0's
THEN TRAMLST.CTRANHR = F(INFORM(1).CONF(CURCONF).CAPACITY,$FOUR);
ELSE TRAMLST.CTRANHR = (5)'b';

IF SUBSTR(ELGRLTY(2).ID.CURCONF,1) EQ 'O'b
    THEN TRAMLST.CFINCAP = F(INFORM(2).CONF(CURCONF).CAPACITY,$FOUR);
    ELSE TRAMLST.CFINCAP = (5)'b';

TRANLST.SCROLL = (4)'b';

END TRAN;
ROUTINE CONSET

IN (CNFGRD(Q));

OUT (CC);

[This routine sets variable CONFIGDATA which signifies runways in a configuration]

CC = 0;
K1 = 0;
K2 = 3;

LOOP: [F = 1 to 12]

IF SUBSTR(CNFGRD(Q).ID,F,1) EQ '1'

THEN

K1 = K1 + 1;
CC(K1) = F;

IF SUBSTR(CNFGRD(Q).ID,F+12,1) EQ '1'

THEN

K2 = K2 + 1;
CC(K2) = F;

ENDLOOP;

END CONSET;
ROUTINE DEMSET

IN (X, XP, INDEX, INFORM, FRCARR, CVTORDM, CONFIDATA, CHFGRQ);

OUT (DEN);

[This routine computes demand at fixes for current and forecast configurations]

DEN(1) = CVTORDM(X).DEP.NORTH;
DEN(2) = CVTORDM(X).DEP.EAST;
DEN(3) = CVTORDM(X).DEP.SOUTH;
DEN(4) = CVTORDM(X).DEP.WEST;
DEN(5) = CVTORDM(X).DEP.ME_D;

DISDEN = 0;
E2 = 3;

LOOP: [F = 1 to 12]

IF SUBSTR(CHFGRQ(INDEX).ID,F+12,1) EQ '0'

THEN

DISDEN = DISDEN + DIBDIN(0);

ENDLOOP;

ARRCAP = INFORM(X).CONF(INDEX).ARRCAP + INFORM(X).CONF(INDEX).ARRCAP;
DEPCAP = INFORM(X).CONF(INDEX).DEPCAP + INFORM(X).CONF(INDEX).DEPCAP;

IF FRCARR(X).TOTAAR + FRCARR(X).TODFEP EQ 0

THEN ATOTPRC = .5;
ELSE ATOTPRC = FRCARR(X).TOTAAR/(FRCARR(X).TOTAAR + FRCARR(X).TODFEP);
ETOTFRC = ARRCAP/(ARRCAP + DEPCAP);

IF ATOTFRC GT ETOTFRC

THEN DECP = (1.0 - ATOTFRC)*ARRCAP/ATOTFRC;

ELSEIF ATOTFRC LT ETOTFRC

THEN ARRCAP = ATOTFRC*DECP/(1.0*ATOTFRC);

DE(H(X,1)) = ARRCAP*CHYIRED(X).ARC.RUSK/PRCARE(X).TOTAEE;
DE(H(X,2)) = ARRCAP*CHYIRED(X).ARC.PSMT/PRCARE(X).TOTAEE;
DE(H(X,3)) = ARRCAP*CHYIRED(X).ARC.COT/PRCARE(X).TOTAEE;
DE(H(X,4)) = ARRCAP*CHYIRED(X).ARC.VAGN/PRCARE(X).TOTAEE;
DE(H(X,5)) = ARRCAP*CHYIRED(X).ARC.PARNO/PRCARE(X).TOTAEE;
DE(H(X,6)) = ARRCAP*CHYIRED(X).ARC.XEX_A/PRCARE(X).TOTAEE;

IF PRCARE*(X).TOTDEP GT 0

THEN

REPEAT WHILE CONFIGDATA(XP, K+3) NE 0 [K = 1 TO 4]

DE(H(XP, K+6)) = DEPCAP*DISPRED(X)/PRCARE(X).TOTDEP;

ENDREPEAT;

END DEMSET;
Routine TDEP

IN (X, CONFIGDATA, INT, DEPMAT);

OUT (MATDEP, TRAVTIM);

[This routine computes dependence matrix]

MATDEP = 0;

LOOP [0 = 1 To 7]

IF CONFIGDATA(1, G) NE 0

THEN

LOOP: [H = 1 to 7]

IF (G LE 3) AND (H LE 3)

THEN JJ = 1;

ELSEIF (G LE 3) AND (H GT 3)

THEN JJ = 2;

ELSEIF (G GT 3) AND (H LE 3)

THEN JJ = 3;

ELSE JJ = 4;

IF CONFIGDATA(2, H) NE 0

THEN

MATDEP(G, H) = DEPMAT(X).SECT(JJ).MATRIC(CONFIGDATA(1, G), CONFIGDATA(2, H));

ENDLOOP;

ENDLOOP:
TRAVIN = 0;

IF CONFIGDATA(1,7) NE 0 THEN DP = 4;
ELSEIF CONFIGDATA(1,6) NE 0 THEN DP = 3;
ELSE DP = 2;
LOOP: [R = 1 to DP]
QFLAG = 0
LOOP: [RR = 1 to 7]
IF MATHEP(3 + R, RR) NE 0 THEN
MATHEP(3 + R, RR) = MATHEP(3 + R, RR) + INT(R);
QFLAG = 1;
ENDLOOP;

IF QFLAG EQ 0 THEN TRAVIN(R) = INT(R);

END LOOP;
END TRYP;
ROUTINE SPTBSM

IN (QBM, VARMNI, VARMNO, INT, FIX);

OUT (TRANSINI, TRANSAP);  
(This routine performs a special transition algorithm in cases when two configurations are not
mutually eligible)

SAFESEP = 3;
FIXTRAV = FIX;
DEM1 = DEM(1, *);
DEM2 = DEM(2, *);

TEMP = 0;

[compute transition duration]

LOOP:  [N = 1 to 3]

LOOP:  [L = 1 to 6]

IF FIXTRAV(1, M, L) OT TEMP

THEN

TEMP = FIXTRAV(1, M, L);
RDIND = N;
FIXIND = L;

ENDLOOP;

ENDLOOP;

CURTIME = TEMP;

LOOP:  [N = 1 to 6]

IF N BE FIXIND

THENIF CURTIME EQ FIXTRAV(1, RDIND, M)
\[
\text{THEN } \text{CURTIME} = \text{CURTIME} + \text{SAFESEP}; \text{ [contribution of current configuration to transition duration]}
\]

ENDLOOP;

\text{FIXTAV} (1, \text{REWIND}, \text{FIXING}) = \text{CURTIME};

\text{TEMP} = 0;

\text{LOOP: } [M = 1 \text{ to } 4]

\text{TEMP} = \text{MAX(TEMP, INT(H))};

\text{ENDLOOP};

\text{CURTIME} = \text{MAX(CURTIME, TEMP)}; \text{ [include effect of departure queues]}

\text{IF } \text{CURTIME} \text{ GE } 60.

\text{THEN } \text{CURTIME} = 59;

\text{TEMP} = 0;

\text{LOOP: } [M = 1 \text{ to } 3]

\text{LOOP: } [L = 1 \text{ to } 6]

\text{IF } \text{FIXTAV}(2, M, L) \text{ GT TEMP}

\text{THEN}

\text{TEMP} = \text{FIXTAV}(2, M, L);

\text{REWIND} = M;

\text{FIXING} = L;

\text{ENDDOOP};

\text{ENDLOOP};

\text{FIXTIME} = \text{TEMP};

\text{LOOP: } [M = 1 \text{ to } 6]
IF \( M \) BE FIXIND

THEN

IF \( \text{FINTIME} \neq \text{FIXTRAV}(2, \text{MUIND}, M) \)

THEN \( \text{FINTIME} = \text{FINTIME} + \text{SAPESEF} \); \{\text{contribution of current configuration to transition duration}\}

ENDLOOP;

\( \text{FIXTRAV}(2, \text{MUIND}, \text{FIXIND}) = \text{FINTIME} \);

\( \text{TRANSIND} = \text{MAX}(\text{FINTIME, CURTIME}) \); \{\text{transition duration}\}

IF \( \text{TRANSIND} \neq \text{CURTIME} \) [\{\text{if transition duration is driven from current configuration}\}]

THEN

\{\text{compute variables for current configuration}\}

\( \text{VAR1} = 0; \)

\( \text{LOOP:} \ [L = 1 \text{ to } 6] \)

\( \text{LOOP:} \ [L = 1 \text{ to } 6] \)

\( \text{IF} \ \text{FIXTRAV} (1, M, L) \neq 0 \)

\( \text{THEN} \ \text{VAR1}(L) = \text{FIXTRAV}(1, M, L); \)

ENDLOOP;

ENDLOOP;

\( \text{LOOP:} \ [N = 7 \text{ to } \text{VARIND}] \)

\( \text{VAR1}(N) = \text{CURTIME} \)

ENDLOOP;

\( \text{TRANSCEF} = 0; \)

\( \text{LOOP:} \ [T = 1 \text{ to } \text{VARIND}] \)
\[ T - \text{ANCAP} - \text{RACAP} + T(t)\sqrt{\text{VAI}(T)/60}. \quad \text{[compute transition capacity]} \]

ENDLOOP;

ELSE [if transition duration is driven from final configuration]

\[ \text{VARL} = 0; \]

LOOP; \([N = 1 \text{ to } 3]\)

LOOP; \([L = 1 \text{ to } 6]\)

IF \text{FIXTRAV}(1,N,L) NE 0

THEN \text{VARL}(L) = \text{FIXTRAV}(1,N,L);

ENDLOOP;

ENDLOOP;

LOOP; \([N = 7 \text{ to } \text{VARNUM}] \quad \text{[compute variables for current configuration]}\]

\[ \text{VARL}(N) = \text{OPER}; \]

COUNT = 1;

LOOP; \([N = 1 \text{ to } 3]\)

