

AD-A127 828

SOFTWARE DESCRIPTION FOR THE O'HARE RUNWAY
CONFIGURATION MANAGEMENT SYSTEM. (U) MITRE CORP MCLEAN
VA METREK DIV 5 KAYDOSSI OCT 82 MTR-82W125-VOL-2
FAA-EM-82-28-VOL-2 DTFA01-81-C-10003 F/G 17/7

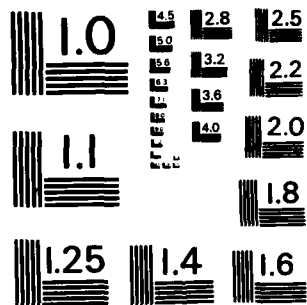
1/5

UNCLASSIFIED

F/G 17/7

NÌ

[illegible]



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

- AD A 127 828 -

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



FILE COPY

88 05 02 030

AD A 127 828

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.

Technical Report Documentation Page

1. Report No. FAA-EM-82-28 Volume II		2. Government Accession No. <i>A127828</i>		3. Recipient's Catalog No.	
4. Title and Subtitle Software Description for the O'Hare Runway Configuration Management System Volume II: Low Level Pseudocode				5. Report Date October 1982	
				6. Performing Organization Code	
7. Author(s) Sadegh Kavoussi				8. Performing Organization Report No. MTR-82W125 Volume II	
9. Performing Organization Name and Address The MITRE Corporation Metrek Division 1820 Dolley Madison Blvd. McLean, Virginia 22102				10. Work Unit No. (TRAIS)	
				11. Contract or Grant No. DTFA01-81-C-10003	
12. Sponsoring Agency Name and Address Department of Transportation Federal Aviation Administration Office of Systems Engineering Management Washington, D. C. 20591				13. Type of Report and Period Covered	
				14. Sponsoring Agency Code AEM-100	
15. Supplementary Notes					
16. 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.					
17. Key Words			18. Distribution Statement Document is available to the public through the National Technical Information Service, Springfield, VA 22161		
19. Security Classif. (of this report) Unclassified		20. Security Classif. (of this page) Unclassified		21. No. of Pages	
				22. Price	

ACKNOWLEDGEMENT

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



A

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 data base which contains all information on O'Hare status over the planning horizon. When CMS is implemented at

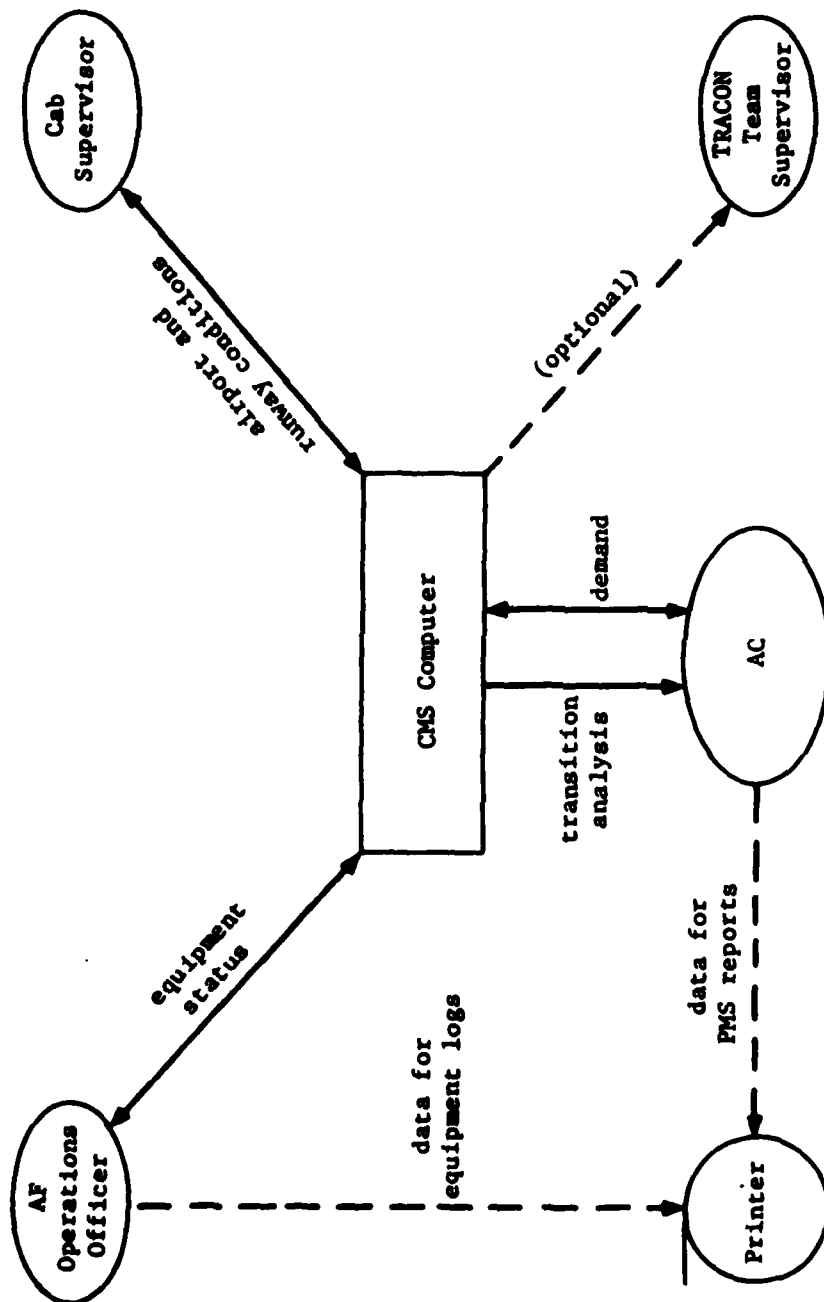


FIGURE A
PHYSICAL CONFIGURATION OF CMS

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.

RWY	EQUIPMENT	OTS	RTS	REMARKS
4L	ALS	1500	1600	REPAIRS

DATA STORED AT 1447

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

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

FORWARDING PAGE BLANK-NOT FILLED

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.

2. CMS SOFTWARE LOGIC (LOW LEVEL PSEUDOCODES)

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]

[***CONFIGURATION REQUIREMENTS***]

STRUCTURE CNPGRQ (73)

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

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

CHR DEP_RWY (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 arrival fixes, 1-KUBBS, 2-PLANT, 3-CGT, 4-VAINS, 5-FARM, 6-MILWAUKEE; contents of array holds numbers indicating arrival runways, 1-4R, 2-4L, ..., 12-32L]

INT DEP(5) [integers indicating fix to runway assignments for a particular configuration. Indices 1 through 5 signify departure fixes, 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;

[***OFFICIAL AIRLINE GUIDE DEMAND INFORMATION***]

STRUCTURE OAGDEM

GROUP TABLE (0.23)

CHR GMT [character representation (length 4) of times associated with demand information given in Greenwich Mean Time]

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

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

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

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

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

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

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

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

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

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

ENDSTRUCTURE;

[***CAPACITY DATA***]

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

[***DEPENDENCE MATRIX DATA***]

STRUCTURE DEPMAT(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 FIXTRAV (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 RWYMIN (12) [for each of 12 runways]

GROUP CAT II [pertaining to CAT II]

GROUP NONE

FLT CEIL [ceiling minima with CATII operable]

FLT VIS [visibility minima with CATII operable]

GROUP ILS [pertaining to ILS]

GROUP NONE

FLT CEIL [ceiling minima with both localizer and glide slope operable]

FLT VIS [visibility minima with both localizer and glide slope operable]

GROUP MM [pertaining to middle marker]

FLT CEIL [ceiling minima with both localizer and glide slope operable and middle marker inoperable]

FLT VIS [visibility minima with both localizer and glide slope operable and middle marker inoperable]

GROUP RAIL_ALS [pertaining to RAIL and ALS]

FLT CEIL [ceiling minima with both localizer and glide slope operable, RAIL and ALS inoperable]

FLT 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]

CHR VEL [character representation (length 2) of runway wind velocity]

CHR CRSS [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 RVR [runway's RVR 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]

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

GROUP RUNWAY (12) [12 runways]

CHR CAT II [character indicator (length 2) 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 OM [character indicator (length 2) for status of outer marker]

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

CHR IM [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 HIRL [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]

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

ENDSTRUCTURE;

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

CHR MKE_A [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;

[DEMAND PROFILE DATA FOR SCREEN USE]

STRUCTURE CNVTDEM (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 FARMH [arrival demand at fix FARMH]

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 CONF

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

CHR DEM [character representation (length 3) of departure demand for airport's north complex]

CHR CAP [character representation (length 3) of departure capacity for airport's north complex]

GROUP SOUTH

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

CHR SAT [character representation (length 4) of saturation level for airport's south complex]

GROUP ARR

CHR DEM [character representation (length 3) of arrival demand for airport's south complex]

CHR CAP [character representation (length 3) of arrival capacity for airport's south complex]

GROUP DEP

CHR DEM [character representation (length 3) of departure demand for airport's south complex]

CHR CAP [character representation (length 3) of departure capacity for airport's south complex]

GROUP BALANCING [demand balancing information]

CHR AMOVE [character representation (length 3) of number of arrivals moved from one complex to other for purpose of demand balancing]

CHR ACOMPLEX [character representation complex to which arrivals are moved]

CHR DMOVE [character representation (length 3) of number of departures moved from one complex to other for purpose of demand balancing]

CHR DCOMPLEX [character representation (length 5) of complex to which departures are moved]

CHR WMSG0 [character field (length 80) reserved for user warning messages]

CHR WMSG1 [character field (length 80) reserved for user warning messages]

CHR WMSG2 [character field (length 80) reserved for user warning messages]

CHR MSG [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 CONFLST(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 CONFIG (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 of remarks 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 TRANLIST

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, 4R, up to 3 arrival runways]

CHR DEP(4) [character representation (length 3) of forecast configuration's departure 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 RWY [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 WXLOG

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]

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

ENDSTRUCTURE;

[WEATHER AND WIND PLANNING LOG DATA FOR SCREEN USE]

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 SURFLOGGROUP TABLE (13) [up to 13 log entries]CHR TIME [character representation (length 4) of time used in weather and wind planning log]CHR RWY [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 CNVTSRFGROUP 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 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 arrival demand at fix VAINS]

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

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]

STRUCTURE CNVTOAG

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 GWT [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 CGT [floating point value of hourly arrival demand at fix CGT]

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

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

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 CNVTPRM

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 CNVTQLN (4) [integer representing length of departure queues]

[O'HARE STATUS INFORMATION FOR SCREEN USE]

STRUCTURE OHSTAT

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 OLDMES

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 PARMOPT [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 times 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 TRANLIST [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 PRCARR (2) [2 environments: current and forecast]

FLT TOTARR [total arrival demand]

FLT TOTDEP [total departure demand]

GROUP CONF (73) [73 configurations]

FLT NPRCNT [north complex percentage of arrivals]

FLT SPRCNT [south complex percentage of arrivals]

FLT NARDEM [north complex arrival demand]

FLT SARDEM [south complex arrival demand]

FLT NDEPDEM [north complex departure demand]

FLT SDEPDEM [south complex departure demand]

FLT BNPRCNT [north complex percentage of arrivals after demand balancing]

FLT BSPRCNT [south complex percentage of arrivals after demand balancing]

FLT BNARDEM [north complex balanced arrival demand]

FLT BSARDEM [south complex balanced arrival demand]

FLT BNDEPDEM [north complex balanced departure demand]

FLT BSDEPDEM [south complex balanced departure demand]

ENDSTRUCTURE;

[AIRPORT INFORMATION DATA]

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

GROUP CONF (73) [up to 73 configurations]

INT INDEX [index associated with each configuration for of table look-up]

FLT CAPACITY [total configuration capacity]

FLT NARRCAP [arrival capacity of north complex]

FLT NDEPCAP [departure capacity of north complex]

FLT SARRCAP [arrival capacity of south complex]

FLT SDEPCAP [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 CHANGENDEP [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 ELGLTY (2) [2 environments: current, forecast]

BITS ID [73 bits string signifying eligibility, 0-eligible, 1-ineligible]

INT NUM [number of eligible configurations]

ENDSTRUCTURE;

[CONDITION DATA]

INT CNDTN (2) {1-VPR, 2-IPR}

[AIRPORT STATUS INFORMATION MOST CURRENT FROM DATA BASE]

STRUCTURE ANOW (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]

CNR MNOW (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 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 MOST CURRENT
FROM DATA BASE]

STRUCTURE RNOW (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 OM [character indicator (length 2) for status of outer marker]
CHR MM [character indicator (length 2) for status of middle marker]
CHR IM [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 HIRL [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 DATA BASE]

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

CHR MKE_A [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]

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 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 INDEX MOST CURRENT FROM DATA BASE]

INT CNOW (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 RWY [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 WNOW

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;

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

STRUCTURE CVTWNOW

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 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 RWY [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 FARMH [character representation (length 3) of hourly arrival demand at fix FARMH]
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;

[DEMAND PLANNING LOG DATA MOST CURRENT FROM DATA BASE]

STRUCTURE CVTGNOW

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

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

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;

[DEPARTURE QUEUES INFORMATION MOST CURRENT FROM DATA BASE]

STRUCTURE QNOW

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

ENDSTRUCTURE;

[DEPARTURE QUEUES DATA MOST CURRENT FROM DATA BASE]

INT CVTQNOW (4) [integer representing length of departure queues]

[AIRPORT STATUS INFORMATION ORIGINAL FROM DATA BASE]

STRUCTURE ABEGIN (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 bit 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 RBEGIN (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 EVR [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 DBEGIN (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 at fix PLANT]

CHR CGT [4 bit character representation of arrival demand at fix CGT]

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 MKE_A [4 bit character representation of arrival demand at fix MILWAUKEE]

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 CVIDBCN (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 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 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 rvy [3 bit character representation of a runway used in log entry, e.g., 14R]

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 CVTEBGN

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 ORIGINAL FROM DATA BASE]

STRUCTURE WBEGIN

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 CWTWBCN

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 SBEGIN

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 CVTSBGN

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 GBEGIN

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 FARMH [3 bit character representation of hourly arrival demand at fix FARMH]
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]

END\$STRUCTURE;

[DEMAND PLANNING LOG DATA ORIGINAL FROM DATA BASE]

STRUCTURE CVTGBGN

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 CGT [floating point value of hourly arrival demand at fix CGT]
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]

ENDSTRUCTURE;

[PARAMETERS INFORMATION ORIGINAL FROM DATA BASE]

STRUCTURE PBEGINGROUP 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 DEP [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 CVTPBCNGROUP ARR [wind thresholds for arrivals]FLT CRSS [crosswind component of wind thresholds]FLT TAIL [tailwind component of wind thresholds]GROUP DEP [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 CVTQBN (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 CMS]

BITS RSTATUS [8 bit string containing current program status intialized to '01111011'B or equivalent for PFI1 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 data base, read data base]

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

CALL CHOOSE;
INOUT (RSTATUS, OLD_STATUS,I); [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 data base]

CALL WRITER; [write new central data base]

CALL TODTACH; [release central data base]

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 RNWYMIN file;

CALL MILES;

INOUT(RWYMIN); [convert RVR readings to miles]

Read CNFGRQ from CNFGREQ file;

Read CAPFILE from CAPACTY file;

Read FIXTRAV from TRAVEL file;

Read DEPMAT from DEPEND file;

Read OAGDEM from OAGDMND 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;ENDLOOP;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.MM.VIS = M(RWYMIN(J).ILS.MM.VIS);

```
RWYMIN(J).ILS.RAIL_ALS.VIS = M(RWYMIN(J).ILS.RAIL_ALS.VIS);  
RWYMIN(J).ILS.TDZ.VIS = M(RWYMIN(J).ILS.TDZ.VIS);  
RWYMIN(J).ILS.CL.VIS = M(RWYMIN(J).ILS.CL.VIS);  
RWYMIN(J).LOC.NONE.VIS = M(RWYMIN(J).LOC.NONE.VIS);  
RWYMIN(J).LOC.MM.VIS = M(RWYMIN(J).LOC.MM.VIS);  
RWYMIN(J).LOC.RAIL.VIS = M(RWYMIN(J).LOC.RAIL.VIS);  
RWYMIN(J).LOC.ALS.VIS = M(RWYMIN(J).LOC.ALS.VIS);  
RWYMIN(J).NDB_VOR.NONE.VIS = M(RWYMIN(J).NDB_VOR.NONE.VIS);  
RWYMIN(J).NDB_VOR.RAIL.VIS = M(RWYMIN(J).NDB_VOR.RAIL.VIS);  
RWYMIN(J).NDB_VOR.ALS.VIS = M(RWYMIN(J).NDB_VOR.ALS.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, CVTPNOW;

Read ANOW;

Read CVTANOW;

Read RNOW;

Read DNOW;

Read CVTDNOW;

Read ENOW;

Read CVTENOW;

Read WNOW;

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 = MNOW;
PBEGIN = PNOW;
CVTPBGN = CVTPNOW;
ABEGIN = ANOW;
CVTABGN = CVTANOW;
RBEGIN = RNOW;
DBEGIN = DNOW;
CVTDBGN = CVTDNOW;
CBEGIN = CNOW;
EBEGIN = ENOW;
CVTEBGN = CVTENOW;
QBEGIN = QNOW;
CVTQBG = CVTQNOW;
WBEGIN = WNOW;
CVTWBGN = CVTWNOW;
SBEGIN = SNOW;
CVTSBGN = CVTSNOW;
GBEGIN = GNOW;
CVTGBGN = CVTGNOW;

END SET_ORIGINAL_PROGRAM_VARIABLES;

PROCESS SET_CMS_PROGRAM_VARIABLES

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

MIDFLAG = MBEGIN;
PARAM = PBEGIN, BY NAME;
CNVTPRM = CVTPBCN, BY NAME; [BY NAME OPTION chooses only those variables each structure that have
similar names]
APTSTAT = ABEGIN, BY NAME;
CNVTAPT = CVTABGN, BY NAME;
RWYEQP = RBEGIN, BY NAME;
DEMAND = DBEGIN, BY NAME;
CNVTDEM = CVTDBGN, BY NAME;
CONFIND = CBEGIN;
EQPLOG = EBEGIN, BY NAME;
CNVTEQP = CVTEBGN;
QUELEN = QBEGIN, BY NAME;
CNVTQLN = CVTQBGN;
WXLOG = WBEGIN, BY NAME;
CNVTWX = CVTWBGN;
SURFLOG = SBEGIN, BY NAME;
CNVTSRF = CVTSBGN;
OAGLOG = GBEGIN, BY NAME;
CNVTOAG = CVTGBGN, 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 CMS user]