LOOP; \([L = 1 \text{ to } 6]\)

IF \text{FIXTRAV}(2,N,L) NE 0

THEN

IF \text{FIXTRAV}(2,N,L) GT \text{CURTIME}

THEN \(P(\text{COUNT}) = L; \)

COUNT = COUNT + 1;
ENDLOOP;
ENDLOOP;

[compute transition capacity]
TRANCAP = 0.;
LOOP; T = 1 to VARNUN1]
TRANCAP = TRANCAP + DEMI(T)*VARI(T)/60.;
ENDLOOP;
CONST = FINTIME - CURTIME;
LOOP; [T = 1 to VARNUN2]}
FLAG = '0'B;
LOOP; [K =1 to COUNT - 1]}
IF T EQ F(K)
THEN FLAG = '1'B;
ENDLOOP;
IF FLAG EQ '0'B
THEN TRANCAP = TRANCAP + DEM2(T)*CONST/60.;
ENDLOOP;
END SPTTRAN;
ROUTINE CALC

IN (ROMMAX, COLDEMAX, VARNUM1, VARNUM2, CONFIGDATA, DEPMAT, DEM, TRAVTIN, FIX);

OUT (TRANSITIME, TRANSCAP);

[This routine computes LP solution using special algorithm, also transition duration is computed]

CRCONF = 1;
FINCONF = 2;
FIXTRAV = FIX;
DEPHAT = DEPHAT/60.; [conversion from seconds to minutes]

IF (CONFIGDATA(1,1) EQ CONFIGDATA(2,1)) AND (CONFIGDATA(1,2) EQ CONFIGDATA(2,2)) AND
(CONFIGDATA(1,3) EQ CONFIGDATA(2,3))

THEN
CALL ADJUST;

IN (TRAVTIN, CONFIGDATA, DEPMAT, DEM);

OUT (TRANSITIME, TRANSCAP);

[If both configurations in transition have same arrival runways then routine ADJUST
uses a different algorithm to compute transition duration and capacity]

ELSE
CALL DUR;

IN (CONFIGDATA, DEPMAT, FIXTRAV);

OUT (TLAG, TRANSITIME);

[This routine computes transition duration]

TRAVTIN = TRAVTIN/60.; [conversion from seconds to minutes]

IF SUM(TRAVTIN) NE 0 [modify transition duration with information on current departure
queues]

THEN COMPARXX = 0;
LOOP; \{W = 1 to 4\}

IF \(\text{TRAVTIM}(W) \gt C\O\text{NAEX}\)
THEN \(C\O\text{NAEX} = \text{TRAVTIM}(W)\)
ENDLOOP;

IF \(C\O\text{NAEX} \ge 60.\)
THEN \(C\O\text{NAEX} = 59;\)
IF \(\text{TFLAG} = 0;\)
THENIF \(C\O\text{NAEX} \gt \text{TRANSTIME}\)
THEN \(\text{TRANSTIME} = \text{C\O\text{NAEX}};\)
ELSEIF \(C\O\text{NAEX} \le (\text{TRANSTIME} - \text{SAFESEP})\)
THEN \(\text{TRANSTIME} = \text{C\O\text{NAEX}};\)
IF \(\text{TRANSTIME} \ge 60;\)
THEN \(\text{TRANSTIME} = 59;\)
IF \((\text{CONFIDATA}(1,1) \neq \text{CONFIDATA}(2,1))\) \(\text{OR} \) \((\text{CONFIDATA}(1,2) \neq \text{CONFIDATA}(2,2))\) \(\text{OR} \) \((\text{CONFIDATA}(1,3) \neq \text{CONFIDATA}(2,3))\)
THEN
\(\text{EDM} = 0;\)
CALL \text{EDP};
IN \((\text{ROWNUM}, \text{COLUMNMAX}, \text{DEMAT}, \text{CONFIDATA}, \text{FIXTRAY});\)
OUT \((\text{EDM});\)
[\text{This routine prepares expanded dependence matrix}]
[\text{Compute upperbound constraints}]
LOOP: [I = 1 to VARNUM2]
UB2(I) = TRANSTIME
ENDLOOP;
LOOP: [I = 1 to 7]
COMPAREX = 0;
LOOP: [J = 1 to 7]
IF DEPMAT(I,J) GT COMPAREX
THEN COMPAREX = DEPMAT(I,J);
ENDLOOP;
MAXDELAY(I) = COMPAREX;
ENDLOOP;
LOOP: [{I = 1 to 6}]
LOOP: {J = 1 to 3}
IF FIXTRAV(CURCONF,J,I) NE 0
THEN URL(I) = TRANSTIME - MAXDELAY(J) - FIXTRAV(CURCONF,J,I)
IF ABS(URL(I)) LT .001
THEN URL(I) = 0
ENDLOOP;
ENDLOOP;
IF CONFIGDATA(1,3) NE 0
THEN I3 = 3;
SOFTWARE DESCRIPTION FOR THE O'HARE RUNWAY
CONFIGURATION MANAGEMENT SYSTEM

MITRE CORP MCLEAN
VA METREK DIV S KAVOUSSI OCT 82 NAV-82W125-VOL-2

UNCLASSIFIED FAA-EM-82-28-VOL-2 DTFA01-81-C-10003 F/O 17/7
ELSE \( I_3 = 4; \)

LOOP: [\( I = 7 \) to \( \text{VARNUM1} \)]

\( U_{21}(I) = \text{TRANSTIME} - \text{MAXDELAY}(I - 13) - \text{TRAVTIM}(I - 6) \)

ENDLOOP;

[determine initial solutions for LP variables]

LOOP: [\( N = 1 \) to \( \text{VARNUM1} \)]

COMPARE = 0;

LOOP: [\( N = 1 \) to \( \text{VARNUM1} \)]

IF \( \text{EDM}(N,N) \gt \text{COMPARE} \)

THEN COMPARE = EDM(N,N);

ENDLOOP;

VAR2(N) = COMPARE;

ENDLOOP;

LOOP: [\( N = 1 \) to \( \text{VARNUM1} \)]

COMPARE = TRANSTIME;

LOOP [\( N = 1 \) to \( \text{VARNUM2} \)]

IF \( N \not= N \) OR \( \text{EDM}(N,N) \neq 0 \)

THEN

\( \text{INDEX} = \text{MIN}((\text{VAR2}(N) - \text{EDM}(N,N),U_{21}(N)) \)

IF INDEX \lt \text{COMPARE}

THEN COMPARE = INDEX;

ENDLOOP;
VAR1(N) = COMPARE;
ENDLOOP;
PERFORM LP_ALGORITHM_ITERATIONS;
[compute final transition capacity using LP solution and transition duration]
TRANCAP = OBJFUN (DEM1, DEM2, TRANSTIME, VAR1, VAR2, FIXTRAV, VARNUM1, VARNUM2, CONFIGDATA, TRAVTIM);
END CALC;
PROCESS LP_ALGORITHM_ITERATION
[This process performs LP algorithm]

TEMP = "S;"

LOOP: [1 = 1 to VARNUM2]

TEMP = TEMP CONCATENATE '0';

ENDLOOP;
[construct a matrix with zeroes and ones; 1's along diagonal]

CNT = 0;
ALT = 0;

REPEAT UNTIL (FLAG EQ 0);

CNT = CNT + 1;
FLAG = 0;

LOOP: [I = 1 to VARNUM1]

LOOP: [J = 1 to VARNUM2]

IF ((I EQ J) OR (EDM(I,J) NE 0)) AND (VAR2(J) - VAR1(I) EQ EDM(I,J))

THEN ALT(I,J) = 1;
ELSE ALT(I,J) = 0;

ENDLOOP;

ENDLOOP;

ALT1 = ALT;
COUNTER = 0;
ROW = "B;"
COL = "B;"

[construct hit strings with rows and columns of matrix ALT]
LOOP [I = 1 to VARNUM 1]

LOOP [J = 1 to VARNUM 2]

IF ALT(I,J) EQ 1

THEN

COUNTER = COUNTER + 1;
ALT(I,J) = -1;

LOOP [K = 1 to VARNUM 2]

IF ALT(I,K) NE 0

THEN

COL(COUNTER) = COL(COUNTER) CONCATENATE '1' B
ALT(I,K) = -1;

ELSE

COL(COUNTER) = COL(COUNTER) CONCATENATE '0' B;

END LOOP;

LOOP [K = 1 to VARNUM 1]

IF ALT(K,J) NE 0

THEN

ROW(COUNTER) = ROW(COUNTER) CONCATENATE '1' B;
ALT(K,J) = -1;

ELSE

ROW(COUNTER) = ROW(COUNTER) CONCATENATE '0' B;

END LOOP;

IF COUNTER GT 1

THEN

L = COUNTER;
LOOP; [K = COUNTER - 1 to 1 BY -1]

AUX1 = ROW(L) AND ROW(K);
AUX2 = COL(L) AND COL(K);

IF (AUX1 GT 0) OR (AUX2 GT 0)

THEN
ROW(K) = ROW(K) OR ROW(L);
COL(K) = COL(K) OR COL(L);
ROW(L) = "B"
COL(L) = "B"
IND = 0;

LOOP; [K = 1 to COUNTER]

IF TEMP1(K) NE "B"

THEN
IND = IND + 1;
ROW(IND) = TEMP1(K);
COL(IND) = TEMP2(K);

ENDLOOP;
COUNTER = IND;

ENDLOOP;

ENDLOOP;

ST = "S"

LOOP; [R = 1 to VARNUM1]

LOOP; [Q = 1 to VARNUM 2]

IF ALT(R, Q) EQ 1

THEN ST(R) = ST(R) CONCATENATE '1';
ELSE \texttt{ST(R) = ST(R) CONCATENATE '0';}

\texttt{ENDLOOP;}

\texttt{ENDLOOP;}

\texttt{LOOP; \{I = 1 to COUNTER\}}

\texttt{ROWSTCK = 0;}
\texttt{COLCOUNT = 0;}
\texttt{COUNT = 0;}

\texttt{LOOP; \{Q = 1 to VARNUM1\}}

\texttt{IF SUBSTR(Row(I),Q,1) EQ '1'\}}

\texttt{THEN}
\texttt{COUNT = COUNT + 1;}
\texttt{ROWSTCK(COUNT) = Q;}

\texttt{ENDLOOP;}

\texttt{LOOP; \{R = 1 to COUNT\}}

\texttt{Q = ROWSTCK(R);}
\texttt{COLCOUNT = 0;}

\texttt{LOOP; \{S = 1 to VARNUM2\}}

\texttt{IF ALT(Q,S) EQ 1}

\texttt{THEN COLCOUNT = COLCOUNT + 1;}

\texttt{ENDLOOP;}
\texttt{COLCOUNT(R) = COLCOUNT;}

\texttt{ENDLOOP;}
LOOP: [Q = 2 to COUNT]

QQ = Q;
SORT_FLAG = 0

REPEAT WHILE ((QQ GE 1) AND (SORT_FLAG EQ 0));

IF COLCOUNT(QQ) GT COLCOUNT(QQ - 1)

THEN

TEMP1 = COLCOUNT(QQ-1);
TEMP2 = RONSTCK(QQ-1);
COLCOUNT(QQ-1) = COLCOUNT(QQ);
RONSTCK(QQ-1) = RONSTCK(QQ)
COLCOUNT(QQ) = TEMP1;
RONSTCK(QQ) = TEMP2;
QQ = QQ - 1;

ELSE SORT_FLAG = 1;

ENDREPEAT;

ENDLOOP;

SUBCOUNT = COUNT;
PSUDEK = DESI;
ALTRONSTCK = 0
IND3 = 0;

LOOP: [Q = 1 to COUNT]

IF PSUDEK(RONSTCK(Q)) NE 0

THEN

IND3 = IND3 + 1;
ALTRONSTCK(IND3) = RONSTCK(Q);
QQ = Q;

ENDLOOP;
LOOP: [R = QQ + 1 to COUNT]

IF ST(ROWSTCK(Q)) EQ ST(ROWSTCK(R))

THEN

PSUDEM(ROWSTCK(Q)) = PSUDEM(ROWSTCK(Q)) + DEM1(ROWSTCK(R));
PSUDEM(ROWSTCK(R)) = 0;
SUBCOUNT = SUBCOUNT - 1;

ENDLOOP;

ENDLOOP;

ROWSTCK = ALTRWSTCK;
COUNT = SUBCOUNT;

X = 0;
IND1 = 0;

LOOP: [Q = 1 to COUNT]

IND2 = 1;
IND1 = IND1 + 1;
X(IND1, IND2) = ROWSTCK(Q);

LOOP: [R = Q + 1 to COUNT]

IF (ST(ROWSTCK(Q)) OR ST(ROWSTCK(R)) EQ ST(ROWSTCK(Q))

THEN

IND2 = IND2 + 1;
X(IND1, IND2) = ROWSTCK(R);

ENDLOOP;

ENDLOOP;

IF ST(ROWSTCK(1)) NE COL(1)
THEN
IND_1 = IND_1 + 1;
LOOP: [Q = 1 to COUNT]
X(IND_1, Q) = ROWSTCK(Q);
ENDLOOP;
SFLAG = 0;
REPEAT WHILE (SFLAG EQ 0); [R = 1 to IND_1]
ADD1 = 0;
ADD2 = 0;
NEWST = "B";
REPEAT WHILE (X(R,Q) NE 0); [Q = 1 to IND_2]
ADD1 = ADD1 + PSIWDX(X(R,Q));
NEWST = NEWST OR ST(X(R,Q));
ENDREPEAT;
LOOP: [T = 1 to VARNUM 2]
IF SUBSTR(NEWST,T,1) EQ '1'B
THEN ADD2 = ADD2 + DEM2(T);
ENDLOOP;
IF ADD2 LT ADD1;
THEN
UPSLCK2 = 9999.;
UPSCLCK2 = 9999.;
MINSLCK = 9999.;
REPEAT WHILE (X(R,Q) NE 0); [Q = 1 to 10]
LOOP \( [W = 1 \text{ to } \text{VARM}21] \)

\[ \text{IF} \quad \text{SUBSTR(ST}(X(R,Q),W,1) \text{ EQ '0'\}}\]

\[ \text{THEN} \]

\[ \text{IF} \quad (\text{EDM}(X(R,Q),W) \text{ NE } 0) \text{ OR } X(R,Q) \text{ EQ } W) \]

\[ \text{THEN} \]

\[ \text{SLC} = \text{VARZ}(W) - \text{VAR1}(W) - \text{EDM}(X(R,Q),W); \]

\[ \text{MINSLC} = \text{MIN} (\text{SLC}, \text{MINSLC}); \]

\[ \text{ELSE} \]

\[ \text{SLC} = \text{VAR2}(W) - \text{VAR2}(W); \]

\[ \text{UPSLC2} = \text{MIN} (\text{SLC}, \text{UPSLC2}); \]

\[ \text{ENDDO} \]

\[ \text{SLC} = \text{UI1}(X(R,Q)) - \text{VAR1}(X(R,Q)); \]

\[ \text{UPSLC1} = \text{MIN} (\text{SLC}, \text{UPSLC1}); \]

\[ \text{ENDDO} \]

\[ \text{ABSMIN} = \text{MIN} (\text{MINSLC}, \text{UPSLC1}, \text{UPSLC2}); \]

\[ \text{IF} \quad \text{ABSMIN GT .001} \]

\[ \text{THEN} \]

\[ \text{FLAG} = 1; \]

\[ \text{SFLAG} = 1; \]

\[ \text{REPEAT} \quad \text{WHILE} \quad (X(R,Q) \text{ NE } 0) \quad [Q = 1 \text{ to } \text{VARMIN}1] \]

\[ \quad \text{VAR1}(X(R,Q)) = \text{VAR1}(X(R,Q)) + \text{ABSMIN}; \]

\[ \text{ENDREPEAT} \]

\[ \text{LOOP} \quad [W = 1 \text{ to } \text{VARM}21] \]

\[ \text{IF} \quad \text{SUBSTR(HE5ST,W,1) EQ '1'\}} \]

\[ \quad \text{THEN} \quad \text{VAR2}(W) = \text{VAR2}(W) + \text{ABSMIN}; \]
ENDLOOP

ENDREPEAT;

ENDLOOP;

ENDREPEAT;

END LOOP LP_ALGORITHM_ITERATION;
ROUTINE DUR

IN (CONFIGDATA, DEPMAT, FIXTRAV);

OUT (TFLAG, TRANSTIME);
[This routine computes transition duration]

TFLAG = 0;
CURCONF = 1;
FINCONF = 2;
SAFESEP = 3;
MAXDELAY = 0; [compute contribution of current configuration to transition duration]

LOOP: [I = 1 To 77]

COMPARE1 = 0;

IF CONFIGDATA(CURCONF,I) NE 0

THEN

LOOP [J = 1 to 7]

IF DEPMAT(I,J) GT COMPARE1;

THEN COMPARE1 = DEPMAT(I,J);

ENDLOOP;
MAXDELAY(I) = COMPARE1;

ENDLOOP;

COMPARE1 = 0;

REPEAT WHILE (CONFIGDATA(CURCONF,I) NE 0); [I = 1 to 3]

COMPARE2 = 0;

LOOP: [J = 1 To 6]
IF FIXTRAV(CURCONF, I, J) NE 0
    THENIF FIXTRAV(CURCONF, I, J) GT COMPARE2
        THEN
            COMPARE2 = FIXTRAV(CURCONF, I, J);
            FIX_INDICATOR = J;
            RUN_INDICATOR = I;
        ENDLOOP:
    COMPARE2 = COMPARE2 + MAXDELAY(I);
    IF COMPARE2 GT COMPARE1
        THEN
            COMPARE1 = COMPARE2;
            FIX_INDICATOR1 = FIX_INDICATOR;
            RUN_INDICATOR1 = RUN_INDICATOR
        ENDCOMPARE1:
    CURTIME = COMPARE1;
    LOOP; [I = 4 to 7]
        IF CURTIME LT MAXDELAY(I)
            THEN CURTIME = MAXDELAY(I);
        ENDCURTIME:
    ENDCOMPARE1:
[compute contribution of final configuration to transition duration]
COMPAR1 = 0;
REPEAT WHILE (CONFIGDATA(FINCONF, I) NE 0); [I = 1 to 3]
    COMPARE2 = 0;
    LOOP; [J = 1 to 6]
        IF FIXTRAV(FINCONF, I, J) NE 0
THEN IF FIXTRAV(FINCONF, I, J) GT COMPARE2
    THEN
        COMPARE2 = FIXTRAV(FINCONF, I, J);
        FIX_INDICATOR = J;
        RUN_INDICATOR = I;
    ENDLOOP;
    IF COMPARE2 GT COMPARE1
        THEN
            COMPARE1 = COMPARE2;
            FIX_INDICATOR1 = FIX_INDICATOR;
            RUN_INDICATOR2 = RUN_INDICATOR;
        ENDREPRAT;
    FINTIME = COMPARE1;
    [compute transition duration]
    IF CURTIME GT FINTIME
        THEN
            TRASTIME = CURTIME;
            FIX_INDICATOR = FIX_INDICATOR1;
            RUN_INDICATOR = RUN_INDICATOR1;
            DURAT = TRASTIME_MAXDELAY(RUN_INDICATOR1);
            LOOP: [1 to 6]
                IF (DURAT EQ FIXTRAV(CURCONF, RUN_INDICATOR1, I)) AND I NE FIX_INDICATOR1
                    THEN
                        TRASTIME = TRASTIME + SAFETF;
                        TFLAG = 1;
                ENDLOOP;
ELSEIF TRANS_TIME GT CUR_TIME THEN