```

INOUT (RSTATUS, OLD_STATUS, 1);
  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 HSTAT;
    IN (OHSTAT, APTSTAT(1), INFORM(1), CNFGRQ(CONFIND(1)), CONFIND(1), EQPLOG, CNVTEQP, WXLOG,
        CNVTWX, SURFLOG, CNVTSRF);
    INOUT (OLDMES, RSTATUS); [O'Hare status screen]
  ELSEIF RSTATUS EQ PF2;
    THEN CALL LOGS;
    INOUT (RSTATUS); [log selection screen]
  ELSEIF RSTATUS EQ PF13;
    THEN CALL WLOG;
    INOUT (WXLOG, CNVTWX, RSTATUS); [wind & weather planning log screen];

```

```
ELSEIF RSTATUS EQ PF14;  
  THEN CALL SLOG;  
    INOUT (SURFLOG, CVTSRF, RSTATUS); [airport planning log screen]  
ELSEIF RSTATUS EQ PF15  
  THEN CALL ELOG;  
    INOUT (EQPLOG, CVTEQP, RSTATUS); [equipment planning log screen]  
ELSEIF RSTATUS EQ PF16;  
  THEN CALL GLOG;  
    IN (OAGDEM)  
    INOUT (OAGLOG, CVTOAG, RSTATUS); [demand planning log screen]  
ELSEIF RSTATUS EQ PF3;  
  THEN CALL ARPT;  
    IN (CVTPRM)  
    INOUT (APTSTAT, MIDFLAG, CVTIPT, RSTATUS, I); [airport status  
    screen (current, forecast)]  
ELSEIF RSTATUS EQ PF4  
  THEN CALL RWY;  
    INOUT (RWYEQP, RSTATUS, I); [equipment status screen  
    (current, forecast)]  
ELSEIF RSTATUS EQ PF5;  
  THEN CALL DMND;
```


IN (CNVTOAG)

INOUT (DEMAND, CNVTDEM, RSTATUS, I); [demand
profile screen]

ELSEIF RSTATUS EQ PF6

THEN CALL ORDER;

IN (PRCARR, INFORM, CNFGRQ, RWYEQP, MIDFLAG,
CONFLST, CONFIND);

INOUT (RSTATUS, I): [ordered list of
configurations screen (current, forecast)]

ELSEIF RSTATUS EQ PF7

THEN CALL QUEUE;

INOUT (QUELEN, CNVTQLM, RSTATUS);
[current queue length screen]

ELSEIF RSTATUS EQ PF8;

THEN CALL TSETUP;

IN (PRCARR, CNFGRQ, CNVTDEM,
DEPMAT, FIXTRAV, TRANLST,
INFORM, CONFIND, CNDTN,
ELGMLTY, CNVTQLM, QUELEN);

INOUT (RSTATUS); [ordered list of
transitions screen]

ELSEIF RSTATUS EQ PF9

THEN CALL CMFG;

END SCREEN_SELECTION;

IN (CONFIG, CNFGREQ, CONFIND,
PRCARR, INFORM, MIDFLAG,
RWYEQP);

INOUT (RSTATUS, I);
[configuration information
screen (current, forecast)]

ELSEIF RSTATUS EQ PF10

THEN OLD_STATUS = PF11;

ELSE CALL MENUPRM;

INOUT (PARAM,
CNVTFRM, RSTATUS);

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

```

ROUTINE TOLINK
  [This routine establishes a link to data base]

  INT IRETCD [contains return code]

  CALL COMMD; [try to link to data base]

  IN (' CP LINK K11576B 191 197 M CMS #');

  OUT (IRETCD);
  [this system routine issues a CP or CMS message]

  IF (IRETCD 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 COMMD;

    IN (' CP SLEEP 5 SEC#');

    OUT (IRETCD);

    CALL COMMD;

    IN (' CP LINK K11576B 191 197 M CMS #');

    OUT (IRETCD);

  ENOREPEAT;

  CA L COMMD; [once linked, access]

  IN (' ACC 197 B #');

  OUT (IRETCD);

END TOLINK;

```

ROUTINE TODTACH

[This routine detaches user from data base]

INT IRETCD [contains return code]CALL COMMD; {detach and release data base}IN (' CP DET 197 #');OUT (IRETCD);CALL COMMD;IN (' REL 197 #');OUT (IRETCD);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, CVTPNOW);

Write ANOW;

Write CVTANOW;

Write RNOW;

Write DNOW;

Write CVTDNOW;

Write ENOW;

Write CVTENOW;

Write WNOW;

Write CVTWNOW;

Write SNOW;

Write CVTSNOW;

Write GNOW;

Write CVTGNOW;

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 PF1) AND (SUBSTR(OHSTAT.MSG, 1, 15) NE DATA_STORED)

THEN

STORED.OHSTATUS = GMT; [time update]

ELSEIF (OLD_STATUS EQ PF13) AND (SUBSTR(WXLOG.MSG, 1, 15) NE DATA_STORED)

THENCALL WGLOBAL;

INOUT (WXLOG, CNVTWX, WBEGIN, CVTWBGN, WNOW, CVTWNOW);

[This routine reconciles different versions of wind and weather planning log information]

STORED.APLOG1 = GMT; [time updates]

STORED.OHSTATUS = STORED.APLOG1;

ELSEIF (OLD_STATUS EQ PF14) AND (SUBSTR(SURFLOG.MSG, 1, 15) NE DATA_STORED)

THENCALL SGLOBAL;

INOUT (SURFLOG, CNVTSRF, SBEGIN, CVTSBGN, SNOW, CVTSNOW);

[This routine reconciles different versions of airport planning log information]

STORED.APLOG2 = GMT; [time updates]

STORED.OHSTATUS = STORED.APLOG2;

ELSEIF (OLD_STATUS EQ PF15) AND (SUBSTR(EQPLOG.MSG, 1, 15) NE DATA_STORED)

THENCALL EGLOBAL;

INOUT (EQPLOG, CNVTEQP, EBEGIN, CVTEBGN, ENOW, CVTENOW);

[This routine reconciles different versions of equipment planning log information]

```

        STORED.RWYLOG = GMT;
        STORED.OHSTATUS = STORED.RWYLOG; [time updates]
    ELSEIF (OLD_STATUS EQ PF16) AND (SUBSTR(OAGLOG.MSG, 1, 15) NE DATA_STORED)
        THENCALL GGLOBAL;
        INOUT (OAGLOG, CNVTOAG, GBEGIN, CVTGBGN, GNOW, CVTGNOW);
        [This routine reconciles different versions of demand planning
         log information]
        STORED.DMNDLOG = 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), CNVTAPT(J), MIDFLAG(J),
                        ABEGIN(J), CVTABGN(J), MBEGIN(J), ANOW(J),
                        CVTANOW(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.CONF(J) = STORED.AIRPORT(J);
                    STORED.TRANLIST = STORED.AIRPORT(J);
                ENDLOOP;
            ELSEIF OLD_STATUS EQ PF4
                THEN

```

LOOP; [J = 1 to 2]

IF SUBSTR(RWYEQP(J).MSG, 1, 15) NE DATA_STORED

THENCALL RGLOBAL;

INOUT (RWYEQP(J), RBEGIN(J), RNOW(J));
 [This routine reconciles different
 versions of equipment status equipment]

STORED.RUNWAY(J) = GMT; [time updates]
 STORED.AIRPORT(J) = STORED.RUNWAY(J);
 STORED.OLIST(J) = STORED.RUNWAY(J);
 STORED.CONF(J) = STORED.RUNWAY(J);
 STORED.TRANLIST = 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

THENCALL DGLOBAL;

INOUT (DEMAND(J), CNVTDEM(J),
 DBEGIN(J), CVTDBGN(J), DNOW(J),
 CVTDNOW(J));
 [This routine reconciles different
 versions of demand profile
 information]

STORED.DEMAND(J) = GMT; [time updates]
 STORED.OLIST(J) = STORED.DEMAND(J);
 STORED.CONF(J) = STORED.DEMAND(J);
 STORED.TRANLIST = STORED.DEMAND(J);

ENDLOOP;

ELSEIF OLD_STATUS EQ PF6

THEN

IF SUBSTR(CONFLST(1).MSG, 1, 15) NE DATA_STORED

THENCALL CGLOBAL;

INOUT (CONFIND(1), CBEGIN(1), CNOW(1),
QNOW, CVTQNOW, \$ONE);

[This routine reconciles different
versions of operating
configuration information]

STORED.OLIST(1) = GMT; [Time updates]

STORED.CONF(1) = STORED.OLIST(1);

STORED.QLENGTH = STORED.OLIST(1);

STORED.TRANLIST = STORED.OLIST(1);

IF SUBSTR(CONFLST(2).MSG, 1, 15) NE DATA_STORED

THENCALL CGLOBAL;

INOUT (CONFIND(2), CBEGIN(2), CNOW(2),
QNOW, CVTQNOW, \$TWO);

STORED.OLIST(2) = GMT; [Time updates]

STORED.CONF(2) = STORED.OLIST(2);

ELSEIF (OLD_STATUS EQ PF7) AND SUBSTR(QUELEN.MSG, 1,
15) NE DATA_STORED

THENCALL QGLOBAL;

INOUT (QUELEN, CNVTQIN, QBEGIN, CVTQBGN,
QNOW, CVTQNOW);

[This routine reconciles different
versions of current departure queue
information]

STORED.QLENGTH = GMT; [Time updates]

STORED.TRANLIST = STORED.QLENGTH;

ELSEIF OLD_STATUS EQ PF8

THEN STORED.TRANLIST = GMT; [Time update]

ELSEIF OLD_STATUS EQ PF9

THEN

IF SUBSTR(CONFIG(1).MSG, 1, 15) NE DATA
STORED

THENCALL CGLOBAL;

INOUT (CONFIND(1), CBEGIN(1),
CNOW(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 CGLOBAL;

INOUT (CONFIND(2),
CBEGIN(2), CNOW(2),
QNOW, CVTQNOW, \$TWO);

STORED.CONF(2) = GMT

[time updates]

STORED.OLIST(2) = STORED.

CONF(2);

ELSEIF (OLD_STATUS EQ PF11) AND (SUBSTR
(PARAM.MSG, 1, 15) NE DATA
STORED

2-78

END MERGE;

THENCALL PGLOBAL;
INOUT (PARAM, CNVTPRM,
PBEGIN, CVTPBGN, PNOW,
CVTPNOW);
[This routine reconciles
different versions of
parameters information]

STORED.PARMOPT = GMT; [time
updates]
STORED.AIRPORT(1) = STORED.
PARMOPT;
STORED.AIRPORT(2) = STORED.
PARMOPT;
STORED.OLIST(1) = STORED.
PARMOPT;
STORED.OLIST(2) = STORED.
PARMOPT;
STORED.CONF(1) = STORED.
PARMOPT;
STORED.CONF(2) = STORED.
PARMOPT;
STORED.TRAMLIST = STORED.
PARMOPT;

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]

APTSTAT(1).MSG = DATA STORED CONCATENATE STORED.AIRPORT(1);
 APTSTAT(2).MSG = DATA STORED CONCATENATE STORED.AIRPORT(2);
 RWYEQP(1).MSG = DATA STORED CONCATENATE STORED.RUNWAY(1);
 RWYEQP(2).MSG = DATA STORED CONCATENATE STORED.RUNWAY(2);
 DEMAND(1).MSG = DATA STORED CONCATENATE STORED.DEMAND(1);
 DEMAND(2).MSG = DATA STORED CONCATENATE STORED.CONF(1);
 CONFIG(1).MSG = DATA STORED CONCATENATE STORED.CONF(2);
 CONFIG(2).MSG = DATA STORED CONCATENATE STORED.OLIST(1);
 CONFLST(1).MSG = DATA STORED CONCATENATE STORED.OLIST(2);
 CONFLST(2).MSG = DATA STORED CONCATENATE STORED.PARMOFT;
 TRANLST.MSG = DATA STORED CONCATENATE STORED.TRANLST;
 EQPLOG.MSG = DATA STORED CONCATENATE STORED.RWYLOG;
 QUELEN.MSG = DATA STORED CONCATENATE STORED.QLENGTH;
 OHSTAT.MSG = DATA STORED CONCATENATE STORED.OHSTATUS;
 WKLOG.MSG = DATA STORED CONCATENATE STORED.APLOG1;
 SURFLOG.MSG = DATA STORED CONCATENATE STORED.APLOG2;
 OAGLOG.MSG = DATA STORED CONCATENATE STORED.DMNDLOG;

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; [J = 1 to 2]

CALL MINIMA; [Compute minima based on equipment status]

IN (RWYEQP(J), RWYMIN);

INOUT (APTSTAT(J), CNVTAPT(J));

CALL WIND; [compute crosswind & tailwind components of wind for each runway]

INOUT (APTSTAT(J), CNVTAPT(J));

CALL CLOSING; [determine runway closures]

IN (CNVTFRM);

INOUT (APTSTAT(J), CNVTAPT(J));

CALL FILES; [determine capacity file number for each configuration and set CMDTN variable to indicate VPR(=1) or IPR(=2)]

IN (APTSTAT(J), CNVTAPT(J));

INOUT (FILENUM(J), CMDTN(J));

CALL ELIG; [determine eligibility of configurations]

IN (CNFGRQ, APTSTAT(J), CNVTAPT(J), RWYEQP(J));

INOUT (ELGBLTY(J));

CALL PERCENT; [compute north and south demands based on fix-to-runway assignments]

IN (CNVTDEM(J), CNFGRQ);

INOUT (PRCARR(J));

```
      CALL CAPSAT; [compute capacity and balance demand for each configuration]
      IN (PRCARR(J), CNFGRQ, CAPFILE, FILENUM(J), ELGBLTY(J));
      INOUT (INFORM(J));
    ENDLOOP;
    CALL QFIX; [update departure runways for current configuration]
    IN (CNFGRQ(CONFIND(1));
    INOUT (QUELEN);
  END UPDATE;
```

ROUTINE AGLOBAL

INOUT (APTSTAT(I), CNVTAPT(I), MIDFLAG(I), ABEGIN(I), CVTABGN(I), MBEGIN(I), ANOW(I), CVTANOW(I),
MNOW(I));

[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 ABEGIN(I).RUNWAY(K).TOWER.ARR

THEN ANOW(I).RUNWAY(K).TOWER.ARR = APTSTAT(I).RUNWAY(K).TOWER.ARR;
ELSE APTSTAT(I).RUNWAY(K).TOWER.ARR = ANOW(I).RUNWAY(K).TOWER.ARR;

IF APTSTAT(I).RUNWAY(K).TOWER.DEP NE ABEGIN(I).RUNWAY(K).TOWER.DEP

THEN ANOW(I).RUNWAY(K).TOWER.DEP = APTSTAT(I).RUNWAY(K).TOWER.DEP;
ELSE APTSTAT(I).RUNWAY(K).TOWER.DEP = ANOW(I).RUNWAY(K).TOWER.DEP;

IF APTSTAT(I).RUNWAY(K).SURF NE ABEGIN(I).RUNWAY(K).SURF;

THEN ANOW(I).RUNWAY(K).SURF = APTSTAT(I).RUNWAY(K).SURF;
ELSE APTSTAT(I).RUNWAY(K).SURF = ANOW(I).RUNWAY(K).SURF;

IF APTSTAT(I).RUNWAY(K).BRK NE ABEGIN(I).RUNWAY(K).BRK

THEN ANOW(I).RUNWAY(K).BRK = APTSTAT(I).RUNWAY(K).BRK;
ELSE APTSTAT(I).RUNWAY(K).BRK = ANOW(I).RUNWAY(K).BRK;

ENDLOOP;

IF CNVTAPT(I).WX.CEIL NE CVTABGN(I).WX.CEIL

THEN

CVTANOW(I).WX.CEIL = CNVTAPT(I).WX.CEIL;
ANOW(I).WX.CEIL = APTSTAT(I).WX.CEIL;

ELSE

CNVTAPT(I).WX.CEIL = CVTANOW(I).WX.CEIL;
APTSTAT(I).WX.CEIL = ANOW(I).WX.CEIL;

```
IF CNVTAPT(I).WX.VIS NE CVTABGN(I).WX.VIS
  THEN
    CVTANOW(I).WX.VIS = CNVTAPT(I).WX.VIS;
    ANOW(I).WX.VIS = APTSTAT(I).WX.VIS;
  ELSE
    CNVTAPT(I).WX.VIS = CVTANOW(I).WX.VIS;
    APTSTAT(I).WX.VIS = ANOW(I).WX.VIS;
IF CNVTAPT(I).WIND.DIR NE CVTABGN(I).WIND.DIR
  THEN
    CVTANOW(I).WIND.DIR = CNVTAPT(I).WIND.DIR;
    ANOW(I).WIND.DIR = APTSTAT(I).WIND.DIR;
  ELSE
    CNVTAPT(I).WIND.DIR = CVTANOW(I).WIND.DIR;
    APTSTAT(I).WIND.DIR = ANOW(I).WIND.DIR;
IF CNVTAPT(I).WIND.VEL NE CVTABGN(I).WIND.VEL
  THEN
    CVTANOW(I).WIND.VEL = CNVTAPT(I).WIND.VEL;
    ANOW(I).WIND.VEL = APTSTAT(I).WIND.VEL;
  ELSE
    CNVTAPT(I).WIND.VEL = CVTANOW(I).WIND.VEL;
    APTSTAT(I).WIND.VEL = ANOW(I).WIND.VEL;
IF MIDFLAG(I) NE MBEGIN(I)
  THEN MNOW(I) = MIDFLAG(I);
  ELSE MIDFLAG(I) = MNOW(I);
END AGLOBAL;
```


ROUTINE RGLOBAL

INOUT (RWYEQP(I), RBEGIN(I), RNOW(I));

[This routine reconciles different versions of equipment status information]

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

IF RWYEQP(I).RUNWAY(K).CATII NE RBEGIN(I).RUNWAY(K).CATII

THEN RNOW(I).RUNWAY(K).CATII = RWYEQP(I).RUNWAY(K).CATII;

ELSE RWYEQP(I).RUNWAY(K).CATII + RNOW(I).RUNWAY(K).CATII;

IF RWYEQP(I).RUNWAY(K).LOC NE RBEGIN(I).RUNWAY(K).LOC

THEN RNOW(I).RUNWAY(K).LOC = RWYEQP(I).RUNWAY(K).LOC;

ELSE RWYEQP(I).RUNWAY(K).LOC = RNOW(I).RUNWAY(K).LOC;

IF RWYEQP(I).RUNWAY(K).GS NE RBEGIN(I).RUNWAY(K).GS

THEN RNOW(I).RUNWAY(K).GS = RWYEQP(I).RUNWAY(K).GS;

ELSE RWYEQP(I).RUNWAY(K).GS = RNOW(I).RUNWAY(K).GS;

IF RWYEQP(I).RUNWAY(K).OM NE RBEGIN(I).RUNWAY(K).OM

THEN RNOW(I).RUNWAY(K).OM = RWYEQP(I).RUNWAY(K).OM;

ELSE RWYEQP(I).RUNWAY(K).OM = RNOW(I).RUNWAY(K).OM;

IF RWYEQP(I).RUNWAY(K).MM NE RBEGIN(I).RUNWAY(K).MM;

THEN RNOW(I).RUNWAY(K).MM = RWYEQP(I).RUNWAY(K).MM;

ELSE RWYEQP(I).RUNWAY(K).MM = RNOW(I).RUNWAY(K).MM;

IF RWYEQP(I).RUNWAY(K).IM NE RBEGIN(I).RUNWAY(K).IM;

THEN RNOW(I).RUNWAY(K).IM = RWYEQP(I).RUNWAY(K).IM;

ELSE RWYEQP(I).RUNWAY(K).IM = RNOW(I).RUNWAY(K).IM;