TRANS_TIME = FIN_TIME;
DUMMY = TRANS_TIME;
LOOP: [I = 1 to 6]

IF (DUMMY EQ FIXTRAV(FINCONF, RUN_INDICATOR1, I)) AND (I NE FIX_INDICATOR2)

THEN
TRANS_TIME = TRANS_TIME + SAFESEP;
TFLAG = 1;
ENDLOOP;

ELSE
TRANS_TIME = CUR_TIME;
DUMMY = TRANS_TIME - MAXDELAY(RUN_INDICATOR1);
LOOP: [I = 1 to 6]

IF (DUMMY EQ FIXTRAV(CURCONF, RUN_INDICATOR1, I)) AND (I NE FIX_INDICATOR1)

THEN
TRANS_TIME = TRANS_TIME + SAFESEP;
TFLAG = 1;
ENDLOOP;

DUMMY = TRANS_TIME;
LOOP: [I = 1 to 6]

IF (DUMMY EQ FIXTRAV(FINCONF, RUN_INDICATOR2, I)) AND (I NE FIX_INDICATOR2)

THEN
TRANS_TIME = TRANS_TIME + SAFESEP;
TFLAG = 1;
ENDLOOP;

END DUR;
ROUTINE ADJST

IN (TRANSTIM, CONFIGDATA, DEPMAT, DEM);

OUT (TRANSTIME, TRANCAP);

[If two configurations in transition have same arrival runways then routine ADJST uses a different algorithm to compute transition duration and capacity]

TRANSTIME = 0;
TRANCAP = 0;

LOOP: [I = 4 to 7]

COMPARE = 0;

LOOP: [J = 1 to 7]

IF DEPMAT(I,J) GT COMPARE

THEN COMPARE = DEPMAT(I,J);

ENDLOOP;

MAXDELAY(I-3) = COMPARE;

ENDLOOP;

ITEMP = 0;

LOOP: [I = 1 to 4]

ITEMP = TRAVTIM(I) + MAXDELAY(I);
ITEMP = MAX(ITEMP, TEMP);

ENDLOOP;

TRANSTIME = ITEMP;

IF TRANSTIME NE 0;

2-389
THEN
CAP = 0;
LOOP; [I = 1 to 10]
    CAP = CAP + DEM(I, I);
ENDLOOP;
TRANCAP = CAP * (TRANSTIM/60);
END Ajust;
ROUTINE RDF

IN (ROWMAX, COLUMNMAX, DEPHAT, CONFIGDATA, FIXTRAY);
OUT (RDW);

[This routine prepares expanded dependence matrix]
CURCONF = 1;
FINCONF = 2;
SAFESEP = 3;

[initialization]
IF CONFIGDATA(CURCONF, 3) NE 0
THEN II = 3; [number of arrival runways in current configuration]
ELSE II = 2;
IF CONFIGDATA(CURCONF, 7) NE 0
THEN 12 = 4;
ELSEIF CONFIGDATA(CURCONF, 6) NE 0
THEN 12 = 3;
ELSE 12 = 2;
IF CONFIGDATA(FINCONF, 3) NE 0
THEN J1 = 3; [number of arrival runways in final configuration]
ELSE J1 = 2;
IF CONFIGDATA(FINCONF, 7) NE 0
THEN J2 = 4;
ELSEIF CONFIGDATA(FINCONF, 6) NE 0
THEN $J_2 = 3$
ELSE $J_2 = 2$

IF $I_1 \equiv 2$
THEN $I_3 = 4$
ELSE $I_3 = 3$

IF $J_1 \equiv 2$
THEN $J_3 = 4$
ELSE $J_3 = 3$

LOOP: [$I = 1$ to $II$]
INDEX1 = 1
LOOP: [$K = 1$ to $6$]

IF FISTRAY($CONCONF$, $I$, $K$) NE 0
THEN
RECORD1(IINDEX1) = $K$
INDEX1 = INDEX1 + 1;
ENDLOOP;

LOOP: [$J = 1$ to $II$]
INDEX2 = 1
LOOP: [$K = 1$ to $6$]

IF FISTRAY($CONCONF$, $J$, $K$) NE 0
THEN
RECORD2(INDEX2) = $K$
INDEX2 = INDEX2 + 1;
ENDLOOP;

LOOP; [L = 1 to INDEX1 - 1]
  LOOP; [M = 1 to INDEX2 - 1]
    EKN(RECORD(1),RECORD(2)) = DEPMAT(I,J);
  ENDLOOP;
ENDLOOP;
ENDLOOP;
ENDLOOP;

ENDLOOP;

ENDLOOP;

ENDLOOP;
LOOP; [L = L1 + 1 to ROWMAX]
  LOOP; [K = 1 to COLUMNMAX]
    IF K LE J1
      THEN:
        LOOP; [M = 1 to 6]
          IF FIXTRAY(FIXCOMP,K,M) NE 0
            THEN
              EKN(L+13,M) = DEPMAT(34-L+11,K);
            ENDIF;
          ENDLOOP:
        ENDLOOP:
    ENDIF;
  ENDLOOP;
ENDLOOP;

ENDLOOP;
LOOP; [L = J1 + 1 to COLUMNMAX]
  LOOP; [K = 1 to ROWMAX]
IF K LE 11
THEN
  LOOP; [M = 1 to 6]
  IF FIXTRAV(CURCONF, K, M) NE 0
  THEN EDM(N, L + J3) = DEPMAT(K, L+3-J1);
  ENDLOOP;
ENDLOOP;

ENDLOOP;

ENDLOOP;

LOOP; [L = 11 + 1 to ROWMAX]
  LOOP; [N = J1 + 1 to COLUMNMAX]
    EDM(L + 13, N + J3) = DEPMAT(L+3-11, N+3-J1);
  ENDLOOP;
ENDLOOP;

ENDLOOP;

LOOP; [K = 1 to 6]
  IF EDM(K, K) EQ 0.
  THEN
    LOOP; [L = 1 to J1]
      IF FIXTRAV(FINCONF, L, K) NE 0
      THEN EDM(K, K) = FIXTRAV(FINCONF, L, K);
      ENDLOOP;
    ELSE
LOOP; [L = 1 to ll]
  IF FIXTRAV(CURCONF,L,K) NE 0
    THEN EDM(K,K) = EDM(K,K) + FIXTRAV(CURCONF,L,K);
  ENDLOOP;
ENDLOOP;

LOOP; [I = 1 to 6]
  LOOP; [J = 1 to 64J2]
    IF (EDM(I,J) NE 0) AND (I NE J)
      THEN LOOP; [K = 1 to ll]
        IF FIXTRAV(CURCONF,K,I) NE 0
          THEN EDM(I,J) = EDM(I,J) + FIXTRAV(CURCONF,K,I);
        ENDLOOP;
    ENDLOOP;
ENDLOOP;
ENDLOOP;
END EDP;
FUNCTION OBJFUN

IN (DEMI, DEM2, TRANSTIME, VAR1, VAR2, FIXTRAV, VARNUM1, VARNUM2, CONFIGDATA, TRAVTIN);

OUT (TRANCAP);

[This routine computes the value of the objective function which is transition capacity]

CURCONF = 1;
CAP = 0.;
$SIXTY = 60;

LOOP; [I = 1 to VARNUM1]

IF I LE 6

THEN

LOOP [J = 1 to 3]

IF FIXTRAV(CURCONF, J, I) NE 0

THEN

CAP = CAP + (VAR1(I) + FIXTRAV(CURCONF, J, I) * DEMI(I)/$SIXTY;

ENDLOOP;

ELSE

ENDLOOP

CAP = CAP + (TRAVTIN(I-6) + VAR1(I)*DEMI(I)/$SIXTY;

ENDLOOP;

TRANCAP = CAP;

END OBJFUN;
2.15 Configuration Information Screen

The Configuration Information Screen is presented on pages 2-398 to 2-420.
STRUCTURE CONFIG_DATA(2) LIKE CONFIG
   [This structure is similar to CONFIG used internally by screen programs]
ENDSTRUCTURE;

INT CINDEX(2) [Integers containing current operating configuration's index similar to CONFIND used internally by screen programs]

INT SWITCH(2) [This variable is used for switching between current and forecast screens, initialized to (1,2)]

STRUCTURE CONFIG_LOADLIST [A structure of pointers, one for each data field on screen used by panel manager for loading and unloading to and from screen]

PTR TIME [Pointer for environment data field]

GROUP CONF
   PTR ARR(12) [Pointer for arrival runway indicator data field]
   PTR DEF(12) [Pointer for departure runway indicator data field]

GROUP TOTAL
   PTR PCT_ARR [Pointer for total percentage of arrivals data field]
   PTR SAT [Pointer for total saturation data field]

GROUP ARR
   PTR DEM [Pointer for total arrival demand data field]
   PTR CAP [Pointer for total arrival capacity data field]

[***LOCAL VARIABLES***]
GROUP DEP

PTR DEM [pointer for total departure demand data field]
PTR CAP [pointer for total departure capacity data field]

GROUP NORTH

PTR PCT_ARR [pointer for north percentage of arrivals data field]
PTR SAT [pointer for north saturation data field]

GROUP ARR

PTR DEM [pointer for north arrival demand data field]
PTR CAP [pointer for north arrival capacity data field]

GROUP SOUTH

PTR PCT_ARR [pointer for south percentage of arrivals data field]
PTR SAT [pointer for south saturation data field]

GROUP ARR

PTR DEM [pointer for south arrival demand data field]
PTR CAP [pointer for south arrival capacity data field]

GROUP DEM

PTR DEM [pointer for south departure demand data field]
PTR CAP [pointer for south departure capacity data field]

GROUP BALANCING

PTR ANDVR [pointer for arrival aircraft moved data field]
PTR ACOMPLEX [pointer for complex data field]
PTR DMOVE [pointer for departure aircraft moved data field]