```

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

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

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 RGLOBAL;

AD-A127 828

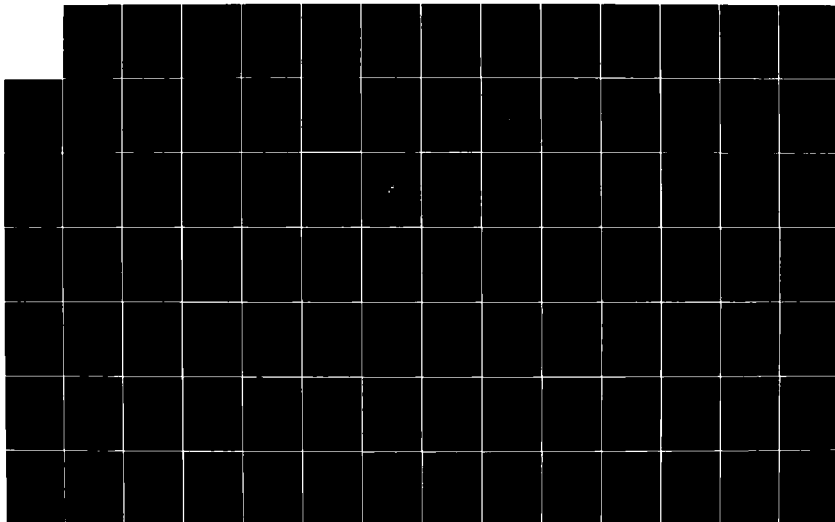
SOFTWARE DESCRIPTION FOR THE O'HARE RUNWAY
CONFIGURATION MANAGEMENT SYSTE.. (U) MITRE CORP MCLEAN
VA METREX DIV 5 KAPOUSS! OCT 82 MTR-82W125-VOL-2
FAA-EM-82-28-VOL-2 DTFA01-81-C-10003

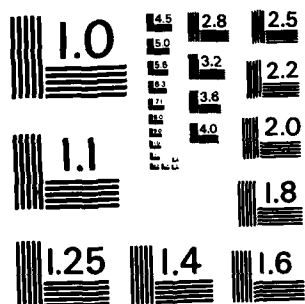
2/5

UNCLASSIFIED

F/G 17/7

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

ROUTINE EGLOBAL

INOUT (EQPLOG, CNVTEQP, EBEGIN, CVTERGN, ENOW, CVTENOW);
[This routine reconciles different versions of equipment planning log]

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

ENOW.TABLE(J).RWY = EQPLOG.TABLE(J).RWY;
ENOW.TABLE(J).EQUIPMENT = EQPLOG.TABLE(J).EQUIPMENT;
ENOW.TABLE(J).OTS = EQPLOG.TABLE(J).OTS;
ENOW.TABLE(J).RTS = EQPLOG.TABLE(J).RTS;
ENOW.TABLE(J).REMARKS = EQPLOG.TABLE(J).REMARKS;
CVTENOW.TABLE(J).OTS = CNVTEQP.TABLE(J).OTS;
CVTENOW.TABLE(J).RTS = CNVTEQP.TABLE(J).RTS;

ENDLOOP;

END EGLOBAL;

ROUTINE DGLOBAL

INOUT (DEMAND(I), CNVTDEM(I), DBEGIN(I), CVTDBGN(I), DNOW(I), CVTDNOW(I));
[This routine reconciles different versions of demand profile information]

IF CNVTDEM(I).ARR.TOTAL NE CVTDBGN(I).ARR.TOTAL

THEN

DNOW(I).ARR.TOTAL = DEMAND(I).ARR.TOTAL;
CVTDNOW(I).ARR.TOTAL = CNVTDEM(I).ARR.TOTAL;

ELSE

DEMAND(I).ARR.TOTAL = DNOW(I).ARR.TOTAL;
CNVTDEM(I).ARR.TOTAL = CVTDNOW(I).ARR.TOTAL;

IF CNVTDEM(I).ARR.KUBBS NE CVTDBGN(I).ARR.KUBBS

THEN

DNOW(I).ARR.KUBBS = DEMAND(I).ARR.KUBBS;
CVTDNOW(I).ARR.KUBBS = CNVTDEM(I).ARR.KUBBS;

ELSE

DEMAND(I).ARR.KUBBS = DNOW(I).ARR.KUBBS;
CNVTDEM(I).ARR.KUBBS = CVTDNOW(I).ARR.KUBBS

IF CNVTDEM(I).ARR.PLANT NE CVTDBGN(I).ARR.PLANT

THEN

DNOW(I).ARR.PLANT = DEMAND(I).ARR.PLANT;
CVTDNOW(I).ARR.PLANT = CNVTDEM(I).ARR.PLANT;

ELSE

DEMAND(I).ARR.PLANT = DNOW(I).ARR.PLANT;
CNVTDEM(I).ARR.PLANT = CVTDNOW(I).ARR.PLANT

IF CNVTDEM(I).ARR.CGT NE CVTDBGN(I).ARR.CGT

THEN

DNOW(I).ARR.CGT = DEMAND(I).ARR.CGT;
CVTDNOW(I).ARR.CGT = CNVTDEM(I).ARR.CGT;

```
ELSE
DEMAND(I).ARR.CGT = DNOW(I).ARR.CGT
CNVTDEM(I).ARR.CGT = CVTDNOW(I).ARR.CGT

IF CNVTDEM(I).ARR.VAINS NE CVTDGCH(I).ARR.VAINS

THEN
DNOW(I).ARR.VAINS = DEMAND(I).ARR.VAINS;
CVTDNOW(I).ARR.VAINS = CNVTDEM(I).ARR.VAINS;

ELSE
DEMAND(I).ARR.VAINS = DNOW(I).ARR.VAINS;
CNVTDEM(I).ARR.VAINS = CVTDNOW(I).ARR.VAINS;

IF CNVTDEM(I).ARR.FARMH NE CVTDGCH(I).ARR.FARMH

THEN
DNOW(I).ARR.FARMH = DEMAND(I).ARR.FARMH;
CVTDNOW(I).ARR.FARMH = CNVTDEM(I).ARR.FARMH;

ELSE
DEMAND(I).ARR.FARMH = DNOW(I).ARR.FARMH;
CNVTDEM(I).ARR.FARMH = CVTDNOW(I).ARR.FARMH;

IF CNVTDEM(I).ARR.MKE_A NE CVTDGCH(I).ARR.MKE_A

THEN
DNOW(I).ARR.MKE_A = DEMAND(I).ARR.MKE_A;
CVTDNOW(I).ARR.MKE_A = CNVTDEM(I).ARR.MKE_A;

ELSE
DEMAND(I).ARR.MKE_A = DNOW(I).ARR.MKE_A;
CNVTDEM(I).ARR.MKE_A = CVTDNOW(I).ARR.MKE_A;

IF CNVTDEM(I).DEP.TOTAL NE CVTDGCH(I).DEP.TOTAL

THEN
DNOW(I).DEP.TOTAL = DEMAND(I).DEP.TOTAL;
CVTDNOW(I).DEP.TOTAL = CNVTDEM(I).DEP.TOTAL;
```

```
ELSE
  DEMAND(I).DEP.TOTAL = DNOW(I).DEP.TOTAL;
  CNVTDEM(I).DEP.TOTAL = CVTDNOW(I).DEP.TOTAL;

  IF CNVTDEM(I).DEP.NORTH NE CVTDBGN(I).DEP.NORTH

    THEN
      DNOW(I).DEP.NORTH = DEMAND(I).DEP.NORTH;
      CVTDNOW(I).DEP.NORTH = CNVTDEM(I).DEP.NORTH;

    ELSE
      DEMAND(I).DEP.NORTH = DNOW(I).DEP.NORTH;
      CNVTDEM(I).DEP.NORTH = CVTDNOW(I).DEP.NORTH;

    IF CNVTDEM(I).DEP.EAST NE CVTDBGN(I).DEP.EAST

      THEN
        DNOW(I).DEP.EAST = DEMAND(I).DEP.EAST;
        CVTDNOW(I).DEP.EAST = CNVTDEM(I).DEP.EAST;

      ELSE
        DEMAND(I).DEP.EAST = DNOW(I).DEP.EAST;
        CNVTDEM(I).DEP.EAST = CVTDNOW(I).DEP.EAST;

    IF CNVTDEM(I).DEP.SOUTH NE CVTDBGN(I).DEP.SOUTH

      THEN
        DNOW(I).DEP.SOUTH = DEMAND(I).DEP.SOUTH;
        CVTDNOW(I).DEP.SOUTH = CNVTDEM(I).DEP.SOUTH;

      ELSE
        DEMAND(I).DEP.SOUTH = DNOW(I).DEP.SOUTH;
        CNVTDEM(I).DEP.SOUTH = CVTDNOW(I).DEP.SOUTH;

    IF CNVTDEM(I).DEP.WEST NE CVTDBGN(I).DEP.WEST

      THEN
        DNOW(I).DEP.WEST = DEMAND(I).DEP.WEST;
        CVTDNOW(I).DEP.WEST = CNVTDEM(I).DEP.WEST;
```



```
ELSE
  DEMAND(I).DEP.WEST = DNOW(I).DEP.WEST;
  CNVTDEM(I).DEP.WEST = CVTDNOW(I).DEP.WEST;
IF CNVTDEM(I).DEP.MKE_D NE CVTDBGN(I).DEP.MKE_D
THEN
  DNOW(I).DEP.MKE_D = DEMAND(I).DEP.MKE_D;
  CVTDNOW(I).DEP.MKE_D = CNVTDEM(I).DEP.MKE_D;
ELSE
  DEMAND(I).DEP.MKE_D = DNOW(I).DEP.MKE_D;
  CNVTDEM(I).DEP.MKE_D = CVTDNOW(I).DEP.MKE_D;
END DGLOBAL;
```

ROUTINE CGLOBAL

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

IF CONFIND(I) NE CBEGIN(I)

THEN

 CNOW(I) = CONFIND(I);

IF K EQ 1

THEN

LOOP; [J = 1 to 4]

 CVTQNOW(J) = 0;

 QNOW.QL(J) = '0';

ENDLOOP;

ELSE

 CONFIND(I) = CNOW(I);

END CGLOBAL;

ROUTINE PGLOBAL

INOUT (PARAM, CNVTPRM, PBEGIN, CVTPBGN, PNOW, CVTPNOW);
 [This routine reconciles different versions of parameters information]

IF CNVTPRM.ARR.CRSS NE CVTPBGN.ARR.CRSS

THEN

 PNOW.PARAMETER.ARR.CRSS = PARAM.PARAMETER.ARR.CRSS;

 CVTPNOW.ARR.CRSS = CNVTPRM.ARR.CRSS;

```
      ELSE
        PARAM.PARAMETER.ARR.CRSS = PNOW.PARAMETER.ARR.CRSS;
        CNVTPRM.ARR.CRSS = CVTPNOW.ARR.CRSS;
    IF CNVTPRM.ARR.TAIL NE CVTPBGN.ARR.TAIL
      THEN
        PNOW.PARAMETER.ARR.TAIL = PNOW.PARAMETER.ARR.TAIL;
        CVTPNOW.ARR.TAIL = CNVTPRM.ARR.TAIL;
      ELSE
        PARAM.PARAMETER.ARR.TAIL = PNOW.PARAMETER.ARR.TAIL;
        CNVTPRM.ARR.TAIL = CVTPNOW.ARR.TAIL;
    IF CNVTPRM.DEP.CRSS NE CVTPBGN.DEP.CRSS
      THEN
        PNOW.PARAMETER.DEP.CRSS = PARAM.PARAMETER.DEP.CRSS;
        CVTPNOW.DEP.CRSS = CNVTPRM.DEP.CRSS;
      ELSE
        PARAM.PARAMETER.DEP.CRSS = PNOW.PARAMETER.DEP.CRSS;
        CNVTPRM.DEP.CRSS = CVTPNOW.DEP.CRSS;
    IF CNVTPRM.DEP.TAIL NE CVTPBGN.DEP.TAIL
      THEN
        PNOW.PARAMETER.DEP.TAIL = PARAM.PARAMETER.DEP.TAIL;
        CVTPNOW.DEP.TAIL = CNVTPRM.DEP.TAIL;
      ELSE
        PARAM.PARAMETER.DEP.TAIL = PNOW.PARAMETER.DEP.TAIL;
        CNVTPRM.DEP.TAIL = CVTPNOW.DEP.TAIL;
  END PGLOBAL;
```

ROUTINE WGLOBALINOUT (WXLOG, CNVTWX, WBEGIN, CVTWBGN, WNOW, CVTWNOW);

[This routine reconciles different versions of wind and weather planning log information]

LOOP; [J = 11 to 13] [From 11th to 13th message for AC]

WNOW.TABLE(J).TIME = WXLOG.TABLE(J).TIME;
 WNOW.TABLE(J).CEIL = WXLOG.TABLE(J).CEIL;
 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).REMARKS = WXLOG.TABLE(J).REMARKS;
 CVTWNOW.TABLE(J).TIME = CNVTWX.TABLE(J).TIME;
 CVTWNOW.TABLE(J).CEIL = CNVTWX.TABLE(J).CEIL;
 CVTWNOW.TABLE(J).VIS = CNVTWX.TABLE(J).VIS;
 CVTWNOW.TABLE(J).DIR = CNVTWX.TABLE(J).DIR;
 CVTWNOW.TABLE(J).VEL = CNVTWX.TABLE(J).VEL;

ENDLOOP;END WGLOBAL;ROUTINE QGLOBALINOUT (QUELEN, CNVTQLN, QBEGIN, CVTQBGN, QNOW, CVTQNOW)

[This routine reconciles different versions of current departure queue information]

LOOP; [J = 1 to 4] [up to 4 departure runways]IF CNVTQLN(J) NE CVTQBGN(J)THEN

QNOW.QL(J) = QUELEN.QL(J);
 CVTQNOW(J) = CNVTQLN(J);

ELSE

QUELEN.QL(J) = QNOW.QL(J);
 CNVTQLN(J) = CVTQNOW(J);

ENDLOOP;END QGLOBAL;

ROUTINE SGLOBALINOUT (SURFLOG, CNVTSRF, SERGIN, CVTSEGN, SNOW, CVTSNOW);

[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).RWY = SURFLOG.TABLE(J).RWY;
SNOW.TABLE(J).SURF = SURFLOG.TABLE(J).SURF;
SNOW.TABLE(J).BRAK = SURFLOG.TABLE(J).BRAK;
SNOW.TABLE(J).CLOSED = SURFLOG.TABLE(J).CLOSED;
SNOW.TABLE(J).OPEN = SURFLOG.TABLE(J).OPEN;
SNOW.TABLE(J).REMARKS = SURFLOG.TABLE(J).REMARKS;

CVTSNOW.TABLE(J).TIME = CNVTSRF.TABLE(J).TIME;

ENDLOOP;END SGLOBAL;

ROUTINE GGLOBALINOUT (OAGLOG, CNVTOAG, GBEGIN, CVTGBGN, GNOW, CVTGNOW);

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

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

GNOW.TABLE(J).TTLARR = OAGLOG.TABLE(J).TTLARR;
 GNOW.TABLE(J).TTLDEP = OAGLOG.TABLE(J).TTLDEP;
 GNOW.TABLE(J).KUBBS = OAGLOG.TABLE(J).KUBBS;
 GNOW.TABLE(J).CGT = OAGLOG.TABLE(J).CGT;
 GNOW.TABLE(J).VAINS = OAGLOG.TABLE(J).VAINS;
 GNOW.TABLE(J).FARMH = OAGLOG.TABLE(J).FARMH;
 GNOW.TABLE(J).NORTH = OAGLOG.TABLE(J).NORTH;
 GNOW.TABLE(J).EAST = OAGLOG.TABLE(J).EAST;
 GNOW.TABLE(J).SOUTH = OAGLOG.TABLE(J).SOUTH;
 GNOW.TABLE(J).WEST = OAGLOG.TABLE(J).WEST;

CVTGNOW.TABLE(J).TTLARR = CNVTOAG.TABLE(J).TTLARR;
 CVTGNOW.TABLE(J).TTLDEP = CNVTOAG.TABLE(J).TTLDEP;
 CVTGNOW.TABLE(J).KUBBS = CNVTOAG.TABLE(J).KUBBS;
 CVTGNOW.TABLE(J).CGT = CNVTOAG.TABLE(J).CGT;
 CVTGNOW.TABLE(J).VAINS = CNVTOAG.TABLE(J).VAINS;
 CVTGNOW.TABLE(J).FARMH = CNVTOAG.TABLE(J).FARMH;
 CVTGNOW.TABLE(J).NORTH = CNVTOAG.TABLE(J).NORTH;
 CVTGNOW.TABLE(J).EAST = CNVTOAG.TABLE(J).EAST;
 CVTGNOW.TABLE(J).SOUTH = CNVTOAG.TABLE(J).SOUTH;
 CVTGNOW.TABLE(J).WEST = CNVTOAG.TABLE(J).WEST;

ENDLOOP;END GGLOBAL;

ROUTINE CLOSINGIN (CNVTPRM);INOUT (ARPT_DATA(I), CNVRT_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.ARR;

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

[closed due to wind]

IF (CNVRT_APT(I).RUNWAY(J).CRSS GT CNVTPRM.ARR.CRSS) OR (CNVRT_APT(I).RUNWAY(J).TAIL GT CNVTPRM.ARR.TAIL)THEN

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

IF (CNVRT_APT(I).RUNWAY(J).CRSS GT CNVTPRM.DEP.CRSS) OR (CNVRT_APT(I).RUNWAY(J).TAIL GT CNVTPRM.DEP.TAIL)THEN

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

[closed due to minima]