PTR DCOMPLEX [pointer for complex data field]

PTR WMSG0 [pointer for 1st warning message data field]

PTR WMSG1 [pointer for 2nd warning message data field]

PTR WMSG2 [pointer for 3rd warning message data field]

PTR MSG [pointer for screen message data field]

BITS FENCE [32 bit variable as prescribed by DMS manual, initialized to string of (32) '1'B]

ENDSTRUCTURE;
ROUTINE CNFG

IN (CONFIG, CNFGSRQ, PRCAEPH, INFORM, MIDFLAG, RWYEQP);

INOUT (CONFIND, RSTATUS, I);

\[\text{This routine invokes configuration information screen for both current and forecast environments}\]

CNFG DATA = CONFIG;
CINDEX = CONFIND;

REPEAT UNTIL (RSTATUS NE PF12);

CNFG DATA(I) = CONFIG(I);
CINDEX(I) = CONFIND(I);
I = SWITCH(I); \[\text{switch between two screens}\]

REPEAT UNTIL (RSTATUS NE PF9);
I = SWITCH(I);

CALL CGSCREEN;

IN (CNFG_DATA(I),CINDEX(I),CNFGSRQ, PRCAEPH, INFORM(I), MIDFLAG(I), RWYEQP(I));

INOUT (RSTATUS);

\[\text{This routine controls configuration information screen}\]

ENDREPEAT;

ENDREPEAT;

LOOP: \[\text{[J = 1 To 2]}\]

IF SUBSTR(CNFG_DATA(J),MSG,1,12) EQ 'DATA ENTERED'

THEN

CONFIG(J) = CNFG DATA(J);
CONFIND(J) = CINDEX(J);

ENDLOOP;

END CNFG;
ROUTINE CSSCREEN

IN (CNFG_DATA(I), CNFQRO, PRGARR(I), PRGARR(I), INFORM(I), MIDFLAG(I), RWYEQP(I));

INOUT (CINDEX(I), RSTATUS);
[This routine controls configuration information screen]

CHR PNAME [character variable of length 8 containing name of DMS panel initialized to 'CONF', name of panel that controls configuration information screen]

INT CURSOR [integer variable containing cursor's position on screen]

BITS DM(51) [8 bit variable of data masks used in DMS]

BITS TM(62) [8 bit variable of text masks used in DMS]

STRUCTURE AUX_DATA LIKE CNFG_DATA

ENDSTRUCTURE;

BITS CID [24 bit variable containing current operating configuration]

STRUCTURE CNFGLIST(73)

BITS ID [24 bit configuration ID]

ENDSTRUCTURE;

STRUCTURE MID(5)

BITS NUM [24 bit variable indicating runway in need of coordination with MIDWAY airport]

CHR CHIR [character representation of NUM]

ENDSTRUCTURE;

PERFORM INITIALIZATION;

AUX_DATA = CNFG_DATA(I), BY NAME;
PERFORM SET_UP_SCREEN_POINTERS_(CNFG);

REPEAT UNTIL (RSTATUS NE ENTER);

IF FLAG EQ 'O' B

THEN IF IMPRIM(1).COMP(CINDEX(1)).INDEX LT 999

THEN

PERFORM OUTPUT_SET_UP_(TOTAL);
PERFORM OUTPUT_SET_UP_(MORTH);
PERFORM OUTPUT_SET_UP_(SOUTH);
PERFORM OUTPUT_SET_UP_(BALANCING_ARRIVALS);
PERFORM OUTPUT_SET_UP_(BALANCING_DEPARTUR);
PERFORM OUTPUT_SET_UP_(OTHERS);

ELSE

LOOP; [J = 29 to 62]
TM(J) = FLDARK;
ENDLOOP;

LOOP; [J = 26 to 49]
DM(J) = FLDARK;
ENDLOOP;

CNFG_DATA(I).WMSG2 = (29) 'CONCATENATE 'THIS CONFIGURATION IS INELIGIBLE';

PERFORM DISPLAY_PANEL;

IF RSTATUS EQ PA1

THEN stop;
IF RSTATUS NE ENTER

THEN CNFG_DATA(I) = AUX_DATA, BYNAME;

ELSE

FLAG = '0'B;
TN = FLDDDF;
DM = FLDDDF;
DM(1) = FLDHIGH;
DM(52) = FLDHIGH;

CALL CCHECK;

INOUT (CNFG_DATA(I), CID, CURSOR);

[This routine checks for errors occurred on screen as a result of an erroneous entry and returns value for cursor pointing to first data field where an error has occurred; and an appropriate screen message is issued advising user with corrections]

IF CNFG_DATA(I).MSG NE 'DATA ENTERED'

THEN DM(CURSOR) = FLDHIGH;

ELSE

CALL CVALID;

IN (CNFG_LIST, CID);

INOUT (CNFG_DATA(I), CINDEX(I));

[This routine performs data validation checks on screen entries and returns value for cursor pointing to first invalid data field. Also, an appropriate screen message is issued advising user with corrections]

IF CNFG_DATA(I).MSG NE 'DATA ENTERED'

THEN

FLAG = '1'B;
LOOP: \[ J = 29 \text{ to } 62 \]
\[ TH(J) = FLDDARK; \]
ENDLOOP;

LOOP: \[ J = 26 \text{ to } 50 \]
\[ DH(J) = FLDDARK; \]
ENDLOOP;

ELSE
CALL UPDATE;

INPUT (CMFG_DATA(I));

[This routine performs local updates on screen]

CMFG_DATA(I).MSG = 'DATA ENTERED AT ' CONCATENATE GMT;

AUX_DATA = CMFG_DATA(I), BY NAME;

ENDREPEAT;

END SCREEN;
PROCESS_INITIALIZATION:

[This process performs a number of necessary initializations]

$THREE = 3;
FLAG = '0'B;
DM = FLDFEED;
TM = FLDFEED;
D1(1) = FLDFEED1;
D1(2) = FLDFEED2;
CMFG_LIST = CMFGQ, BY NAME;
CURSOR = 2;

LOOP: [J = 1 to 12]

IF SUBSTR(CMFGQ(CINDEX(I)).ID,J,1) EQ '1'B
  THEN CMFG_DATA(I).CONF.ARR(J) = 'X';
ELSE CMFG_DATA(I).CONF.ARR(J) = (2) '1';
  IF SUBSTR(CINDEX(I)).ID,J+12,1) EQ '1'B;
  THEN CMFG_DATA(I).CONF.DEP(J) = 'X';
  ELSE CMFG_DATA(I).CONF.DEP(J) = (2) '1';
ELSE ENDLOOP;

MIDIND = '11000000000100001001000015';
MID(1).NUM = '1000000000000000000000000000000000000';
MID(2).NUM = '0100000000000000000000000000000000000';
MID(3).NUM = '0000000000000000000000000000000000000';
MID(4).NUM = '0000000000000000000000000000000000000';
MID(5).NUM = '0000000000000000000000000000000000000';
MID(1).CHR = '4R ARR';
MID(2).CHR = '4L ARR';
MID(3).CHR = '32L ARR';
MID(4).CHR = '14R DIP';
MID(5).CHR = '22L DIP';

HIRL(1) = '4R';
HIRL(2) = '4L';
HIRL(3) = '9R';
HIRL(4) = '9L';
HIRL(5) = '14R';
HIRL(6) = '14L';
HIRL(7) = '22R';
HIRL(8) = '22L';
HIRL(9) = '27R';
HIRL(10) = '27L';
HIRL(11) = '32R';
HIRL(12) = '32L';

END INITIALIZATION;
PROCESS SET_UP_SCREEN_POINTERS(CNFG);
   [this process sets up screen pointers for DNS use]
   CONFIG_LOADLIST.TIME = ADDR(CNFG_DATA(I).TIME);
   LOOP;  [J = 1 to 12]
      CONFIG_LOADLIST.COMF.ARR(J) = ADDR(CNFG_DATA(I).CONF.ARR(J));
      CONFIG_LOADLIST.COMF.ARR(J) = ADDR(CNFG_DATA(I).CONF.DEF(J));
   ENDDO;
   CONFIG_LOADLIST.TOTAL.SAT = ADDR(CNFG_DATA(I).TOTAL.SAT);
   CONFIG_LOADLIST.TOTAL.PCT_ARR = ADDR(CNFG_DATA(I).TOTAL.PCT_ARR);
   CONFIG_LOADLIST.TOTAL.ARR.DEM = ADDR(CNFG_DATA(I).TOTAL.ARR.DEM);
   CONFIG_LOADLIST.TOTAL.ARR.CAP = ADDR(CNFG_DATA(I).TOTAL.ARR.CAP);
   CONFIG_LOADLIST.TOTAL.DEP.DEM = ADDR(CNFG_DATA(I).TOTAL.DEP.DEM);
   CONFIG_LOADLIST.TOTAL.DEP.CAP = ADDR(CNFG_DATA(I).TOTAL.DEP.CAP);
   CONFIG_LOADLIST.NORTH.SAT = ADDR(CNFG_DATA(I).NORTH.SAT);
   CONFIG_LOADLIST.NORTH.PCT = ADDR(CNFG_DATA(I).NORTH.PCT);
   CONFIG_LOADLIST.NORTH.ARR = ADDR(CNFG_DATA(I).NORTH.ARR);
   CONFIG_LOADLIST.NORTH.DEP = ADDR(CNFG_DATA(I).NORTH.DEP);
   CONFIG_LOADLIST.SOUTH.SAT = ADDR(CNFG_DATA(I).SOUTH.SAT);
   CONFIG_LOADLIST.SOUTH.PCT = ADDR(CNFG_DATA(I).SOUTH.PCT);
   CONFIG_LOADLIST.SOUTH.ARR = ADDR(CNFG_DATA(I).SOUTH.ARR);
   CONFIG_LOADLIST.SOUTH.DEP = ADDR(CNFG_DATA(I).SOUTH.DEP);
   CONFIG_LOADLIST.BALANCING.ANDEV = ADDR(CNFG_DATA(I).BALANCING.ANDEV);
   CONFIG_LOADLIST.BALANCING.ACOMPLEX = ADDR(CNFG_DATA(I).BALANCING.ACOMPLEX);
   CONFIG_LOADLIST.BALANCING.DMDEV = ADDR(CNFG_DATA(I).BALANCING.DMDEV);
   CONFIG_LOADLIST.BALANCING.DCOMPLEX = ADDR(CNFG_DATA(I).BALANCING.DCOMPLEX);
   CONFIG_LOADLIST.MSG0 = ADDR(CNFG_DATA(I).MSG0);
   CONFIG_LOADLIST.MSG1 = ADDR(CNFG_DATA(I).MSG1);
   CONFIG_LOADLIST.MSG2 = ADDR(CNFG_DATA(I).MSG2);
   CONFIG_LOADLIST.MSG = ADDR(CNFG_DATA(I).MSG);
END SET_UP_SCREEN_POINTERS(CNFG);
PROCESS OUTPUT_SET_UP(TOTAL)

[This process sets up screen variable with total airport information]

IF FRCARR(I).CONF(CINDEX(I)).ENPRINT LT 0.
    THEN
      CHFG_DATA(I).MSG = (29) ' CONCATENATE *** SATURATED ***
      LOOP; [J = 57 to 62]
         TM(J) = FLDDARK;
      ENDLOOP;
      LOOP; [J = 44 to 47]
         DN(J) = FLDDARK;
      ENDLOOP;
    ENDIF;

IF INFORM(I).CONF(CINDEX(I)).SATURATION EQ 0.
    THEN
      CHFG_DATA(I).TOTAL.SAT = '00';
    ELSEIF INFORM(I).CONF(CINDEX(I)).SATURATION LT .095
      THEN
        C = SUBSTR(F(100.*INFORM(I).CONF(CINDEX(I)).SATURATION, $THREE),1,3);
        CHFG_DATA(I).TOTAL.SAT = SUBSTR(C,1,1) CONCATENATE '0' CONCATENATE SUBSTR(C,3,1);
      ELSE
        C = SUBSTR(F(100.*INFORM(I).CONF(CINDEX(I)).SATURATION, $THREE),1,3);
        CHFG_DATA(I).TOTAL.SAT = SUBSTR(C,1,1) CONCATENATE '.' CONCATENATE SUBSTR(C,2,2);
    ENDIF;

IF FRCARR(I).TOTAL + FRCARR(I).TOTDEP EQ 0.
    THEN
      ATOTFRC = .5;
ELSE
ATOTPRC = PRCARR(I).TOTARR/(PRCARR(I).TOTARR + PRCARR(I).TOTDEP);
CNFG_DATA(I).TOTAL.PCT.ARR = SUBSTR(F(100.*ATOTPRC,$THREE),1,3);
ARRCAP = INFORM(I).CONF(CINDEX(I)).NARRCAP + INFORM(I).CONF(CINDEX(I)).SARRCAP;
DEPCAP = INFORM(I).CONF(CINDEX(I)).NDEPCAP + INFORM(I).CONF(CINDEX(I)).SDEPCAP;
BTOTPRC = ARRCAP/(ARRCAP + DEPCAP);
IF ATOTPRC GT BTOTPRC
THEN DEPCAP = (1.0 - ATOTPRC)*ARRCAP/ATOTPRC;
ELSEIF ATOTPRC LT BTOTPRC
THEN ARRCAP = ATOTPRC*DEPCAP/(1.0 - ATOTPRC);
CNFG_DATA(I).TOTAL.ARR.DEM = SUBSTR(F(PRCARR(I).TOTARR, $THREE),1,3);
CNFG_DATA(I).TOTAL.DEP.DEM = SUBSTR(F(PRCARR(I).TOTDEP, $THREE),1,3);
CNFG_DATA(I).TOTAL.ARR.CAP = SUBSTR(F(ARRCAP,$THREE),1,3);
CNFG_DATA(I).TOTAL.DEP.CAP = SUBSTR(F(DEPCAP,$THREE),1,3);
END OUTPUT_SET_UP(TOTAL);
PROCESS OUTPUT_SET_UP (NORTH)

IF INFORM(I).CONF(CINDEX(I)).NSAT EQ 0.

THENIF INFORM(I).CONF(CINDEX(I)).NSAT EQ 0.

THEN CNFG_DATA(I).NORTH.SAT = ' 0.0';

ELSEIF INFORM(I).CONF(CINDEX(I)).NSAT LT .95

THEN

C = SUBSTR(F(100.0 * INFORM(I).CONF(CINDEX(I)).NSAT,$THREE),1,3);

CNFG_DATA(I).NORTH.SAT = SUBSTR(C,1,1) CONCATENATE '.0' CONCATENATE SUBSTR (C,2,1);

ELSE

C = SUBSTR(F(100.0*INFORM(I).CONF(CINDEX(I)).NSAT,$THREE),1,3);

CNFG_DATA(I).NORTH.SAT = SUBSTR(C,1,1) CONCATENATE '.1' CONCATENATE SUBSTR (C,2,2);

ELSE CNFG_DATA(I).NORTH.SAT = (4) ' '

IF PRCAAIR(I).CONF(CINDEX(I)).BMPFINT GT 0.0

THEN

CNFG_DATA(I).NORTH.PCT_ARR = SUBSTR(F(100.*PRCAAIR(I).CONF(CINDEX(I)).BMPFINT,$THREE)),1,3);

CNFG_DATA(I).NORTH.ARE.AIM = SUBSTR(F(PRCAAIR(I).CONF(CINDEX(I)).BMPFINT,$THREE),1,3);

CNFG_DATA(I).NORTH.ARE.CAP = SUBSTR(F(PRCAAIR(I).CONF(CINDEX(I)).BMPFINT,$THREE),1,3);

CNFG_DATA(I).NORTH.DEP.DEM = SUBSTR(F(PRCAAIR(I).CONF(CINDEX(I)).BMPFINT,$THREE),1,3);

CNFG_DATA(I).NORTH.DEP.CAP = SUBSTR(F(PRCAAIR(I).CONF(CINDEX(I)).BMPFINT,$THREE),1,3);

ELSE

CNFG_DATA(I).NORTH.PCT_ARR = SUBSTR(F(100.*PRCAAIR(I).CONF(CINDEX(I)).BMPFINT,$THREE),1,3);

CNFG_DATA(I).NORTH.ARE.AIM = SUBSTR(F(PRCAAIR(I).CONF(CINDEX(I)).BMPFINT,$THREE),1,3);

CNFG_DATA(I).NORTH.ARE.CAP = SUBSTR(F(PRCAAIR(I).CONF(CINDEX(I)).BMPFINT,$THREE),1,3);

CNFG_DATA(I).NORTH.DEP.DEM = SUBSTR(F(PRCAAIR(I).CONF(CINDEX(I)).BMPFINT,$THREE),1,3);

CNFG_DATA(I).NORTH.DEP.CAP = SUBSTR(F(PRCAAIR(I).CONF(CINDEX(I)).BMPFINT,$THREE),1,3);