IF (CNVRT_APT(I).WX.CEIL LT CNVRT_APT(I).RUNWAY(J).CEIL) OR (CNVRT_APT(I).WX.VIS LT CNVRT_APT(I).RUNWAY(J).VIS)THEN

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

ENDLOOP;END CLOSING;

ROUTINE WINDINOUT (ARPT_DATA(I), CNRT_APT(I));

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

\$TWO = 2;

ANGLE(1) = 220.;
 ANGLE(2) = 220.;
 ANGLE(3) = 270.;
 ANGLE(4) = 270.;
 ANGLE(5) = 320.;
 ANGLE(6) = 320.;
 ANGLE(7) = 40.;
 ANGLE(8) = 40.;
 ANGLE(9) = 90.;
 ANGLE(10) = 90.;
 ANGLE(11) = 140.;
 ANGLE(12) = 140.;

ARPT_DATA(I).RUNWAY.DIR = ARPT_DATA(I).WIND.DIR;

ARPT_DATA(I).RUNWAY.VEL = ARPT_DATA(I).WIND.VEL;

CNVRT_APT(I).RUNWAY.DIR = CNVRT_APT(I).WIND.DIR;

CNVRT_APT(I).RUNWAY.VEL = CNVRT_APT(I).WIND.VEL;

ANGLE = (ANGLE - CNVRT_APT(I).RUNWAY.DIR) * 0.01745; [convert to radians]

ARPT_DATA(I).RUNWAY.RVR = (2) ' ';

LOOP;

CNVRT_APT(I).RUNWAY(J).CRSS = CNVRT_APT(I).WIND.VEL * ABS(SIN(ANGLE(J)));

CNVRT_APT(I).RUNWAY(J).CRSS = FLOAT(FLOOR(CNVRT_APT(I).RUNWAY(J).CRSS + .5));

ARPT_DATA(I).RUNWAY(J).CRSS = SUBSTR(P(CNVRT_APT(I).RUNWAY(J).CRSS.\$TWO),1,2);

IF ABS (ANGLE(J)) GE 1.57079

THEN

CNVRT_APT(I).RUNWAY(J).TAIL = 0.0;
ARPT_DATA(I).RUNWAY(J).TAIL = ' 0';

ELSE

CNVRT_APT(I).RUNWAY(J).TAIL = CNVRT_APT(I).WIND.VEL * COS (ANGLE(J));
CNVRT_APT(I).RUNWAY(J).TAIL = FLOAT(FLOOR(CNVRT_APT(I).RUNWAY(J).TAIL + .5));
ARPT_DATA(I).RUNWAY(J).TAIL = SUBSTR(P(CNVRT_APT(I).RUNWAY(J).TAIL,\$TWO),1,2);

ENDLOOP;

END WIND;

ROUTINE MINIMA

IN (RWYEQ(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 RWYEQ(I).RUNWAY(J).CATII EQ (2) ' '

THEN [CATII is up]

CNVTAPT(I).RWY(J).CEIL = RWYMIN(J).CATII.NONE.CEIL;

CNVTAPT(I).RWY(J).VIS = RWYMIN(J).CATII.NONE.VIS;

APTSTAT(I).RWY(J).CEIL = SUBSTR(F(CNVTAPT(I).RWY(J).CEIL, \$FOUR),1,4);

C = SUBSTR(F(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 (RWYEQ(I).RUNWAY(J).LOC NE (2) ' ') AND (RWYEQ(I).RUNWAY(J).NDB_VOR NE (2) ' ')

THEN [both localizer and NDB_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 NDB_VOR are not down]

IF (RWYEQ(I).RUNWAY(J).LOC NE (2) ' ') AND (RWYEQ(I).RUNWAY(J).NDB_VOR EQ (2) ' ')

THEN [Localizer is down and NDB_VOR is up]

CNVTAPT(I).RWY(J).CEIL = RWYMIN(J).NDB_VOR.NONE.CEIL;
CNVTAPT(I).RWY(J).VIS = RWYMIN(J).NDB_VOR.NONE.VIS;

IF RWYEQP(I).RUNWAY(J).RAIL NE (2) ' '

THEN [RAIL is also down]

CNVTAPT(I).RWY(J).CEIL = MAX(CNVTAPT(I).RWY(J).CEIL,
RWYMIN(J).NDB_VOR.RAIL.CEIL);
CNVTAPT(I).RWY(J).VIS = MAX(CNVTAPT(I).RWY(J).VIS,
RWYMIN(J).NDB_VOR.ALS.VIS)

APTSTAT(I).RWY(J).CEIL = SUBSTR(F(CNVTAPT(I).RWY(J).
CEIL,\$FOUR),1,4);

C = SUBSTR(F(100. * CNVTAPT(I).RWY(J).VIS,\$THREE),1,3);

APTSTAT(I).RWY(J).VIS = SUBSTR(C,1,1) CONCATENATE '.'
CONCATENATE SUBSTR(C,2,2);

ELSE [localizer is up]

IF (RWYEQP(I).RUNWAY(J).LOC EQ (2) ' ') AND (RWYEQP(I).
RUNWAY(J).GS NE (2) ' ')

THEN [glide slope is down]

CNVTAPT(I).RWY(J).CEIL = RWYMIN(J).LOC.NONE.CEIL;
CNVTAPT(I).RWY(J).VIS = RWYMIN(J).LOC.NONE.VIS;

IF RWYEQP(I).RUNWAY(J).MM NE (2) ' '

THEN [middle marker is down]

CNVTAPT(I).RWY(J).CEIL =
MAX(CNVTAPT(I).RWY(J).CEIL,
RWYMIN(J).LOC.MM.CEIL);
CNVTAPT(I).RWY(J).VIS = MAX(CNVTAPT(I).
RWY(J).VIS, RWYMIN(J).LOC.MM.VIS);

IF RWYEQP(I).RUNWAY(J).RAIL NE (2) ' '

THEN [RAIL is also down]

CNVTAPT(I).RWY(J).CEIL =
MAX(CNVTAPT(I).RWY(J).CEIL,RWYMIN(J)
.LOC.RAIL.CEIL;
CNVTAPT(I).RWY(J).VIS =
MAX(CNVTAPT(I).RWY(J).VIS,RWYMIN(J).
LOC.RAIL.VIS);

IF RWYEQP(I).RUNWAY(J).ALS NE (2) ' '

THEN [ALS is also down]

CNVTAPT(I).RWY(J).CEIL =
MAX(CNVTAPT(I).RWY(J).CEIL,RWYMIN(J)
.LOC.ALS.CEIL);

CNVTAPT(I).RWY(J).VIS =
MAX(CNVTAPT(I).RWY(J).VIS,RWYMIN(J).
LOC.ALS.VIS);

APTSTAT(I).RWY(J).CEIL =
SUBSTR(F(CNVTAPT(I).RWY(J).CEIL,\$FOU
R),1,4);

C = SUBSTR(F(100. *
CNVTAPT(I).RWY(J).VIS,\$THREE),1,3);

APTSTAT(I).RWY(J).VIS =
SUBSTR(C,1,1) CONCATENATE '.'
CONCATENATE SUBSTR(C,2,2);

ELSE [localizer is up and glide slope is up]

IF (RWYEQP(I).RUNWAY(J).LOC EQ (2) ' ') AND
(RWYEQP(I).RUNWAY(J).GS EQ (2) ' ')

THEN [glide slope is down]

CHVTAPT(I).RWY(J).CEIL =
RWYMIN(J).ILS.NONE.CEIL;

CHVTAPT(I).RWY(J).VIS =
RWYMIN(J).ILS.NONE.VIS;

IF RWYBQP(I).RUNWAY(J).MM NE (2) ' '

THEN [middle marker is also down]

CHVTAPT(I).RWY(J).CEIL =
MAX(CHVTAPT(2).RWY(J).CEIL,RWYMIN(J)
.ILS.MM.CEIL);

CHVTAPT(I).RWY(J).VIS =
MAX(CHVTAPT(1).RWY(J).VIS,RWYMIN(J).
ILS.MM.VIS);

IF (RWYBQP(I).RUNWAY(J).RAIL NE (2) ' ')
OR (RWYBQP(I).RUNWAY(J).ALS NE (2) ' ')

THEN [RAIL is also down or ALS is down]

CHVTAPT(I).RWY(J).CEIL =
MAX(CHVTAPT(1).RWY(J).CEIL,RWYMIN(J)
.RAIL_ALS.CEIL);

CHVTAPT(I).RWY(J).VIS =
MAX(CHVTAPT(1).RWY(J).VIS,RWYMIN(J).
RAIL_ALS.VIS);

IF (RWYBQP(I).RUNWAY(J).TDZ NE (2) ' ')

THEN [TDZ is also down]

CHVTAPT(I).RWY(J).CEIL =
MAX(CHVTAPT(1).RWY(J).CEIL,RWYMIN(J)
.ILS.TDZ.CEIL);

2-103

```

CNVTAPT(I).RWY(J).VIS =
MAX(CNVTAPT(I).RWY(J).VIS,RWYMIN(J).
ILS.TDZ.VIS);

IF (RWYEQP(I).RUNWAY(J).CL NE (2) ' '
THEN [CL is also down]
CNVTAPT(I).RWY(J).CEIL =
MAX(CNVTAPT(I).RWY(J).CEIL,RWYMIN(J)
.ILS.CL.CEIL);
CNVTAPT(I).RWY(J).VIS =
MAX(CNVTAPT(I).RWY(J).VIS,RWYMIN(J).
ILS.CL.VIS);
APTSTAT(I).RWY(J).CEIL =
SUBSTR(F(CNVTAPT(I).RWY(J).CEIL,$FOU
R),4);
C = SUBSTR(F(100. * CNVTAPT(I).
RWY(J).VIS,$THREE),1,3);
APTSTAT(I).RWY(J).VIS =
SUBSTR(C,1,1) CONCATENATE '.'
CONCATENATE SUBSTR(C,2,2);

IF RWYEQP(I).RUNWAY(J).HIRL NE (2) ' '
THEN [HIRL is down]
CNVTAPT(I).RWY(J).VIS = 2.0;
APTSTAT(I).RWY(J).VIS = '2.00';

END MINIMA;
```

ROUTINE ELIG

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

INOUT (ELGBLTY(I));

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

PERFORM CONFIGURATION_ID_SET_UP;

\$TWO = 2;

ELGBLTY(I).ID = (73) '0'B;
ELGBLTY(I).NUM = 0;

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

THEN ELGBLTY(I).ID = (73)'1'B [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'B [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;
            ELGELTY(I).ID = SUBSTR(ELGELTY(I).ID,1,J-1)
            CONCATENATE EFLAG CONCATENATE
            SUBSTR(ELGELTY(I).ID,J, 73-J);
ENDLOOP;
LOOP; [J = 1 to 73]
  IF SUBSTR(ELGELTY(I).ID,J,1) = '0'B
    THEN ELGELTY(I).NUM = ELGELTY(I).NUM + 1;
ENDLOOP;
END ELIG;

```


PROCESS CONFIGURATION_ID_SET_UP

[This process initializes certain necessary variables for KLIG routine]

BZERO = (12)'0'B;

[set up parallel runway configuration ID's]

PARAPP(1) = '110000000000'B CONCATENATE BZERO; [4R, 4L]
 PARAPP(2) = '001100000000'B CONCATENATE BZERO; [9R, 9L]
 PARAPP(3) = '000011000000'B CONCATENATE BZERO; [14R, 14L]
 PARAPP(4) = '000000110000'B CONCATENATE BZERO; [22R, 22L]
 PARAPP(5) = '000000001100'B CONCATENATE BZERO; [27R, 27L]
 PARAPP(6) = '000000000011'B CONCATENATE BZERO; [32R, 32L]

[set up certain dual runway configuration ID's]

DUALAPP(1) = '101000000000'B CONCATENATE BZERO; [4R, 9R]
 DUALAPP(2) = '001010000000'B CONCATENATE BZERO; [9R, 14R]

[set up triple runway configurations]

TRIPAPP(1) = '101100000000'B CONCATENATE BZERO; [4R, 9R, 9L]
 TRIPAPP(2) = '001011000000'B CONCATENATE BZERO; [9R, 14R, 14L]
 TRIPAPP(3) = '001010100000'B CONCATENATE BZERO; [9R, 14R, 22R]
 TRIPAPP(4) = '000011010000'B CONCATENATE BZERO; [14R, 22R, 22L]
 TRIPAPP(5) = '000010110000'B CONCATENATE BZERO; [14R, 22R, 22L]
 TRIPAPP(6) = '000010100100'B CONCATENATE BZERO; [14R, 22R, 27L]
 TRIPAPP(7) = '000000101100'B CONCATENATE BZERO; [22R, 27R, 27L]
 TRIPAPP(8) = '000000001101'B CONCATENATE BZERO; [27R, 27L, 32L]

[set up hold short configurations]

HLDShRT(1) = '000010000100'B CONCATENATE BZERO; [14R, 27L]
 HLDShRT(2) = '00000000101'B CONCATENATE BZERO; [27L, 32L]
 HLDShRT(3) = '000000101000'B CONCATENATE BZERO; [22R, 27R]
 HLDShRT(4) = '001010000000'B CONCATENATE BZERO; [9R, 14R]

END CONFIGURATION_ID_SET_UP;

PROCESS BELOW 200 CEILING PLUS EQUIPMENT OUTAGE ELIGIBILITY CHECK
 (This determines eligibility of configurations with ceiling below 200 and certain equipment out)

IF (CNVTAPT(1).WX.CEIL LT 200.) AND
 (RWYEQP(1).RUNWAY(5).LOC NE (2) ' ') OR
 (RWYEQP(1).RUNWAY(6).LOC NE (2) ' ') OR
 (RWYEQP(1).RUNWAY(5).GS NE (2) ' ') OR
 (RWYEQP(1).RUNWAY(6).GS NE (2) ' ') OR
 (RWYEQP(1).RUNWAY(5).OM NE (2) ' ') OR
 (RWYEQP(1).RUNWAY(6).OM NE (2) ' ') OR
 (RWYEQP(1).RUNWAY(5).MM NE (2) ' ') OR
 (RWYEQP(1).RUNWAY(6).MM NE (2) ' ') OR
 (RWYEQP(1).RUNWAY(5).ALS NE (2) ' ') OR
 (RWYEQP(1).RUNWAY(6).ALS NE (2) ' ') OR

THEN ELGLTY(1).ID = (73) '1'B;
 [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 PLUS 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(1).CLOSED.ARR EQ (2) ' 'THEN AINELIG = '0'B;ELSE AINELIG = '1'B;LOOP; [J = 2 To 12]IF APTSTAT(I).RUNWAY(J).CLOSED.ARR EQ (2) ' 'THEN AINELIG = AINELIG CONCATENATE '0'B;ELSE AINELIG = AINELIG CONCATENATE '1'B;ENDLOOP;IF APTSTAT(I).RUNWAY(1).CLOSED.DEP EQ (2) ' 'THEN DINELIG = '0'B;ELSE DINELIG = '1'B;LOOP; [J = 2 to 12]IF APTSTAT(I).RUNWAY(J).CLOSED.DEP 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_RVR runways]

RVRCK = ''B;

```

LOOP;    [K = 1 to 12]
          IF RWYEQP(I).RUNWAY(K).RVR NE (2) ' '
            THEN RVRCK = RVRCK CONCATENATE '1'B;
            ELSE RVRCK = RVRCK CONCATENATE '0'B;
          ENDLOOP;
          RVRCK = RVRCK CONCATENATE BZERO;
END RUNWAY_CLOSURE_ELIGIBILITY_SET_UP;

```

PROCESS RUNWAY_CLOSURE_ELIGIBILITY_CHECK

[This process determines eligibility of configurations with runways closed]

IF ((CNFGRQ(J).ID) AND (INELIG)) NE (24) '0'BTHEN 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 (CNVTAPT(1).WX.CEIL LT 200.) AND ((CNFGRQ(J).ID) AND (PARAPP(3)) NE PARAPP(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_CHECKPROCESS BELOW_.5_VIS_PLUS_NON_RVR_CONFIGURATION_ELIGIBILITY_CHECK

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

IF (CNVTAPT(1).WX.VIS LT .5) AND ((CNFGRQ(J).ID) AND (RVRCK) NE '0' BTHEN 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;

2-111

```
PROCESS BELOW_800_CEIL_2_VIS_ELIGIBILITY_CHECK
  [This process determines eligibility of configurations with ceiling and visibility below 800 and 2
  respectively]

  IF CNVTAPT(I).WX.CEIL LT 800) OR (CNVTAPT(I).WX.VIS LT 2)
    THEN
      FLAG = '0'B;
      REPEAT UNTIL (FLAG = '1'B); [K = 1 to 6]
        IF ((CNFGRQ(J).ID) AND (PARAPP(K)) EQ PARAPP(K))
          THEN FLAG = '1'B;
        ENDREPEAT;
      IF FLAG NE '1'B
        THEN EFLAG = '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 (CNVTAPT(I).WX.CEIL LT 1000.) OR (CNVTAPT(I).WX.VIS LT 3)) AND
 (((CNFGRQ(I).ID) AND (TRIPAPP(1)) EQ TRIPAPP(1)) OR
 ((CNFGRQ(J).ID) AND (TRIPAPP(2)) EQ TRIPAPP(2)) OR
 ((CNFGRQ(I).ID) AND (TRIPAPP(3)) EQ TRIPAPP(3)) OR
 ((CNFGRQ(J).ID) AND (TRIPAPP(4)) EQ TRIPAPP(4)) OR
 ((CNFGRQ(I).ID) AND (TRIPAPP(5)) EQ TRIPAPP(5)) OR
 ((CNFGRQ(J).ID) AND (TRIPAPP(6)) EQ TRIPAPP(6)) OR
 ((CNFGRQ(I).ID) AND (TRIPAPP(7)) EQ TRIPAPP(7)) OR
 ((CNFGRQ(J).ID) AND (TRIPAPP(8)) EQ TRIPAPP(8)) OR
 ((CNFGRQ(J).ID) AND (DUALAPP(1)) EQ DUALAPP(1)) OR
 ((CNFGRQ(J).ID) AND (DUALAPP(2)) EQ DUALAPP(2)))

THEN EFLAG = '1'B;

{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;

PROCESS BETWEEN 4800 TO 200 CEILING AND 5 TO .25 VISIBILITY PLUS EQUIPMENT OUTAGE ELIGIBILITY CHECK
 [This process determines eligibility of configurations with ceiling between 200 and 4800, visibility between .25 and 5 and certain equipment inoperable]

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

THEN

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

IF (RWYEQP(I).RUNWAY(K).GS NE (2) ' ' OR
 RWYEQP(I).RUNWAY(K-1).GS NE (2) ' ' OR
 RWYEQP(I).RUNWAY(K).OM NE (2) ' ' OR
 RWYEQP(I).RUNWAY(K-1).OM NE (2) ' ' OR
 RWYEQP(I).RUNWAY(K).MM NE (2) ' ' OR
 RWYEQP(I).RUNWAY(K-1).MM NE (2) ' ' OR
 RWYEQP(I).RUNWAY(K).ALS NE (2) ' ' OR
 RWYEQP(I).RUNWAY(K-1).ALS NE (2) ' ') AND
 ((CNFGRQ(J).ID) AND (PARAPP(K/\$TWO)) EQ PARAPP(K/\$TWO))

THEN EFLAG = '1'B;

[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 ((CNVTAPT(I).WX.CEIL LT 4800) AND (CNVTAPT(I).WX.CEIL GE 200)) OR
 ((CNVTAPT(I).WX.VIS LT 5.) AND (CNVTAPT(I).WX.VIS GE .25))

THEN

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

IF (RWYEQP(I).RUNWAY(K).LOC NE (2) ' ') OR (RWYEQP(I).RUNWAY(K-1).LOC NE (2) ' ') AND ((CNFGRQ(J).ID) AND PARAPP(K/\$TWO)) EQ PARAPP(K/\$TWO)

THEN EFLAG = '1'B;

[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(7).SURF NE (2) ' ') OR (APTSTAT(I).RUNWAY(7).BRK NE (2) ' ')) AND
((CNFGRQ(I).ID AND HLDSSHRT(3)) EQ HLDSSHRT(3))

THEN EFLAG = '1'B

IF ((APTSTAT(I).RUNWAY(10).SURF NE (2) ' ') OR (APTSTAT(I).RUNWAY(10).BRK NE (2) ' ')) AND
((CNFGRQ(J).ID AND HLDSSHRT(1)) EQ HLDSSHRT(1)) OR (CNFGRQ(J).ID AND HLDSSHRT(2)) EQ HLDSSHRT(2))

THEN EFLAG = '1'B;

IF (APTSTAT(I).RUNWAY(5).BRK NE (2) ' ') AND ((CNFGRQ(J).ID AND HLDSSHRT(4)) EQ HLDSSHRT(4)) OR
((CNFGRQ(J).ID AND HLDSSHRT(1)) EQ HLDSSHRT(1))

THEN EFLAG = '1'B;

END HOLD_SHORT_ELIGIBILITY_CHECK;

ROUTINE FILES

```

IN (APTSTAT(I), CNVTAPT(I));
INOUT (FILENUM(I), CNDTN(I));
    [This routine determines capacity file number for each configuration and sets CNDTN variable to
    indicate VFR(-1) or IFR(-2)]
LOOP;    [N = 1 to 73] [determine appropriate capacity file]
    IF (CNVTAPT(I).WX.CRIL LT 800) OR (CNVTAPT(I).WX.VIS LT 2)
        THEN
            CNDTN(I) = 2;
            FILENUM(I).CONF(N) = 3;
            REPEAT WHILE (FILENUM(I).CONF(N) EQ 3); [K = 1 to 12]
                IF (APTSTAT(I).RUNWAY(K).BRK EQ 'X '
                    THEN FILENUM(I).CONF(N) = 4;
            ENDREPEAT;
        ELSE
            CNDTN(I) = 1;
            FILENUM(I).CONF(N) = 1;
            REPEAT WHILE (FILENUM(I).CONF(N) EQ 1); [K = 1 to 12]
                IF (APTSTAT(I).RUNWAY(K).BRK EQ 'X '
                    THEN FILENUM(I).CONF(N) = 2;
            ENDREPEAT;
        ENDLOOP;
END FILES;

```

ROUTINE PERCENTIN (CNVTDEM(I), CNFCRQ);INOUT (PRCARR(I);

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

PERFORM INITIALIZATION_(PERCENT);

PRCARR(I).TOTARR = CNVTDEM(I).ARR.TOTAL;

PRCARR(I).TOTDEP = CNVTDEM(I).DEP.TOTAL;

LOOP; [J = 1 to 73]

PRCARR(I).CONF(J).NARRDEM = 0;

PRCARR(I).CONF(J).SARRDEM = 0;

[compute total arrival demand for north complex]

LOOP; [K1 = 1 to 6]IF (CNFCRQ(J).ID AND COMPLEX.ANORTH1(K1)) NE 0THEN

TEMP = COMPLEX.NORTH2 K1);

LOOP; [K2 = 1 to 6]IF CNFCRQ(J).ARR(K2) EQ TEMPTHEN PRCARR.CONF(J).NARRDEM = PRCARR(I).CONF(J).NARRDEM +
DUNGY.ARR(K2);ENDLOOP;ENDLOOP;

[compute total arrival demand for south complex]

PRCARR(I).CONF(J).SARRDEM = PRCARR(I).TOTARR - PRCARR(I).CONF(J).NARRDEM;

[compute total departure demand for north complex]

```

LOOP; [K1 = 1 to 6]
  IF (CNFGRQ(J).ID AND COMPLEX.DNORTH1(K1)) NE 0
    THEN
      TEMP = COMPLEX.NORTH2 K1;
      LOOP; [K2 = 1 to 5]
        IF CNFGRQ(J).DEP(K2) EQ TEMP
          THEN PRCARR(I).CONF(J).DEPDEM = PRCARR(I).CONF(J).NDEPDEM + DUMMY.
              DEP(K2);
        ENDLOOP;
      ENDLOOP;
    ENDLOOP;

[compute total departure demand for south complex]
PRCARR(I).CONF(J).SDEPDEM = PRCARR(I).TOTDEP_PRCARR(I).CONF(J).NDEPDEM;

[compute percentages]
IF (PRCARR(I).CONF(J).NDEPDEM + PRCARR(I).CONF(J).NARRDEM) EQ 0
  THEN PRCARR(I).CONF(J).NPRCNT = 0.5;
  ELSE PRCARR(I).CONF(J).NPRCNT = PRCARR(I).CONF(J).NARRDEM/(PRCARR(I).CONF(J).NARRDEM +
    PRCARR(I).CONF(J).NDEPDEM);

IF (PRCARR(I).CONF(J).SDEPDEM + PRCARR(I).CONF(J).SARRDEM) EQ 0
  THEN PRCARR(I).CONF(J).SPRCNT = .5;
  ELSE PRCARR(I).CONF(J).SPRCNT = PRCARR(I).CONF(J).SARRDEM/(PRCARR(I).CONF(J).SARRDEM +
    PRCARR(I).CONF(J).SDEPDEM);
ENDLOOP;

END PERCENT;

```

PROCESS INITIALIZATION

[This process performs initialization for PERCENT routine]

BZERO = (12) '0'B;

COMPLEX.ANORTH1(1) = '010000000000'B CONCATENATE BZERO;
 COMPLEX.ANORTH1(2) = '000100000000'B CONCATENATE BZERO;
 COMPLEX.ANORTH1(3) = '000001000000'B CONCATENATE BZERO;
 COMPLEX.ANORTH1(4) = '000000010000'B CONCATENATE BZERO;
 COMPLEX.ANORTH1(5) = '000000000100'B CONCATENATE BZERO;
 COMPLEX.ANORTH1(6) = '000000000001'B CONCATENATE BZERO;

COMPLEX.DNORTH(1) = BZERO CONCATENATE '010000000000'B;
 COMPLEX.DNORTH(2) = BZERO CONCATENATE '000100000000'B;
 COMPLEX.DNORTH(3) = BZERO CONCATENATE '000001000000'B;
 COMPLEX.DNORTH(4) = BZERO CONCATENATE '000000010000'B;
 COMPLEX.DNORTH(5) = BZERO CONCATENATE '000000000100'B;
 COMPLEX.DNORTH(6) = BZERO CONCATENATE '000000000001'B;

DUMMY.ARR(1) = CNVTDEM(I).ARR.KURBS;
 DUMMY.ARR(2) = CNVTDEM(I).ARR.CGT;
 DUMMY.ARR(3) = CNVTDEM(I).ARR.PLANT;
 DUMMY.ARR(4) = CNVTDEM(I).ARR.VAINS;
 DUMMY.ARR(5) = CNVTDEM(I).ARR.FARM;
 DUMMY.ARR(6) = CNVTDEM(I).ARR.NKE_A;

DUMMY.DEP(1) = CNVTDEM(I).DEP.NORTH;
 DUMMY.DEP(2) = CNVTDEM(I).DEP.EAST;
 DUMMY.DEP(3) = CNVTDEM(I).DEP.SOUTH;
 DUMMY.DEP(4) = CNVTDEM(I).DEP.WEST;
 DUMMY.DEP(5) = CNVTDEM(I).DEP.NKE_D;

END INITIALIZATION;

ROUTINE QFIX

IN (CNFGRQ(CONFIND(1)));

INOUT (QUELEN);

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

LOOP; [J = 1 to 4]

QUELEN.DEPRUN(J) = CNFGRQ(CONFIND(1)).DEP_RMY(J);

ENDLOOP;

END QFIX;

ROUTINE CAPSAT

IN (PCARR(I), CNFCRQ, CAPFILE, FILENUM(I), ELGBLTY(I));

INOUT (INFORM(I));

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

SWITCH(1) = 2;

SWITCH(2) = 1;

IF PCARR(I).TOTARR + PCARR(I).TOTDEP EQ 0.

THEN ATOTPRC = .5;

ELSE ATOTPRC = PCARR(I).TOTARR/(PCARR(I).TOTARR + PCARR(I).TOTDEP);

LOOP; [N = 1 to 73] [up to 73 eligible configurations]

IF SUBSTR(ELGBLTY(I).ID,N,1) EQ '0's

THEN [if the configuration N is eligible]

FLAG = 0;

PERFORM CAPACITY_CURVE_SELECTION;

IF FLAG EQ 0

THEN [demand balancing]

CALL DBAL;

IN (CAPACITY,PAIR,PCARR(I).TOTARR,PCARR(I).TOTDEP,
PCARR(I).CONF(N).HARRDEN,PCARR(I).CONF(N).HDEPDEN);

OUT
(PCARR(I).CONF(N).BMPRCNT,PCARR(I).CONF(N).BSPRCNT,PCARR(I).CONF(N).
BHARRDEN,PCARR(I).CONF(N).BSHARRDEN,PCARR(I).CONF(N).BHDEPDEN,PCARR
(I).CONF(N).BSDEPDEN);

[this routine balances demand]

IF PCARR(I).CONF(N).BMPRCNT GE 0.

```

      THEN [not saturated]
        PRCNT = PRCARR(I).CONF(N).BNPCNT; [using balanced percentage
        of arrivals]

        PERFORM NORTH_COMPLEX_CAPACITY_CALCULATIONS;
        PRCNT = PRCARR(I).CONF(N).BSPCNT; [using balanced percentage
        of arrivals]

        PERFORM SOUTH_COMPLEX_CAPACITY_CALCULATIONS;
        ARECAP = INFORM(I).CONF(N).MARECAP + INFORM(I).CONF(N).SARECAP;
        DEPCAP = INFORM(I).CONF(N).NDEPCAP + INFORM(I).CONF(N).SDEPCAP;

      ELSE [saturated]
        PRCNT = PRCARR(I).CONF(N).NPCNT; [using unbalanced
        percentage of arrivals]

        PERFORM NORTH_COMPLEX_CAPACITY_CALCULATIONS;
        PRCNT = PRCARR(I).CONF(N).SPCNT; [using unbalanced
        percentage of arrivals]

        PERFORM SOUTH_COMPLEX_CAPACITY_CALCULATIONS;
        ARECAP = INFORM(I).CONF(N).NDEPCAP + INFORM(I).CONF(N).SARECAP;
        DEPCAP = INFORM(I).CONF(N).NDEPCAP + INFORM(I).CONF(N).SDEPCAP;

    IF FLAG EQ 1
      THEN [north only configuration]
        PERFORM NORTH_ONLY_CAPACITY_COMPUTATION;

      ELSE
        PERFORM SOUTH_ONLY_CAPACITY_COMPUTATION;

        PERFORM CONSTRAIN_CAPACITY_OF_ENTIRE_AIRPORT;
        PERFORM SATURATION_COMPUTATION;
        PERFORM CHANGE_DUE_TO_DEMAND_BALANCING_COMPUTATION;
        PERFORM FINAL_SATURATION_CHECK;

    ELSE [for ineligible configuration]
      INFORM(I).CONF(N).CAPACITY = -1.0;
      INFORM(I).CONF(N).INDEX = 999;

    ENDLLOOP;

  END CAPSAT;

```


PROCESS CAPACITY_CURVE_SELECTION

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

INFORM(I).CONF(N).INDEX = N;

L = FILENUM(I).CONF(N);

M(1) = CNFGREQ(N).NORTH; [obtain north and south complex indices]

M(2) = CNFGREQ(N).SOUTH;

LOOP; [R = 1 to 2] [retrieve north and south capacity curves from CAPFILE]IF M(R) NE 0THENLOOP; [J = 1 to 14]

CAPACITY(R,J) = CAPFILE(L).KEY(M(R)).CAP(J);

ENDLOOP;

PAIR(R) = CAPFILE(L).KEY(M(R)).PNUM;

ELSE 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, PRCNT);

OUT (ACAP, DCAP);

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

INFORM(I).CONF(N).MARRCAP = ACAP;

INFORM(I).CONF(N).MDEPCAP = DCAP;

END NORTH_COMPLEX_CAPACITY_CALCULATIONS;

PROCESS SOUTH_COMPLEX_CAPACITY_CALCULATIONS
[This process computes capacity of south complex]

CALL CAPCAL;

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

OUT (ACAP, DCAP);

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

INFORM(I).CONF(N).SARRCAP = ACAP;

INFORM(I).CONF(N).SDEPCAP = 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(I).TOTARR + PRCARR(I).TOTDEF) LE (ACAP + DCAP)THEN [not saturated]

PRCARR(I).CONF(N).BNPRCNT = ATOTPRC;

PRCARR(I).CONF(N).BSPRCNT = 0.;

PRCARR(I).CONF(N).BNARRDEN = PRCARR(I).TOTARR;

PRCARR(I).CONF(N).BNDEPDEN = PRCARR(I).TOTDEF;

PRCARR(I).CONF(N).BSARRDEN = 0.;

PRCARR(I).CONF(N).BSDEPDEN = 0.;

INFORM(I).CONF(N).NARRCAP = ACAP;

INFORM(I).CONF(N).NDEPCAP = DCAP;

INFORM(I).CONF(N).SARRCAP = 0.;

INFORM(I).CONF(N).SDEPCAP = 0.;

ARRCAP = ACAP;

DEPCAP = DCAP;

ELSE [saturated]

PRCARR(I).CONF(N).BNPRCNT = -1.0;

PRCARR(I).CONF(N).BSPRCNT = -1.0;

PRCARR(I).CONF(N).BNARRDEN = -1.0;

PRCARR(I).CONF(N).BNDEPDEN = -1.0;

```
PRCARR(I).CONF(N).BSARDEM = -1.0;  
PRCARR(I).CONF(N).BSDEPDEN = -1.0;  
  
INFORM(I).CONF(N).MARCAP = ACAP;  
INFORM(I).CONF(N).NDEPCAP = DCAP;  
  
INFORM(I).CONF(N).SARCAP = 0.;  
INFORM(I).CONF(N).SDEPCAP = 0.;  
  
ARRCAP = ACAP;  
DEPCAP = DCAP;
```

```
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), ATOTPRC);OUT (ACAP, DCAP);IF (PRCARR(I).TOTARR + PRCARR(I).TOTDEP) LE (ACAP + DCAP)THEN [not saturated]

```

PRCARR(I).CONF(N).BMPCNT = 0.;
PRCARR(I).CONF(N).BSPCNT = ATOTPRC;
PRCARR(I).CONF(N).BMARRDEM = 0.;
PRCARR(I).CONF(N).BMDEPDEN = 0.;
PRCARR(I).CONF(N).BSARRDEM = PRCARR(I).TOTARR;
PRCARR(I).CONF(N).BSDEPDEN = PRCARR(I).TOTDEP;

```

```

INFORM(I).CONF(N).MARRCAP = 0.;
INFORM(I).CONF(N).NDEPCAP = 0.;
INFORM(I).CONF(N).SARRCAP = ACAP;
INFORM(I).CONF(N).SDEPCAP = DCAP;

```

```

ARRCAP = ACAP;
DEPCAP = DCAP;

```

ELSE [saturated]

```

PRCARR(I).CONF(N).BMPCNT = -1.0;
PRCARR(I).CONF(N).BSPCNT = -1.0;
PRCARR(I).CONF(N).BMARRDEM = -1.0;
PRCARR(I).CONF(N).BMDEPDEN = -1.0;
PRCARR(I).CONF(N).BSARRDEM = -1.0;
PRCARR(I).CONF(N).BSDEPDEN = -1.0;
INFORM(I).CONF(N).MARRCAP = 0.;
INFORM(I).CONF(N).NDEPCAP = 0.;
INFORM(I).CONF(N).SARRCAP = ARRCAP;
INFORM(I).CONF(N).SDEPCAP = DEPCAP;
ARRCAP = ACAP;
DEPCAP = DCAP;

```

END SOUTH ONLY CAPACITY CALCULATIONS:

```

PROCESS CONSTRAIN_CAPACITY_OF_ENTIRE_AIRPORT
  {This process constrain capacity for entire airport}
  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);

  INFORM(1).CONF(N).CAPACITY = ARRCAP + DEPCAP; [total airport capacity (constrained)]
END CONSTRAIN_CAPACITY_OF_ENTIRE_AIRPORT;

```

PROCESS SATURATION COMPUTATION

[This process computes saturation level]

IF PRCARR(I).CONF(N).BNPRCNT GE 0.

THEN [not saturated]

[for north complex]

DEM = PRCARR(I).CONF(N).BNARRDEM + PRCARR(I).CONF(N).BNDEPDEN;
CAP = INFORM(I).CONF(N).NARRCAP + INFORM(I).CONF(N).NDEPCAP;

IF CAP GT 0.

THEN INFORM(I).CONF(N).NSAT = DEM/CAP;
ELSE INFORM(I).CONF(N).NSAT = 1.0;

[for south complex]

DEM = PRCARR(I).CONF(N).BSARRDEM + PRCARR(I).CONF(N).BSDEPDEN;
CAP = INFORM(I).CONF(N).SARRCAP + INFORM(I).CONF(N).SDEPCAP;

IF CAP GT 0.

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

INFORM(I).CONF(N).SATURATION = (PRCARR(I).TOTARR + PRCARR(I).TOTDEP/INFORM(I).
CONF(N).CAPACITY;

ELSE [saturated]

[for north complex]

DEM = PRCARR(I).CONF(N).NARRDEM + PRCARR(I).CONF(N).NDEPDEN;
CAP = INFORM(I).CONF(N).NARRCAP + INFORM(I).CONF(N).NDEPCAP;