ENDIF OUTPUT_SET_UP (NORTH);
PROCESS  OUTPUT SET UP (SOUTH)
[This process sets up screen variable with south complex information]
  IF  INFORM(I).CONF(CINDEX(I)).SSAT GE 0.
    THEN IF INFOENM(I).CONF(CINDEX(I)).SSAT EQ 0.
      THEN CNFG_DATA(I).SOUTH.SAT = '0.0';
      ELSEIF INFORM(I).CONF(CINDEX(I)).SSAT LT .095
        THEN
          C = SUBSTR(F(100.0 * INFORM(I).CONF(CINDEX(I)).SSAT,$THREE),1,3);
          CNFG_DATA(I).SOUTH.SAT = SUBSTR(C,1,1) CONCATENATE '.0' CONCATENATE SUBSTR(C,2,2);
        ELSE CNFG_DATA(I).SOUTH.SAT = (4)' ';
    ELSE CNFG_DATA(I).SOUTH.SAT = (4)' ';
  IF PRCARR(I).CONF(CINDEX(I)).BNFRCNT GE 0.0
    THEN
      CNFG_DATA(I).SOUTH.PCT_ARR = SUBSTR(F(100.0 * PRCARR(I).CONF(CINDEX(I)).BSFRCNT,$THREE),1,3);
      CNFG_DATA(I).SOUTH.AIR.DEN = SUBSTR(F(PRCARR(I).CONF(CINDEX(I)).BSRDEN,$THREE),1,3);
      CNFG_DATA(I).SOUTH.AIR.CAP = SUBSTR(F(INFORM(I).CONF(CINDEX(I)).SARBCAP,$THREE),1,3);
      CNFG_DATA(I).SOUTH.DEM.DEN = SUBSTR(F(PRCARR(I).CONF(CINDEX(I)).BSDEPCAP,$THREE),1,3);
      CNFG_DATA(I).SOUTH.DEM.CAP = SUBSTR(F(INFORM(I).CONF(CINDEX(I)).SDEPCAP,$THREE),1,3);
    ELSE
      CNFG_DATA(I).SOUTH.PCT_ARR = SUBSTR(F(100.0 * PRCARR(I).CONF(CINDEX(I)).BSFRCNT,$THREE),1,3);
      CNFG_DATA(I).SOUTH.AIR.DEN = SUBSTR(F(PRCARR(I).CONF(CINDEX(I)).BSRDEN,$THREE),1,3);
      CNFG_DATA(I).SOUTH.AIR.CAP = SUBSTR(F(INFORM(I).CONF(CINDEX(I)).SARBCAP,$THREE),1,3);
      CNFG_DATA(I).SOUTH.DEM.DEN = SUBSTR(F(PRCARR(I).CONF(CINDEX(I)).BSDEPCAP,$THREE),1,3);
      CNFG_DATA(I).SOUTH.DEM.CAP = SUBSTR(F(INFORM(I).CONF(CINDEX(I)).SDEPCAP,$THREE),1,3);
END OUTPUT_SET_UP (SOUTH);
PROCESS  OUTPUT_SET_UP_(BALANCING_ARRIVALS);
 [This process sets up screen variables with arrival demand balancing information]
 IF  PRCARR(I).CONF(CINDEX(I)).BNPRCNT GE 0.
 THEN IF INFORM(I).CONF(CINDEX(I)).CHANGEMARR GT 0
 THEN
 CHANGE = FLOAT(INFORM(I).CONF(CINDEX(I)).CHANGEMARR);
 CNFG_DATA(I).BALANCING.ACOMPLEX = 'SOUTH';
 CNFG_DATA(I).BALANCING.AMOVE = SUBSTR(P(CHANGE,%THREE),1,3);
 ELSEIF INFORM(I).CONF(CINDEX(I)).CHANGEMARR LT 0
 THEN
 CHANGE = FLOAT(INFORM(I).CONF(CINDEX(I)).CHANGEMARR);
 CNFG_DATA(I).BALANCING.ACOMPLEX = 'NORTH';
 CNFG_DATA(I).BALANCING.AMOVE = SUBSTR(P(CHANGE,%THREE),1,3);
 ELSE
 CNFG_DATA(I).BALANCING.ACOMPLEX = (4)';
 CNFG_DATA(I).BALANCING.AMOVE = 'NO';
 TH(58) = FLDDARK;
 TM(59) = FLDDARK;
 ELSE
 TH(57) = FLDDARK;
 TM(58) = FLDDARK;
 TM(59) = FLDDARK;
 DM(44) = FLDDARK;
 DM(45) = FLDDARK;
 END OUTPUT_SET_UP_(BALANCING_ARRIVALS);
PROCESS OUTPUT_SET_UP (BALANCING_DEPARTURES);
   [This process sets up screen variable with departure demand balancing information]
   IF PRCARR(I).CONF(CINDEX(I)).RNPFRONT GE 0.
   THEN IF INFORN(I).CONF(CINDEX(I)).CHANGEDEP GT 0
       THEN
           CHANGE = FLOAT(INFORN(I).CONF(CINDEX(I)).CHANGEDEP);
           CNFG_DATA(I).BALANCING.DCOMPLEX = 'SOUTH';
           CNFG_DATA(I).BALANCING.DMOVE = SUBSTR(F(CHANGE,5,3),1,3);
       ELSE IF INFORN(I).CONF(CINDEX(I)).CHANGEDEP LT 0
           THEN
               CHANGE = FLOAT(INFORN(I).CONF(CINDEX(I)).CHANGEDEP);
               CNFG_DATA(I).BALANCING.DCOMPLEX = 'NORTH';
               CNFG_DATA(I).BALANCING.DMOVE = SUBSTR(F(CHANGE,5,3),1,3);
           ELSE
               CNFG_DATA(I).BALANCING.DCOMPLEX = 'U';
               CNFG_DATA(I).BALANCING.DMOVE = 'NO';
               TM(61) = FLDARK;
               TM(62) = FLDARK;
       ELSE
           TM(60) = FLDARK;
           TM(61) = FLDARK;
           TM(62) = FLDARK;
           DM(46) = FLDARK;
           DM(47) = FLDARK;
   END OUTPUT_SET_UP (BALANCING_DEPARTURES);
PROCESS OUTPUT SET UP (OTHERS)
[This process sets up screen variable with rest of information needed on configuration information screen]

DUMMY1 = "

IF MIDFLG(I) NE (2)

THEN

LOOP; [R = 1 to 5]

IF (MID(R).NUM AND CNFGQ(CINDEX(I)).ID) NE 0

THEN DUMMY1 = DUMMY1 CONCATENATE MID(R).CHR;

ENDLOOP;

IF DUNMY1 EQ "'

THEN CNFGQ(I).WNSG1 = (80) ' ";
ELSE CNFGQ(I).WNSG1 = DUMMY1 CONCATENATE 'COORDINATE WITH MIDWAY TRAFFIC' ;

HIRLND = " ";

LOOP; [N = 1 to 12]

IF RWYEQP(I).RUNWAY(N).HIRL NE (2) ' 

THEN HIRLND = HIRLND CONCATENATE '1';
ELSE HIRLND = HIRLND CONCATENATE '0';

ENDLOOP;

COMBIND = SUBSTR(CNFGQ(CINDEX(I)).ID,1,12) CONCATENATE SUBSTR(CNFGQ(CINDEX(I)).ID,13,12);
COMBIND = COMBIND AND HIRLND;
DUMMY1 = " " ;
LOOP; [M = 1 to 12]
  IF SUBSTR(COMBIND,M,1) EQ '1'B
    THEN DUMMY1 = DUMMY1 CONCATENATE HIRL(M);
  ENDLOOP;
  IF DUMMY1 NE ''
    THEN CNFG_DATA(I).MSG2 = DUMMY1 CONCATENATE 'INELIGIBLE BETWEEN SUNSET & SUNRISE';
    ELSE CNFG_DATA(I).MSG2 = (80)' ';
  END OUTPUT_SET_UP (OTHERS);
ROUTINE CCHECK

INOUT (CHFG_DATA(I), CID, CURSOR)

[This routine checks for errors occurred on screen as a result of an erroneous entry and returns
screen message is issued advising user with corrections]

ERR5 = 'INPUT MUST BE X OR BLANK';
CID = (24) '0' B;
CHFG_DATA(I).MSG = 'DATA ENTERED'
B1 = 'B';
B2 = '1'B;
B3 = (23)'0'B;

REPEAT WHILE (CHFG_DATA(I).MSG EQ 'DATA ENTERED') {J = 1 to 12}

CURSOR = J + 1;

IF X(CHFG_DATA(I).CONF.ARR(J)) NE 0

THEN CHFG_DATA(I).MSG = ERR5;
ELSE

B = B1 CONCATENATE B2 CONCATENATE B3;
B1 = '0'B CONCATENATE B1;
IF CHFG_DATA(I).CONF.ARR(J) EQ 'Xb'

THEN CID = CID OR B;

ENDREPEAT;
REPEAT WHILE (CNFG_DATA(I).MSG EQ 'DATA ENTERED') [J = 1 To 12]
    CURSOR = J + 13;
    IF X(CNFG_DATA(I).CONF.DEP(J)) NE 0
        THEN CNFG_DATA(I).MSG = ERR5;
    ELSE
        B = B1 CONCATENATE B2 CONCATENATE B3;
        B1 = '0'B CONCATENATE B1;
        IF CNFG_DATA(I).CONF.DEP(J) = 'X'
            THEN CID = CID OR B;
    ENDREREPEAT;
    IF CNFG_DATA(I).MSG EQ 'DATA ENTERED'
        THEN CURSOR = 2;
END CCHECK;
ROUTINE CVALID

IN (CNFG_LIST,CID);
INOUT (CNFG_DATA(I),CINDEX(I));

[This routine performs data validation checks on screen entries and returns value for cursor pointing to first invalid data field. Also, an appropriate screen message is issued advising user with corrections]

ERRCNFG = 'THIS CONFIGURATION IS NOT KNOWN';
CNFG_DATA(I).MSG = 'DATA ENTERED';
L = 1.0;
U = 73.0;
AGAIN = 1;
INDEX = 0;

REPEAT WHILE (AGAIN EQ 1);
INDEX = FLOOR ((L+U)/2.0);

IF U LT L
THEN [ID not in list]
AGAIN = 0;
INDEX = 0;
ELSEIF CID GT CNFG_LIST(INDEX).ID
THEN L = INDEX + 1;
ELSEIF CID LT CNFG_LIST(INDEX).ID
THEN U = INDEX - 1;
ELSE AGAIN = 0 [ID is equal to CNFG_LIST(INDEX)]
ENDREPEAT;

IF INDEX EQ 0
THEN CNFG_DATA(I).MSG = ERRCNFG;
ELSE CINDEX(I) = INDEX;
END CVALID;
ROUTINE CUPDATR

1H CNFG&Q(CINDEX(1)),ID);

INOUT CNFG_DATA(I);

[This routine performs local updates on screen]

LOOP: [J = 1 to 12]

IF SUBSTR(CNFG&Q(CINDEX(1)),ID,J,1) = '1' THEN

THEN CNFG_DATA(I).CONF.ARR(J) = 'X';

ELSE CNFG_DATA(I).CONF.ARR(J) = ' ';

ENDIF;

ELSE SUBSTR(CNFG&Q(CINDEX(1)),ID,J + 12, 1) = '1' THEN

THEN CNFG_DATA(I).CONF.DEF(J) = 'X';

ELSE CNFG_DATA(I).CONF.DEF(J) = ' ';

ENDIF;

ENDLOOP;

END CUPDATR;
2.16 Menu and Parameter Screens

The processing for the Menu and Parameter Screens is presented in pages 2-422 to 2-433.
[***LOCAL VARIABLES***]

STRUCTURE FARM DATA LIKE FARM
   [This structure is similar to FARM used as a working area within screen routine]
ENDSTRUCTURE;

STRUCTURE CNVT PEN LIKE CNVT PEN
   [This structure is similar to CNVT PEN used as a working area within screen routine]
ENDSTRUCTURE;

INT SWITCH(2) [This variable is used for switching between current and forecast screens, initialized to (2,1)]

STRUCTURE FARM LOADLIST [A structure of pointers, one for each data field on screen used by panel manager for loading and unloading data to and from screen]

GROUP PARAMETER

GROUP CESS
   PTR ARR [pointer for arrival crosswind threshold data field]
   PTR DFP [pointer for departure crosswind threshold data field]