```
IF CAP GT 0.  
  THEN INFORM(I).CONF(N).NSAT = DEN/CAP;  
  ELSE INFORM(I).CONF(N).NSAT = -1.0;  
DEN = PCARR(I).CONF(N).SARRDEN + PCARR(I).CONF(N).SDEPDEN;  
CAP = INFORM(I).CONF(N).SARRCAP + INFORM(I).CONF(N).SDEPCAP;  
IF CAP GT 0.  
  THEN INFORM(I).CONF(N).SSAT = DEN/CAP;  
  ELSE INFORM(I).CONF(N).SSAT = -1.0;  
INFORM(I).CONF(N).SATURATION = (PCARR(I).TOTARR + PCARR(I).TOTDEF)/INFORM(I).  
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 PRCARR(I).CONF(N).BNPRCNT GE 0.THEN [not saturated]INFORM(I).CONF(N).CHANGEMARR = FLOOR(PRCARR(I).CONF(N).MARRDEM + .5) -
FLOOR(PRCARR(I).CONF(N).BMARRDEM + .5);INFORM(I).CONF(N).CHANGEMDEP = FLOOR(PRCARR(I).CONF(N).MDEPDEM + .5) -
FLOOR(PRCARR(I).CONF(N).BMDEPDEM + .5);ELSE [saturated]

INFORM(I).CONF(N).CHANGEMARR = 0.;

INFORM(I).CONF(N).CHANGEMDEP = 0.;

END CHANGE_DUE_TO_DEMAND_BALANCING_COMPUTATION;PROCESS FINAL_SATURATION_CHECK

[This process checks saturation level and set appropriate variables]

IF INFORM(I).CONF(N).CAPACITY LT (PRCARR(I).TOTARR + PRCARR(I).TOTDEP)THEN

PRCARR(I).CONF(N).BNPRCNT = -1.0;

PRCARR(I).CONF(N).BSPRCNT = -1.0;

PRCARR(I).CONF(N).BMARRDEM = -1.0;

PRCARR(I).CONF(N).BMDEPDEM = -1.0;

PRCARR(I).CONF(N).BSARRDEM = -1.0;

PRCARR(I).CONF(N).BSDEPDEM = -1.0;

END FINAL_SATURATION_CHECK;

ROUTINE CAPCALIN (PNUM, 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 (PNUM EQ 1) OR (PRCNT EQ 0.)THEN [one pair of points only, or no arrivals]ACAP = CAPFILE(1);
DCAP = CAPFILE(2);ELSEIF PRCNT LT 1.0THEN [some departures]FLAG = 0;
R = PRCNT/(1.0 - PRCNT);
RATIO2 = CAPFILE(1)/CAPFILE(2);REPEAT WHILE (FLAG EQ 0); {I = 2 to PNUM}RATIO1 = RATIO2;
RATIO2 = CAPFILE (2*I_1)/CAPFILE(2*I);IF (R GE RATIO1) AND (R LE RATIO2)THENS = (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;

2-132

```
ENDREPEAT;  
IF FLAG EQ 0  
  THEN [lots of departures]  
    ACAP = CAPFILE (PWUN*2-1);  
    DCAP = ACAP/R;  
  ELSE [all arrivals]  
    ACAP = CAPFILE(PWUN*2-1);  
    DCAP = 0.;  
END CAPCAL;
```

ROUTINE DBAL

IN (CAPACITY,PAIR,PCARR(I).TOTARR,PCARR(I).TOTDEP,PCARR(I).CONF(N).NARRDEM,PCARR(I).CONF(N).NDEPDEM);

OUT (PCARR(I).CONF(N).BNPRCNT,PCARR(I).CONF(N).BSPRCNT,PCARR(I).CONF(N).BNARRDEM,PCARR(I).CONF(N).BSARRDEM,PCARR(I).CONF(N).BNDEPDEM,PCARR(I).CONF(N).BSDEPDEM);
 [This routine performs demand balancing]

PNUM = PAIR;
 A = PCARR(I).TOTARR;
 D = PCARR(I).TOTDEP;
 NARRDEM = PCARR(I).CONF(N).NARRDEM;
 NDEPDEM = PCARR(I).CONF(N).NDEPDEM;
 BNPRCNT = PCARR(I).CONF(N).BNPRCNT;
 BSPRCNT = PCARR(I).CONF(N).BSPRCNT;
 BNARRDEM = PCARR(I).CONF(N).BNARRDEM;
 BNDEPDEM = PCARR(I).CONF(N).BNDEPDEM;
 BSARRDEM = PCARR(I).CONF(N).BSARRDEM;
 BSDEPDEM = PCARR(I).CONF(N).BSDEPDEM;

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

IF (A EQ 0.) AND (D EQ 0.)

THEN [if both arrival and departure demands are zero]

BNPRCNT = .5;
 BSPRCNT = .5;
 BNARRDEM = 0;
 BNDEPDEM = 0;
 BSARRDEM = 0;
 BSDEPDEM = 0;

ELSE

LOOP; [K = 1 to 2]

IF PNUM(K) GT 1

```

THEN
  LOOP;    [J = 1 to 2*PNUM(K)]
            C1(J) = CAPACITY(K,J);
  ENDLOOP;
  IF C1(2*PNUM(K)) NE 0.
    THEN
      P1 = PNUM(K)+1;
      C1(2*P1-1) = C1(2*P1-3);
      C1(2*P1) = 0;
    ELSE P1 = PNUM(K);
  ELSE
    P1 = 1;
    C1(1) = CAPACITY(K,1);
    C1(2) = CAPACITY(K,2);
  IF C1(1) GT 0. OR (C1(2) GT 0)
    THENIF PNUM(SWITCH(K)) GT 1)
      THEN
        LOOP; {J = 1 to 2*PNUM(SWITCH(K))}
              C2(J) = CAPACITY (SWITCH(K),J);
        ENDLOOP;
        IF C2(2*PNUM(SWITCH(K))) NE 0.
          THEN
            P2 = PNUM(SWITCH(K)) + 1;
            C2(2*P2-1) = C2(2*P2-3);
            C3(2*P2) = 0.;
          ELSE P2 = PNUM(SWITCH(K));

```

```

ELSE
  P2 = 2;
  IF CAPACITY(SWITCH(K),1) GT 0.
    THEN
      C2(1) = 0.;
      C2(2) = 0.;
      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.;
      C2(4) = 0.;
  IF (C2(2) GT 0.) OR (C2(3) GT 0)
    THEN
      IF K EQ 1
        THEN
          ARDEM = NARDEM;
          DEPDEN = NDEPDEN;
        ELSE
          ARDEM = A - NARDEM;
          DEPDEN = D - NDEPDEN;
      CALL RMO;
      IN (C1, C2, P1, P2, A, D, ARDEM, NDEPDEN);
      OUT (RHOMIN, INDEX);
      [this routine performs demand balancing algorithm]
      IF RHOMIN LT SAT

```

```

      THEN
        CORNER = INDEX;
        CURVE = K;
        SAT = RHOMIN;

    ENDOLOOP;

    [check integer aircraft]

    IF CURVE = 0
      THEN
        ENPRCNT = -1.0;
        ESPPCNT = -1.0;
        EMARRDEN = -1.0;
        ESARRDEN = -1.0;
        ENDEPDEN = -1.0;
        ESDEPDEN = -1.0;

      ELSEIF (CORNER GT PNUM (CURVE)) AND (PNUM(CURVE) GT 1)
        THEN
          XARR = SAT * CAPACITY (CURVE, 2*PNUM(CURVE)-1);
          YDEP = 0.;

      ELSEIF (CORNER GT PNUM (CURVE)) AND (PNUM(CURVE) EQ 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,*);
      CAPFILE2 = CAPACITY(SWITCH(CURVE),*);

      X = FLOAT(FLOOR(XARR + .5));
      Y = FLOAT(FLOOR(YDEP + .5));

      IF (X + Y) GT 0.

```

```

THEN
  PCT1 = X/(X + Y);
  CALL CAPCAL;
  IN (PNUM(CURVE),CAPFILE1, PCT1);
  OUT (ACAP1, DCAP1);
  S1 = (X + Y)/(ACAP1 + DCAP1);
ELSE S1 = 0.;
IF (A - X GT 0.) AND (D - Y GE 0.)
  THEN
    PCT2 = (A - X)/A + D - X - Y);
    CALL CAPCAL;
    IN (PNUM(SWITCH(CURVE)),CAPFILE2, PCT2);
    OUT (ACAP2, DCAP2);
    S2 = (A + D - X - Y)/(ACAP2 + DCAP2);
  ELSEIF (A - X) GT 0.
    THEN
      PCT2 = 1.0;
      CALL CAPCAL;
      IN (PNUM(SWITCH(CURVE)),CAPFILE2, PCT2);
      OUT (ACAP2, DCAP2);
      S2 = (D - Y)/(ACAP2 + DCAP2);
  IF (S1 LE 1.0) AND (S2 LE 1.0)

```


THENIF CURVE EQ 1

THEN

BNPCNT = PCT1;
 BSPCMT = PCT2;
 BNARDEM = X;
 BSARDEM = A - X;
 BNDEPDEM = Y;
 BSDEPDEM = D - Y;

ELSE

BNPCNT = PCT1;
 BSPCMT = PCT2;
 BNARDEM = X;
 BSARDEM = A - X;
 BNDEPDEM = Y;
 BSDEPDEM = D - Y;

ELSE

PERFORM LOWER_LEFT_LOOKUP;

IF (S1 LE 1.0) AND (S2 LE 1.0)

THEN PERFORM ASSIGNMENT;

ELSE PERFORM UPPER_LEFT_LOOKUP;

IF (S1 LE 1.0) AND (S2 LE 1.0)

THEN PERFORM ASSIGNMENT;

ELSE PERFORM UPPER_RIGHT_LOOKUP;

IF (S1 LE 1.0) AND (S2 LE 1.0)

THEN PERFORM ASSIGNMENT;

ELSE PERFORM LOWER_RIGHT_LOOKUP;

IF (S1 LE 1.0) AND (S2 LE 1.0)

THEN PERFORM ASSIGNMENT;

ELSE

BSPRCHT = -1.0;
BSPRCHT = -1.0;
BSARDEN = -1.0;
BSDEPDEN = -1.0;
BSARDEN = -1.0;
BSDEPDEN = -1.0;

END DBAL;

PROCESS ASSIGNMENT

IF CURVE EQ 1

THEN

BNPCNT = PCT1;
 BSPCNT = PCT2;
 BNARDEM = X;
 BSARDEM = A - X;
 BNDEPDEM = Y;
 BSDEPDEM = D - Y;

ELSE

BNPCNT = PCT1;
 BSPCNT = PCT2;
 BNARDEM = X;
 BSARDEM = A - X;
 BNDEPDEM = Y;
 BSDEPDEM = D - Y;

END ASSIGNMENT;

2-141

```
PROCESS LOWER_LEFT_LOOKUP
  X = FLOAT(FLOOR(XARR));
  Y = FLOAT(FLOOR(YDEP));

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

```
PROCESS UPPER_LEFT_LOOKUP
  X = FLOAT(CEIL(XARR));
  IF (X + Y) GT 0.
    THEN
      PCT1 = X/(X + Y);
      CALL CAPCAL;
      IN (PNUM(CURVE),CAPFILE1, PCT1);
      OUT (ACAP1, DCAP1);
      S1 = (X + Y)/(ACAP1 + DCAP1);
    ELSE S1 = 0.;
  IF (A - X GT 0.)
    THEN
      PCT2 = (A - X)/(A + D - X - Y);
    ELSE PCT2 = 0.;
  CALL CAPCAL;
  IN (PNUM(SWITCH(CURVE)), CAPFILE2, PCT2);
  OUT (ACAP2, DCAP2);
  S2 = (A + D - X - Y)/(ACAP2 + DCAP2);
END UPPER_LEFT_LOOKUP;
```

2-143

```
PROCESS UPPER_RIGHT_LOOKUP
  Y = FLOAT(CELL(YDEP));
  IF (X + Y) GT 0.
    THEN
      PCT1 = X/(X + Y);
      CALL CAPCAL;
      IN (PNUM(CURVE),CAPFILE1, PCT1);
      OUT (ACAP1, DCAP1);
      S1 = (X + Y)/(ACAP1 + DCAP1);
    ELSE S1 = 0;
  IF (A - X GT 0.) AND (D - Y GE 0.)
    THEN
      PCT2 = (A_X)/(A + D_X_Y);
      CALL CAPCAL;
      IN (PNUM(SWITCH(CURVE)),CAPFILE2,PCT2);
      OUT (ACAP2,DCAP2);
      S2 = (A + D_X_Y)/(ACAP2 + DCAP2);
    ELSEIF (A_X) GT 0.
```

```
THEN
  PCT2 = 1.0;
  CALL CAPCAL;
  IN (PNUM(SWITCH(CURVE)),CAPFILE2, PCT2);
  OUT (ACAP2, DCAP2);
  S2 = (A - X)/(ACAP2 + DCAP2);
ELSE
  PCT2 = 0.;
  CALL CAPCAL;
  IN (PNUM(SWITCH(CURVE)),CAPFILE2, PCT2);
  OUT (ACAP2, DCAP2);
  S2 = (D - Y)/(ACAP2 + DCAP2);
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 (PNUM(CURVE),CAPFILE1,PCT1);  
      OUT (ACAP1,DCAP1);  
      S1 = (X + Y)/(ACAP1 + DCAP1);  
    ELSE S1 = 0;  
  IF (D - Y GE 0.)  
    THEN  
      PCT2 = (A - X)/(A + D - X - Y);  
    ELSE PCT2 = 1.0;  
  CALL CAPCAL;  
  IN (PNUM(SWITCH(CURVE)),CAPFILE2,PCT2);  
  OUT (ACAP2,DCAP2);  
  S2 = (A + D - X - Y)/(ACAP2 + DCAP2);  
END LOWER_RIGHT_LOOKUP;
```


ROUTINE RHOIN (C1, C2, P1, P2, A, A, ARDEM, DEPDEN)OUT (RHOMIN, INDEX);

[This routine performs demand balancing algorithm]

RHOMIN = 1.0;

DELTA = 999999.;

FLAG = 0.;

INDEX = 0;

LOOP; [J = P1 to 1 by -1]IF FLAG EQ 0THEN K = 1;ELSE FLAG = 0;IF C2(2*K-1) GT 0.THEN RATIO2 = C2(2*K)/C2(2*K-1);ELSEIF C2(2*K) GT 0.THEN RATIO2 = 99999;ELSE RATIO2 = 0.;REPEAT UNTIL (K EQ P2) OR (FLAG EQ 1);

NUM = C2(2*K+2) - (C2(2*K);

DEN = C2(2*K+1) - C2(2*K-1);

IF DEN GT 0.THEN

M = NUM/DEN;

B = C2(2*K) - M*C2(2*K-1);

T = C1(2*J) - M*C1(2*J-1)+B;

```
IF T GT 0.  
  THEN R = (D - M*A)/T;  
  ELSEIF D GT 0.;  
    THEN R = 1.0;  
    ELSE R = A/C2(2*K+1) + C1(2*J-1));  
ELSE  
  T = C2(2*K+1) + C1(2*J-1);  
  IF T GT 0.  
    THEN R = A/T;  
    ELSEIF 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 = 999999.  
X = A - R*C1(2*J-1);  
Y = D - R*C1(2*J);  
IF ABS(Y) LT .001  
  THEN Y = 0.;  
IF ABS(X) GE .0001;  
  THEN RATIO = Y/X;
```

```

ELSE
  RATIO = 999999;
  X = 0;

IF RATIO1 GT RATIO2
  THENIF (RATIO LE RATIO1) AND (RATIO GE RATIO2)
    THEN FLAG = 1;
    ELSEIF (R*C2(2*K-1) LE X) AND (X LE R*C2(2*K+1) AND (R*C2(2*K) GE Y) AND
      (Y GE R*C2(2*K+2)))
      THEN FLAG = 1;
    IF FLAG EQ 1
      THENIF (R GT 0) AND (R LE 1.0) AND (R LT RMIN + .01)
        THEN
          DEN = R*(C1(2*J) + C1(2*J-1));
          IF ABS(R - RMIN) LT .01
            THENIF (ABS(ARRDEN + DEPDEN - DEN) LT DELTA)
              THEN
                DELTA = ABS(ARRDEN + DEPDEN - DEN);
                RMIN = R;
                INDEX = J;
              ELSE
                DELTA = ABS(ARRDEN + DEPDEN - DEN);
                RMIN = R;
                INDEX = J;
            ELSE
              K = K + 1;
            ENDREPEAT
          ENDLOOP;
        END RHO;

```

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.

[LOCAL VARIABLES]

STRUCTURE MESSAGE_MAKER [data structure where current log messages that appear on O'Hare status summary screen are stored]

GROUP TABLE(108) [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 INKEEP [an integer array of size 108; is used as a flag, if nth element of this array is equal to 1 it implies that nth 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 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 ARRNUM(3) [pointers for current configuration's arrival runways data fields]

PTR DEPRUN(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 DMS manual initialized to string of (32) '1'B]
ENDSTRUCTURE;

ROUTINE HSTAT

IN (OHSTAT, APTSTAT(1), INFORM(1), CNFGRQ (CONFIND(1)), CONFIND(1), EQPLOG, CNVTBQP, WXLOG, CNVTWX, SURFLOG, CNVTSRF);

INOUT (OLDMES, RSTATUS);

[This routine prepares information used on O'Hare status summary screen, and stores that information in structure OHSTAT]

\$FOUR = 4;

\$THREE = 3;

OHSTAT.WX.CEIL = APTSTAT(1).WX.CEIL; [set prevailing ceiling]

OHSTAT.WX.VIS = APTSTAT(1).WX.VIS; [set prevailing visibility]

OHSTAT.WIND.DIR = APTSTAT(1).WIND.DIR; [set wind direction]

OHSTAT.WIND.VEL = APTSTAT(1).WIND.VEL [set wind velocity]

LOOP; [J = 1 to 3]

OHSTAT.ARR(J) = CNFGRQ (CONFIND(1)).ARR_RWT(J); [set current operating configuration's arrival runways]

ENDLOOP;

LOOP; [J = 1 to 4]

OHSTAT.DEF(J) = CNFGRQ (CONFIND(1)).DEF_RWT(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]

OHSTAT.CAPACITY = (5) ' ';

OHSTAT.PCT_HC = (3) ' ';

OHSTAT.MSG = SUBSTR(OHSTAT.MSG, 1, 24) CONCATENATE '***CURRENT CONFIGURATION IS INELIGIBLE***'

ELSE

OHSTAT.CAPACITY = F(INFORM(1).CONF(CONFIND(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;

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

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

LOOP; [J = 1 to 13]

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

ENDLOOP;

LOOP; (J = 1 to 108) [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 = OHSTAT.MSG; [create ALT1 and ALT2 to use for 'return to
ALT2 = OHSTAT.SCROLL; [previously stored data' function]
REPEAT UNTIL (RSTATUS NE PF12);
    OHSTAT.MSG = ALT1;
    OHSTAT.SCROLL = ALT2;
    REPEAT UNTIL (RSTATUS NE PF1);
        CALL HSCREEN;
        IN (OHSTAT,MESSAGE_MAKER,COUNT,INKEEP);
        INOUT (RSTATUS);
        [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]

ENDLOOP; [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: [J = 1 to 15; up to 15 equipment planning log messages]

IF EQPLOG.TABLE(J).RWY 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).RWY CONCATENATE (5) ' ';

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

THEN AUX1 = AUX1 CONCATENATE EQ PLOG.TABLE(J). EQUIPMENT;

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

THEN AUX2 = EQPLOG.TABLE(J).REMARKS

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

THEN

COUNT = COUNT + 1; [increment message counter]

MESSAGE MAKER.TABLE(COUNT).TIME = CHVTBQP.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 = CNVTREQ.TABLE(J).RTS;
[construct message with RTS time]

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

ENDLOOP;

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 to 13; up to 13 weather and wind planning log messages]

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

THENIF (WXLOG.TABLE(J).CEIL NE (5) ' ') OR (WXLOG.TABLE(J).VIS NE (5) ' ')

THEN

AUX = ' ';

COUNT = COUNT + 1; [increment message counter]

MESSAGE_MAKER.TABLE(COUNT).TIME = CNVTWX.TABLE(J).TIME;

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

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

THEN

AUX-AUX CONCATENATE 'WX ' CONCATENATE WXLOG.TABLE(J).CEIL CONCATENATE (3) ' ' CONCATENATE; WXLOG.TABLE(J).VIS CONCATENATE (5)

ELSEIF WXLOG.TABLE(J).CEIL NE (5) ' ';

THEN AUX-AUX CONCATENATE 'CEIL ' CONCATENATE WXLOG.TABLE(J).CEIL CONCATENATE (13) ' ';

ELSE AUX-AUX CONCATENATE 'VIS ' CONCATENATE WXLOG.TABLE(J).VIS CONCATENATE (13) ' ';

IF WXLOG.TABLE(J).REMARKS NE (35) ' ';

THEN AUX-AUX CONCATENATE WXLOG.TABLE(J).REMARKS;

MESSAGE_MAKER.TABLE(COUNT).MSG =AUX

[message is constructed with weather information and stored]

IF (WXLOG.TABLE(J).DIR NE (5) ' ') OR (WXLOG.TABLE(J).VEL(5) ' ';

```

      THEN
        AUX = ' '
        COUNT = COUNT + 1; [increment message counter]
        MESSAGE_MAKER.TABLE(COUNT).TIME = CHVTWX.TABLE(J).TIME;

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

        IF <<(WXLOG.TABLE(J).DIR NE(5) ' ') AND(WXLOG.TABLE(J).VEL NE(5) ' ')>>

          THEN
            AUX-AUX CONCATENATE 'WIND ' CONCATENATE WXLOG.TABLE(J).DIR
            CONCATENATE WXLOG.TABLE(J).VEL CONCATENATE (7) ' ';

            ELSEIF WXLOG.TABLE(J).DIR NE(5) ' '

              THEN
                AUX-AUX CONCATENATE 'DIR ' CONCATENATE WXLOG.TABLE(J).DIR
                CONCATENATE (12) ' ';

                ELSE
                  AUX-AUX CONCATENATE 'VEL ' CONCATENATE WXLOG.TABLE(J).VEL
                  CONCATENATE (12) ' ';

        IF WXLOG.TABLE(J).REMARKS NE(35) ' '

          THEN
            AUX-AUX CONCATENATE WXLOG.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=CHVTSRF.TABLE(J).TIME;

AUX=(3) ' ' CONCATENATE SURFLOG.TABLE(J).TIME CONCATENATE (3) ' ' CONCATENATE
SURFLOG.TABLE(J).SURF CONCATENATE (5) ' ' CONCATENATE SURFLOG.TABLE(J).SURF
CONCATENATE (12) ' ' '

IF SURFLOG.TABLE(J).REMARKS NE (27) ' '

THEN AUX=AUX CONCATENATE SURFLOG.TABLE(J).REMARKS;

MESSAGE_MAKER.TABLE(COUNT).MSG=AUX;

[a message is constructed with surface conditions information]

IF SURFLOG.TABLE(J).BRK NE (5) ' '

THEN

AUX = ' ';

COUNT=COUNT + 1; [increment message counter]

MESSAGE_MAKER.TABLE(COUNT).TIME= CHVTSRF.TABLE(J).TIME;

AUX=(3) ' ' CONCATENATE SURFLOG.TABLE(J).TIME CONCATENATE (3) ' ' CONCATENATE
SURFLOG.TABLE(J).SURF CONCATENATE (5) ' ' CONCATENATE SURFLOG.TABLE(J).BRK
CONCATENATE (12) ' ' '

IF SURFLOG.TABLE(J).REMARKS NE (27) ' '

```

        THEN AUX=AUX CONCATENATE SURFLOG.TABLE(J).REMARKS;
MESSAGE MAKER.TABLE(COUNT).MSG=AUX;
    [a 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 = CHVTSEF.TABLE(J).TIME;
AUX=(3)' ' CONCATENATE SURFLOG.TABLE(J).TIME CONCATENATE (3)' ' CONCATENATE
SURFLOG.TABLE(J).RWY CONCATENATE (5)' 'CONCATENATE SURFLOG.TABLE(J).CLOSED
CONCATENATE ' CLOSED ' ;
        IF SURFLOG.TABLE(J).REMARKS NE(27)' '
            THEN AUX=AUX CONCATENATED 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='';
COUNT=COUNT + 1; [increment message counter]
MESSAGE MAKER.TABLE(COUNT).TIME = CHVTSEF.TABLE(J).TIME;
AUX=(3)' ' CONCATENATE SURFLOG.TABLE(J).TIME CONCATENATE SURFLOG.TABLE(J).RWY
CONCATENATE 5' ' CONCATENATE
SURFLOG.TABLE(J).OPEN CONCATENATE ' OPEN'
        IF SURFLOG.TABLE(J).REMARKS NE (27)' '
            THEN
                AUX=AUX CONCATENATE SURFLOG.TABLE(J).REMARKS;
MESSAGE MAKER.TABLE(COUNT).MSG = AUX;
    [a 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;

  ENDLOOP

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]

INKEEP = 0; [initialize INKEEP]

IF (OLDMES.TABLE(1).TIME EQ 0) AND (OLDMES.TABLE(1).MSG EQ (80) ' ')

THEN

LOOP: [J = 1 TO COUNT]

INKEEP(J) = 1; [no old messages, all new messages will be highlighted]

ENDLOOP;

ELSE [new message is compared with old message table and any new entry is flagged]

INKEEP = 0;

L = 1;

J = 1; [initialization]

IND = 0;

REPEAT WHILE (J LE COUNT)

IF MESSAGE_MAKER.TABLE(J).TIME LT OLDMES.TABLE(L).TIME

THEN [if times of messages are not equal, there exist a new message]

IND = IND + 1;

INKEEP(J) = 1;

J = J + 1;

ELSEIF

MESSAGE_MAKER.TABLE(J).TIME EQ OLDMES.TABLE(L).TIME;

THENIF

MESSAGE_MAKER.TABLE(J).MSG NE OLDMES.TABLE(L).MSG

```
THEN
  IND-IND +1;
  INKEEP(J)=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 OLD MES 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]

    OLD MES.TABLE(J).TIME = MESSAGE MAKER.TABLE(J).TIME;
    OLD MES.TABLE(J).MSG = MESSAGE MAKER.TABLE(J).MSG;

  ENDLOOP;

END OLD_MESSAGE_TABLE_GENERATION;
```

ROUTINE HSCREENIN (OHSTAT, MESSAGE_MAKER, COUNT, INKEEP);INOUT (RSTATUS);

[This routine controls O'Hare status summary screen]

CHR PNAME [character variable of length 8 containing name of DMS panel initialized to 'OHSTATUS';
name of panel that controls O'Hare status summary screen]INT CURSOR [integer variable containing cursor's position on screen]BITS DM(28) [8 bit variable of data mask used in DMS]INT CONVERT_SCROLL [numerical value of scroll data field]INT TM [integer representing current time]

DM = FLDDEF; [set data masks to default intensity (normal)]

DM(28) = FLDHIGH; [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 (RSTAT);

TAUX = GMT; [current time is obtained]

PERFORM CHARACTER_TO_NUMERICAL_CONVERSION_OF_TIME;PERFORM DIVISION_OF_LOG_MESSAGES_BASED_ON_CURRENT_TIME;REPEAT UNTIL (RSTATUS NE ENTER);IF COUNT NE 0 [if there are log messages]

```

THEN
    PERFORM SCROLL_FUNCTION_SET_UP;
    PERFORM SET_UP_SCREEN_LOG_MESSAGES;
ELSE [if no log messages exist]
    PERFORM SET_UP_NULL_SCREEN_MESSAGE;
PERFORM DISPLAY_PANEL;
IF RSTATUS EQ ENTER
    THEN
        DM = FLDDEF; [set data masks to default intensity (normal)]
        DM(28) = FLDHIGH; [set message data mask to high intensity]
        CALL MCHECK;
        INOUT (OHSTAT, 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 OHSTAT.MSG NE 'DATA ENTERED'
            THEN
                DM(CURSOR)=FLDHIGH;
                [highlight erroneous entry]
                CONVERT_SCROLL = 0;
            ELSE
                OHSTAT.SCROLL = (4) ' '; [Scroll data field is blanked]
                OHSTAT.MSG = 'DATA ENTERED AT' CONCATENATE CMT;
                [time is adjusted accordingly on screen message line]
    ENDREPEAT;
END HSCREEN;

```

```
PROCESS SET_UP_SCREEN_POINTERS (HSTAT)
  [This process sets up screen pointers for DMS use]

  OH_LOADLIST.WX.CEIL = ADDR(OHSTAT.WX.CEIL);
  OH_LOADLIST.WX.VIS = ADDR(OHSTAT.WX.VIS);
  OH_LOADLIST.WIND.DIR = ADDR(OHSTAT.WIND.DIR);
  OH_LOADLIST.WIND.VEL = ADDR(OHSTAT.WIND.VEL);

  LOOP;    [J = 1 TO 3]

    OH_LOADLIST.ARRRUN(J) = ADDR(OHSTAT.ARR(J));

  ENDLOOP;

  LOOP;    [J = 1 TO 4]

    OH_LOADLIST.DEPRUN(J) = ADDR(OHSTAT.DRP(J));

  ENDLOOP;

  OH_LOADLIST.CAP = ADDR(OHSTAT.CAPACITY);
  OH_LOADLIST.CAPCV = ADDR(OHSTAT.PCT_HC);
  OH_LOADLIST.SCROLL = ADDR(OHSTAT.SCROLL);

  LOOP;    [J = 1 TO 13]

    OH_LOADLIST.LOG_MSG(J) = ADDR(OHSTAT.LOG_MSG(J));

  ENDLOOP;

  OH_LOADLIST.MSG = ADDR(OHSTAT.MSG);

END SET_UP_SCREEN_POINTERS (HSTAT);
```

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_NUMERICAL;

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 (R + E) LT DELTA
THEN IF R LT 5
THEN
 E = DELTA-R;
IF E 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 + E; [NEXT points to number of last message to be displayed]
END DIVISION OF LOG MESSAGES BASED ON CURRENT TIME;

2-171

PROCESS SCROLL_FUNCTION SET UP

[This process performs scrolling function associated with O'Hare 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)

THEN

N = 1;
L = NEXT;

ELSEIF

(NEXT GT COUNT) AND (INDEX LE COUNT)

THEN

N = INDEX;
L = COUNT;

END SCROLL_FUNCTION_SET_UP;

PROCESS SET UP SCREEN LOG MESSAGES

[This process determines screen messages to be displayed based on pointers calculated before, and sets up pointers for DMS use; and constructs message headings]

FLAG = '0'B;

K = 1;

IF MESSAGE_MAKER.TABLE(M).TIME LE TM
[if first message time is less than current time]

THEN

OHSTAT.LOG.MSG(K) = (22)' ' CONCATENATE

'****RECENT CHANGES FROM' CONCATENATE SUBSTR(MESSAGE_MAKER.TABLE(1).MSG,3,5) CONCATENATE
' '; [set up header]

OH_LOADLIST.LOG MSG(K) = ADDR(OHSTAT.LOG MSG(K));
[set up pointer for header for DMS use]

ELSE

OHSTAT.LOG MSG(K) = 19' ' CONCATENATE '****EXPECTED CHANGES THROUGH' CONCATENATE
SUBSTR(MESSAGE_MAKER.TABLE(COUNT).MSG,3,5) CONCATENATE ' '; [set up header]

OH_LOADLIST.LOG MSG(K) = ADDR(OHSTAT.LOG MSG(K));
[set up pointer for header for DMS use]

FLAG = '1'B;

REPEAT WHILE (K LE 13); [J = M TO L]

K = K + 1;

IF MESSAGE_MAKER.TABLE(J).TIME LE TM

THEN

IF INKEEP(J) EQ 1 [if it is a new message]

```

      THEN DM(K+14) = FLDHIGH;
      [highlight message]

      OHSTAT.LOG_MSG(K) = MESSAGE_MAKER.TABLE(J).MSG;
      OH_LOADLIST.LOG_MSG(K) = ADDR(OHSTAT.LOG_MSG(K));

    ELSE

      IF FLAG EQ '0' B

        THEN

          DM(K+14) = FLDDARK; [darken screen]
          K = K+1;

          OHSTAT.LOG_MSG(K) = (19) ' ' CONCATENATE '****EXPECTED CHANGES THROUGH'
          CONCATENATE SUBSTR(MESSAGE_MAKER.TABLE(COUNT).MSG,3,5) CONCATENATE '
          *****';

          OH_LOADLIST.LOG_MSG(K)=ADDR(OHSTAT.LOG_MSG(K));

          FLAG='1'B;
          K = K + 1;

      IF INKEEP(J) EQ 1 [if it is a new message]

        THEN DM(K+14) = FLDHIGH; [highlight message]

        OHSTAT.LOG_MSG(K)=MESSAGE_MAKER.TABLE(J).MSG;
        OH_LOADLIST.LOG_MSG(K) = ADDR(OHSTAT.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)=FLDDARK;
          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]

OHSTAT.LOG_MSG(1)=(21)' ' CONCATENATE '****NO LOG ENTRIES HAVE BEEN MADE****';

OH_LOADLIST.LOG_MSG(1)=ADDR(OHSTAT.LOG_MSG(1));

END SET_UP_NULL_SCREEN_MESSAGE;

PROCESS DISPLAY_PANEL

[This process invokes PDISPLAY preprocessor which in turn interfaces with DMS panel manager and displays screen]

END DISPLAY_PANEL;

ROUTINE NCHECKINOUT (ONSTAT, 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]

DECIMAL = '.';

ERR1 = 'NUMERIC INPUT REQUIRED'

ERR3 = 'NO DECIMAL POINTS ALLOWED'

NOERR = 'DATA ENTERED';

ONSTAT.MSG = NOERR;

ON CONVERSION BEGIN;

[ON CONVERSION is a PL/I feature, it is invoked if a character data is detected in numerical field]

ONSTAT.MSG = ERR1;

RETURN;

Get STRING (ONSTAT.SCROLL) EDIT (CONVERT_SCROLL);

[conversion to numerical data from character data]

IF VERIFY (DECIMAL, ONSTAT.SCROLL) = 0THEN ONSTAT.MSG = ERR3;END NCHECK;

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]

BITS FENCE [32 bit variable as prescribed by DMS manual initialized to string of (32) '1's]

ENDSTRUCTURE;

2-179

ROUTINE LOGS

INOUT (RSTATUS);

[This routine invokes planning log selection screen]

[initialize screen to blanks]

LOGNUM.CHOICE (1) = ' ';

LOGNUM.CHOICE (2) = ' ';

LOGNUM.CHOICE (3) = ' ';

LOGNUM.CHOICE (4) = ' ';

LOGNUM.MSG = (80) ' ';

REPEAT UNTIL (RSTATUS NE PF12)

LOGNUM_DATA = LOGNUM;

REPEAT UNTIL (RSTATUS NE PF2)

CALL LSCREEN;

INOUT (LOGNUM_DATA, RSTATUS);

[This routine controls planning log selection screen]

ENDREPEAT;

ENDREPEAT;

END LOGS;

ROUTINE LSCREENINOUT (LOGNUM_DATA, RSTATUS);

[This routine controls planning log selection screen]