GROUP TAIL
   PTR ARR [pointer for arrival tailwind threshold data field]
   PTR DFP [pointer for departure tailwind threshold data field]

PTR MSG [pointer for screen message data field]

BITZ FENCE [32 bit variable as prescribed by IMS manual, initialized to string of (32) '1'B]

ENDSTRUCTURE;
STRUCTURE MENULOAD [A structure of pointers, one for each data field on screen used by panel manager for loading and unloading data to and from screen]

PTR TERMINATION [pointer for termination indicator data field]

BITS FENCE [32 bit variable as prescribed by IMS manual, initialized to string of (32) '1'8]

ENDSTRUCTURE;
ROUTINE MENUPRM

INOUT (PARAM, CNVTFRM, ESTATUS);

OUT (TERM);

[This routine invokes menu screen and/or parameter screen]

PARAM_DATA = PARAM;

CNVT_FRM = CNVTFRM;

I = 1;

REPEAT UNTIL (ESTATUS NE FF12);

IF I EQ 2

THEN PARAM_DATA = PARAM

I = SWITCH(I); [switch between two screens]

REPEAT UNTIL (ESTATUS NE FF11);

IF I EQ 1

THEN CALL MSCREEN;

INOUT (ESTATUS);

OUT (TERM);

[This routine controls menu screen]

ELSE CALL PSCREEN;

INOUT (PARAM_DATA, CNVT_FRM, ESTATUS);

[This routine controls parameter screen]

ENDREPEAT;

ENDREPEAT:
IF SUBSTR(PARM_DATA.MEG,1,12) EQ 'DATA ENTERED'
    THEN
        PARAM = PARM DATA;
        CNTYPRM = CNTY_PRM;
    END MENUPERM;
ROUTINE MSCREEN

INOUT (RSTATUS);

OUT (TERM);

CHR PNAME [character variable of length 8 containing name of DNS panel initialized to 'HELP', name of panel that controls menu screen]

INT CURSOR [integer variable containing cursor's position on screen]

MENULOAD.TERMINATION = ADDR(TERM); [set up screen pointer]

CURSOR = 1;

REPEAT UNTIL ((RSTATUS NE ENTER) OR (TERM EQ 'X '));

TERM = ' ';

PERFORM DISPLAY_PANEL;

IF RSTATUS EQ P11

THEN stop;

ENDREPEAT;

END MSCREEN;
ROUTINE PSCREEN

INOUT (PAM_DATA, CNVRT_PARM, RSTATUS);

[This routine controls parameter screen]

CHR PNAME    [character variable of length 8 containing name of DNS panel initialized to 'PAMOPT',
               name of panel that controls parameter screen]

INT CURSOR   [integer variable containing cursor's position on screen]

BITS DM(5)   [8 bit variable of data masks used in DNS]

STRUCTURE AUX_DATA LIKE PARM_DATA

ENDSTRUCTURE;

CURSOR - 1;  [set cursor to position 1; first data field on screen]

DM = FLDDEF; [set data fields to default intensity (normal)]

DM(5) = FLDHIGH; [set last data field to high intensity]

AUX_DATA = PARM_DATA;

PERFORM SET_UP_SCREEN_POINTERS (PARM);

REPEAT UNTIL (RSTATUS NE ENTER);

PERFORM DISPLAY_PANEL;

IF RSTATUS EQ PA1
    THEN STOP;

IF RSTATUS NE ENTER
    THEN PAM_DATA = AUX_DATA;
ELSE
    DM = FLDDEF;
    DM(5) = FLDHIGH;
CALL PCHECK;

INOUT (FARM_DATA, CNVRT, FERM, CURSOR);
[This routine checks for errors occurred on screen as a result of erroneous entry and returns value for cursor pointing to first data field where an error has occurred, and an appropriate screen message is issued advising user with corrections]

IF FARM_DATA.MSG NE 'DATA ENTERED'
   THEN DM(CURSOR) = FLMHIGH;
ELSE
   CALL PVALID;
   INOUT (FARM_DATA, CNVRT, FERM, CURSOR);
   [This routine performs data validation checks on screen entries and returns value for cursor pointing to first invalid data field. Also, an appropriate screen message is issued advising user with corrections]

IF FARM_DATA.MSG NE 'DATA ENTERED'
   THEN DM(CURSOR) = FLMHIGH;
ELSE
   FARM_DATA.MSG = 'DATA ENTERED AT ' CONCATURATE GNT;
   AUX_DATA = FARM_DATA;
ENDREPEAT;
END PSCREEN;
PROCESS SET UP SCREEN POINTERS (PARM)

(This process sets up screen pointers for IMS use)

PARM_LOADLIST_PARAMETER.CRSS.ARR = ADDR(PARM_DATA_PARAMETER.CRSS.ARR);
PARM_LOADLIST_PARAMETER.TAIL.ARR = ADDR(PARM_DATA_PARAMETER.TAIL.ARR);
PARM_LOADLIST_PARAMETER.CRSS.DEF = ADDR(PARM_DATA_PARAMETER.CRSS.DEF);
PARM_LOADLIST_PARAMETER.TAIL.DEF = ADDR(PARM_DATA_PARAMETER.TAIL.DEF);
PARM_LOADLIST.MSG = ADDR(PARM_DATA.MSG);

END SET UP SCREEN POINTERS (PARM)
ROUTINE PCHECK

INPUT (PARM_DATA, CNVRT_FNM, CURSOR);

[This routine checks for errors occurred on screen as a result of erroneous entry and returns value for cursor pointing to first data field where an error has occurred, and an appropriate screen message is issued advising user with corrections]

ERR1 = 'NUMERIC INPUT REQUIRED';

ERR2 = 'NON_NEGATIVE INPUT REQUIRED';

PARM_DATA.MSG = 'DATA ENTERED';

ON CONVERSION BEGIN [ON CONVERSION is a PL/I feature that is invoked if a character data is detected in a numerical data field]

PARM_DATA.MSG = ERR1;

RETURN;

CURSOR = 1;

Get STRING (PARM_DATA.PARAMETER.ARR.CRSS) EDIT (CNVRT_FNM.ARR.CRSS);

IF VERIFY('-', PARM_DATA.PARAMETER.ARR.CRSS) EQ 0 THEN PARM_DATA.MSG = ERR2;

ELSE

CURSOR = 2;

Get STRING (PARM_DATA.PARAMETER.DEF.CRSS) EDIT (CNVRT_FNM.DEF.CRSS);

IF VERIFY ('-', PARM_DATA.PARAMETER.DEF.CRSS) EQ 0 THEN PARM_DATA.MSG = ERR2;
ELSE
  CURSOR = 3;
  Get STRING (PARM_DATA.PARMETER.ARR.TAIL) EDIT (CNVRT_FNM.ARR.TAIL);
  IF VERIFY ('-', PARM_DATA.PARMETER.ARR.TAIL) EQ 0
    THEN PARM_DATA.MSC = ERR2;
  ELSE
    CURSOR = 4;
    Get STRING (PARM_DATA.PARMETER.DEP.TAIL) EDIT (CNVRT_FNM.DEP.TAIL);
    IF VERIFY ('-', PARM_DATA.PARMETER.DEP.TAIL) EQ 0
      THEN PARM_DATA.MSC = ERR2;
    ELSE CURSOR = 1;
ENDIF

END PCHECK;
ROUTINE PVALID

INOUT (PARM_DATA, CNVRT_PRN, CURSOR);

[This routine performs data validation checks on screen entries and returns value for cursor pointing to first invalid data field. Also, an appropriate screen message is issued advising user with corrections]

$THREE = 3;

THREE1 = 'CROSSWIND THRESHOLD MUST NOT EXCEED 50 KNOTS';
THREE2 = 'TAILWIND THRESHOLD MUST NOT EXCEED 50 KNOTS';
CURSOR = 1;

IF CNVRT_PRN.ARR.CRSS GT 50.
    THEN PARM_DATA.MSG = THREE1;
ELSEIF CNVRT_PRN.ARR.CRSS EQ 0.
    THEN PARM_DATA.PARAMETER.ARR.CRSS = ' 0.0';
ELSE
    C = SUBSTR(P(CNVRT_PRN.ARR.CRSS*10.0,$THREE),1,3);
    PARM_DATA.PARAMETER.ARR.CRSS = SUBSTR(C,1,2) CONCATENATE '.' CONCATENATE SUBSTR(C,3,1);
    CURSOR = 2;

IF CNVRT_PRN.DEF.CRSS GT 50.
    THEN PARM_DATA.MSG = THREE1;
ELSEIF CNVRT_PRN.DEF.CRSS EQ 0.
    THEN PARM_DATA.PARAMETER.DEF.CRSS = ' 0.0';
ELSE
    C = SUBSTR(P(CNVRT_PRN.DEF.CRSS*10.0,$THREE),1,3);
PARM DATA.PAAMETER.DEF.CRSS = SUBSTR(C,1,2) CONCATENATE ' ' CONCATENATE SUBSTR(C,3,1);
CURSOR = 3;

IF CHVFT.PEM.ARR.TAIL GT 50.
THEN PARM_DATA.MSG = THERF2;
ELSEIF CHVFT.PEM.ARR.TAIL EQ 0.
THEN PARM_DATA.PARAMETER.ARR.TAIL = ' 0.0';
ELSE

CURSOR = 4;
IF CHVFT.PEM.DEF.TAIL GT 50.
THEN PARM_DATA.MSG = THERF2;
ELSEIF CHVFT.PEM.DEF.TAIL EQ 0.
THEN PARM_DATA.PARAMETER.DEF.TAIL = ' 0.0';
ELSE

CURSOR = 1;
END FVALID;

C = SUBSTR(P(CHVFT.PEM.DEF.TAIL*10.0,$THERE),1,3);
PARM_DATA.PARAMETER.DEF.TAIL = SUBSTR(C,1,2) CONCATENATE ' ' CONCATENATE SUBSTR(C,3,1);