CHR PNAME [character variable of length 8 containing name of DMS panel initialized to 'LOGGER',
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 DMS]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 (RSTATUS NE ENTER)PERFORM DISPLAY_PANELIF RSTATUS EQ PA1THEN stop;IF RSTATUS NE ENTERTHEN;ELSEDM = FLDDDEF; [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.MSG NE 'DATA ENTERED'
  THEN DM(CURSOR) = FLDHIGH; [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.MSG NE 'NOERR'
      THEN DM(CURSOR) = FLDHIGH;
      [highlights invalid entry]
      ELSE
        CALL LUPDATE;
        INOUT (LOGNUM_DATA, RSTATUS);
        [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 RSTATUS]
    ENDREPEAT
  END LSCREEN

```

AU-A127 828

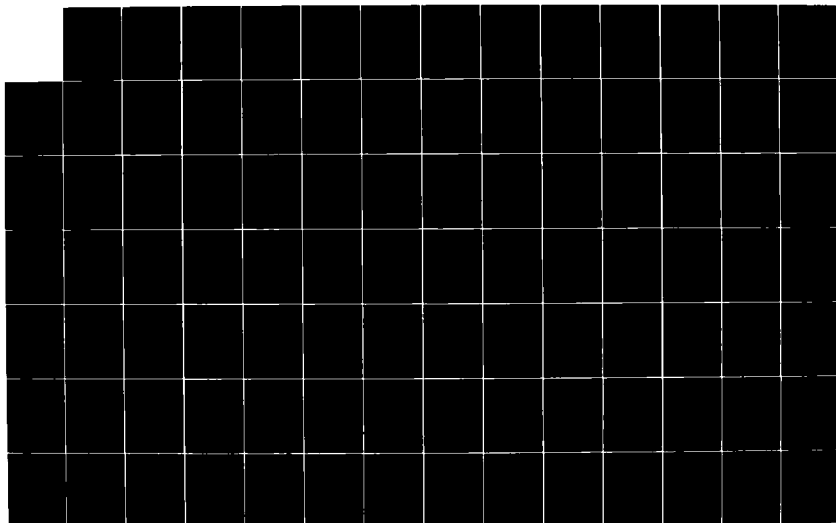
SOFTWARE DESCRIPTION FOR THE O'HARE RUNWAY
CONFIGURATION MANAGEMENT SYSTE..(U) MITRE CORP MCLEAN
VA METREX DIV 5 KAYOUSSI OCT 82 MTR-82M125-VOL-2
FAA-EM-82-28-VOL-2 DTFA01-81-C-10003

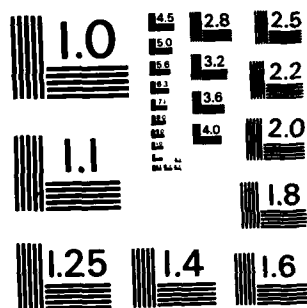
F/G 17/7

NL

3/5

UNCLASSIFIED





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

```
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 (LOGNUM_DATA.CHOICE(J));
  ENDLOOP;
  LOG_LOADLIST.MSG = ADDR (LOGNUM_DATA.MSG);
END SET_UP_SCREEN_POINTERS (LOGS);
```

ROUTINE LCHECKINOUT (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 0THEN 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 LVALIDINOUT (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.MSG = '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 2THEN LOGNUM_DATA.MSG = 'SELECT ONLY ONE PLANNING LOG';ELSE CURSOR = 1;END LVALID;

ROUTINE LUPDATEINOUT (LOGNUM_DATA, RSTATUS)

[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 RSTATUS]

IF LOGNUM_DATA.CHOICE(1) NE ' 'THEN RSTATUS = PF13; [weather and wind planning log is selected]IF LOGNUM_DATA.CHOICE(2) NE ' 'THEN RSTATUS = PF14; [airport planning log is selected]IF LOGNUM_DATA.CHOICE(3) NE ' 'THEN RSTATUS = PF15; [equipment planning log is selected]IF LOGNUM_DATA.CHOICE(4) NE ' 'THEN RSTATUS = PF16; [demand planning log is selected]ELSE LOGNUM_DATA.MSG = (80) ' '; [message line is blanked]END LUPDATE;

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 WXLOG

[This structure is similar to WXLOG used as a working area within screen routine]

ENDSTRUCTURE;

STRUCTURE CHVRT_WX LIKE CHVTWX

[This structure is similar to CHVTWX 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]

BITS FENCE [32 bit variable as prescribed by DMS manual, initialised to string of (32)'1's]

ENDSTRUCTURE;

ROUTINE WLOGINOUT (WXLOG, CNVTWX, RSTATUS);

[This routine invokes weather and wind planning log screen]

REPEAT UNTIL (RSTATUS NE FF12);

WX_DATA = WXLOG;

CNVTWX = CNVTWX;

CALL WSCREEN;INOUT (WX_DATA, CNVTWX, RSTATUS);

[This routine controls weather and wind planning log screen]

ENDREPEAT;IF SUBSTR (WX_DATA.MSG,1,12) EQ 'DATA ENTERED'THEN

WXLOG = WX_DATA;

CNVTWX = CNVTWX;

END WLOG;

ROUTINE WSCREENINOUT (WX_DATA, CNVRT_WX, RSTATUS)

[This routine controls weather and wind planning log screen]

CHR PNAME [character variable of length 8 containing name of DMS panel initialized 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 WX_DATAENDSTRUCTURE;

DM = FLDDEF; [set data masks to default intensity (normal)]

DM(79) = FLDHIGH; [set message data mask to high intensity]

CURSOR = 61; [set cursor to position 61; first data field used]

AUX_DATA = WX_DATA;

PERFORM SET_UP_SCREEN_POINTERS_(WLOG);REPEAT UNTIL (RSTATUS NE ENTER);PERFORM DISPLAY_PANEL;IF RSTATUS EQ PA1THEN stop;IF RSTATUS NE ENTERTHEN NEW_DATA = AUX_DATA;

```

ELSE
  DM = FLDDDEF;
  DM(79) = FLDHIGH;

  CALL WCHECK

  INOUT (WX_DATA,CNVRT 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]

  IF WX_DATA.MSG NE 'DATA ENTERED'
    THEN DM(CURSOR) = FLDHIGH;
    ELSE
      CALL WVALID;
      INOUT (WX_DATA,CNVRT 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]

      IF WX_DATA.MSG NE 'DATA ENTERED'
        THEN DM(CURSOR) = FLDHIGH;
        ELSE
          CALL WUPDATE;
          INOUT (WX_DATA,CNVRT WX);
          [This routine is performed only when there are no errors
          committed on screen, it sorts log entries on screen based
          on time]

          WX_DATA.MSG = 'DATA ENTERED AT' CONCATENATE GMT;
          AUX_DATA = WX_DATA;

  ENOREPEAT;
END WSCREEN;

```

2-191

```
PROCESS SET_UP_SCREEN_POINTERS_(WLOG)
  [This process sets up screen pointers for DMS use]
  LOOP: [J = 1 TO 13]
    WX_LOADLIST.TABLE(J).TIME=ADDR(WX_DATA.TABLE(J).TIME);
    WX_LOADLIST.TABLE(J).CEIL=ADDR(WX_DATA.TABLE(J).CEIL);
    WX_LOADLIST.TABLE(J).VIS=ADDR(WX_DATA.TABLE(J).VIS);
    WX_LOADLIST.TABLE(J).DIR=ADDR(WX_DATA.TABLE(J).DIR);
    WX_LOADLIST.TABLE(J).VEL=ADDR(WX_DATA.TABLE(J).VEL);
    WX_LOADLIST.TABLE(J).REMARKS=ADDR(WX_DATA.TABLE(J).REMARKS);
  ENDLOOP;
  WX_LOADLIST.MSG=ADDR(WX_DATA.MSG);
END SET_UP_SCREEN_POINTERS_(WLOG);
```


ROUTINE WCHECK

INOUT (WX_DATA,CNVRT_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 TIME_DATA_FIELD_ERROR_CHECK;

EXITIF [errors detected]

CURSOR = CURSOR + 1;

PERFORM CKIL_DATA_FIELD_ERROR_CHECK;

EXITIF [error detected]

CURSOR = CURSOR + 1;

PERFORM VIS_DATA_FIELD_ERROR_CHECK;

EXITIF [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 CNVRT_WX.TABLE(J).TIME); [conversion from character data to numerical data]

IF VERIFY('-',WX_DATA.TABLE(J).TIME) EQ 0THEN WX_DATA.MSG = ERR2;ELSEIF VERIFY('.', WX_DATA.TABLE(J).TIME) EQ 0THEN 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(CNVRT_WX.TABLE(J).CEIL)
[conversion from character data to numerical data]IF VERIFY('-',WX_DATA.TABLE(J).CEIL) EQ 0THEN WX_DATA.MSG = ERR2;ELSEIF VERIFY('.',WX_DATA.TABLE(J).CEIL) EQ 0THEN 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 (CNVRT 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 (CNVRT 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(CNVRT 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 WVALIDINOUT (WX_DATA,CNVRT 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;

CURSOR = 60;

WX_DATA.MSG = '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).CEIL 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;
 CNVRT WX.TABLE(J).TIME = 9999;

ELSE

CURSOR = CURSOR + 1;

IF (WX_DATA.TABLE(J).TIME NE (4) ' ') AND
 (WX_DATA.TABLE(J).CEIL 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 (WX_DATA.TABLE(J).TIME EQ (4)' ') AND
(WX_DATA.TABLE(J).CEIL NE (5)' ') OR
(WX_DATA.TABLE(J).VIS NE (5)' ') OR
(WX_DATA.TABLE(J).DIR NE (5)' ') OR
(WX_DATA.TABLE(J).VEL NE (5)' ') OR
(WX_DATA.TABLE(J).REMARKS NE (35)' ')

THEN [time missing]
  WX_DATA.MSG = 'SPECIFY TIME ASSOCIATED WITH ENTRIES';

ELSE [check time]
  PERFORM TIME_CHECK;

  EXITIF [time entry is erroneous]

  PERFORM LEFT_ZERO_PADDING_ON_TIME_ENTRY;

  IF WX_DATA.MSG EQ 'DATA ENTERED'
    THEN
      CURSOR = CURSOR + 1;

      IF WX_DATA.TABLE(J).CEIL NE (5)' '
        THEN
          PERFORM RIGHT_JUSTIFY_CEIL_DATA_ENTRY;

          CURSOR = CURSOR + 1;

          IF WX_DATA.TABLE(J).VIS NE (5)' '
            THEN
              PERFORM RIGHT_JUSTIFY_VIS_DATA_ENTRY;

              CURSOR = CURSOR + 1;

              IF WX_DATA.TABLE(J).DIR NE (5)' '

```

THEN IF CNVRT_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
NE (5) ' ';

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 WVALID;


```

PROCESS TIME_CHECK
  [This process checks validity of time entry]
  HOUR = FLOOR (FLOAT(CNVRT_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 = CNVRT_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_WX.TABLE(J).TIME), $FOUR);  
K = 0;  
FILL = '';  
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;  
  
WX_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_VIS_DATA_ENTRY;

```
PROCESS LEFT ZERO PADDING ON WIND DIRECTION ENTRY  
[This process pads wind direction entry with leading zeroes]  
  
C = F(CNVRT_WX.TABLE(J).DIR,$FOUR);  
K = 1;  
FILL = ' ';  
FLAG = '0'B;  
  
REPEAT WHILE (FLAG EQ '0' B) REPEAT UNTIL (K EQ 3);  
    K = K + 1;  
    IF SUBSTR(C,K,1) = 'b'  
        THEN FILL = FILL CONCATENATE '0'B;  
        ELSE FLAG = '1'B;  
  
ENDREPEAT;  
  
IF FLAG EQ '0'B  
    THEN K = 4;  
  
WX_DATA.TABLE(J).DIR = FILL CONCATENATE SUBSTR(C,K,5-K);  
  
END LEFT_ZERO_PADDING_ON_WIND_DIRECTION_ENTRY;
```

PROCESS LEFT_JUSTIFY_REMARKS_DATA_ENTRY;
 [This process left-justifies remarks]

K = 0;
 FLAG = '0'B;

REPEAT WHILE (FLAG EQ '0' B) REPEAT UNTIL (K EQ 35);

K = K + 1;

IF SUBSTR(WX_DATA.TABLE(J).REMARKS, K, 1) NE ' '

THEN LAG = '1'B;

ENDREPEAT;

C = SUBSTR (WX_DATA.TABLE(J).REMARKS, K, 36-K);

WX_DATA.TABLE(J).REMARKS = SUBSTR (C, 1, 35);

END LEFT_JUSTIFY_REMARKS_DATA_ENTRY;

PROCESS RIGHT_JUSTIFY_VEL_DATA_ENTRY

C = F(CNVET WX.TABLE(J).VEL, \$FOUR)

WX_DATA.TABLE(J).VEL = SUBSTR(C, 1, 4) CONCATENATE 'b';

END RIGHT_JUSTIFY_VEL_DATA_ENTRY;

ROUTINE WUPDATEINOUT (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;
CTEMP1 = CNVRT_WX.TABLE(L).TIME;
CTEMP2 = CNVRT_WX.TABLE(L).CEIL;
CTEMP3 = CNVRT_WX.TABLE(L).VIS;
CTEMP4 = CNVRT_WX.TABLE(L).DIR;
CTEMP5 = 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;  
CNVRT WX.TABLE(L+1).TIME = CTEMP1;  
CNVRT WX.TABLE(L+1).CEIL = CTEMP2;  
CNVRT WX.TABLE(L+1).VIS = CTEMP3;  
CNVRT WX.TABLE(L+1).DIR = CTEMP4;  
CNVRT WX.TABLE(L+1).VEL = CTEMP5;
```

```
L = L - 1;
```

```
ELSE L = 10;
```

```
ENDREPEAT;
```

```
ENDLOOP;
```

```
END WUPDATE;
```

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 CNVRT_SRF LIKE CNVTSRF

[This structure is similar to CNVTSRF used as 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 RWY [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, initialized to string of (32) '1's]

ENDSTRUCTURE;

ROUTINE SLOGINOUT (SURFLOG, CNVTSRF, RSTATUS)

[This routine invokes airport surface and runway planning log screen]

REPEAT UNTIL (RSTATUS NE PF12);

SURF_DATA = SURFLOG;

CNVRT_SRF = CNVTSRF;

CALL SSCREEN;INOUT (SURF_DATA, CNVRT_SRF, 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;

CNVTSRF = CNVRT_SRF;

END SLOG;ROUTINE SSCREENINOUT (SURF_DATA, CNVRT_SRF, RSTATUS);

[This routine controls airport surface and runway planning log screen]

CHR PNAME [character variable of length 8 containing name of DMS 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 DMS]STRUCTURE AUX_DATA LIKE SURF_DATAENDSTRUCTURE;

```

DM = FLDDEF; [set data masks to default intensity (normal)]
DM(92) = FLDHIGH; [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 = FLDDEF;
      DM(92) = FLDHIGH;
      CALL SCHCK;
      INOUT (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) = FLDHIGH;
      ELSE

```

```

CALL SVALID;

INOUT (SURF_DATA, CNVRT_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 DM(CURSOR) = FLDN1GH;
ELSE
CALL SUPDATE;

INOUT (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]

SURF_DATA.MSG = 'DATA ENTERED AT ' CONCATENATE GMT;
AUX_DATA = SURF_DATA;

ENDREPEAT;

END SSCREEN;

```

```
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).RWY = ADDR (SURF_DATA.TABLE(J).RWY);
    SURF_LOADLIST.TABLE(J).SURF = ADDR (SURF_DATA.TABLE(J).SURF)
    SURF_LOADLIST.TABLE(J).BRAK = ADDR (SURF_DATA.TABLE(J).BRAK)
    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 SCHECKINOUT (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]

ERR1 = 'NUMERIC INPUT REQUIRED';

ERR2 = 'NON-NEGATIVE INPUT REQUIRED';

ERR3 = 'NO DECIMAL POINTS ALLOWED';

RWYERR = '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) = ' ';

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 = '0'B;  
REPEAT WHILE (FLAG EQ '0'B); {K = 1 To 14}  
  IF RWYID (K) EQ C  
    THEN FLAG = '1'B;  
  
ENDREPEAT;  
  
IF FLAG = '0'B;  
  THEN SURF_DATA.MSG = RWYERR;  
  
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 (CNVRT_SF.TABLE(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 SVALIDINOUT (SURF_DATA, CNVRT_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]

\$FOUR = 4;

CURSOR = 70;

SURF_DATA.MSG = 'DATA ENTERED';

REPEAT WHILE (SURF_DATA.MSG EQ 'DATA ENTERED'); [J = 11 To 13]

IF (SURF_DATA.TABLE(J).TIME EQ (4) ' ') AND
 (SURF_DATA.TABLE(J).RWY EQ (3) ' ') AND
 (SURF_DATA.TABLE(J).SURF EQ (5) ' ') AND
 (SURF_DATA.TABLE(J).BRK EQ (5) ' ') AND
 (SURF_DATA.TABLE(J).CLOSED EQ (6) ' ') AND
 (SURF_DATA.TABLE(J).OPEN EQ (6) ' ') AND
 (SURF_DATA.TABLE(J).REMARKS = (27) ' ')

THEN [All entries are blank]
 CURSOR = CURSOR + 7
 CNVRT_SRF.TABLE(J).TIME = 9999;

ELSE
 CURSOR = CURSOR + 1;

IF (SURF_DATA.TABLE(J).TIME NE (4) ' ') AND
 (SURF_DATA.TABLE(J).RWY NE (3) ' ') AND
 (SURF_DATA.TABLE(J).SURF EQ (5) ' ') AND
 (SURF_DATA.TABLE(J).BRK 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.MSG = 'ADDITIONAL INFORMATION REQUIRED FOR THIS ENTRY';

```

ELSE
  CURSOR = CURSOR + 1;
  IF (SURF_DATA.TABLE(J).TIME EQ (4) ' ') AND
    ((SURF_DATA.TABLE(J).RWY NE (3) ' ') OR
    (SURF_DATA.TABLE(J).SURF NE (5) ' ') OR
    (SURF_DATA.TABLE(J).BRK 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).RWY EQ (3) ' ') AND
      ((SURF_DATA.TABLE(J).TIME NE (4) ' ') OR
      (SURF_DATA.TABLE(J).SURF NE (5) ' ') OR
      (SURF_DATA.TABLE(J).BRK 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;
      EXITIF [time entry is erroneous]
      PERFORM LEFT_ZERO_PADDING_ON_TIME_ENTRY;
      IF SURF_DATA.MSG EQ 'DATA ENTERED'

```

```

                THEN
                    CURSOR = CURSOR + 6;
                    PERFORM RIGHT_JUSTIFY_RWY_DATA_ENTRY;
                    PERFORM LEFT_JUSTIFY_SURF_DATA_ENTRY;
                    PERFORM LEFT_JUSTIFY_BRK_DATA_ENTRY;
                    PERFORM LEFT_JUSTIFY_CLOSED_DATA_ENTRY;
                    PERFORM LEFT_JUSTIFY_OPEN_DATA_ENTRY;
                    PERFORM LEFT_JUSTIFY_REMARKS_DATA_ENTRY;

                ENDREPEAT;
                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 = CNVRT_SRF.TABLE(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(FLOAT(CNVRT_SRF.TABLE(J).TIME), $FOUR);  
K = 0;  
FILL = '';  
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;  
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'B;

REPEAT WHILE (FLAG EQ '0'B) REPEAT UNTIL (K EQ 5);

K = K + 1;

IF SUBSTR(SURF_DATA.TABLE(J).SURF, K, 1) NE ' '

THEN FLAG = '1'B;

ENDREPEAT;

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'B;

REPEAT WHILE (FLAG EQ '0'B) REPEAT UNTIL (K EQ 5);

K = K + 1;

IF SUBSTR(SURF_DATA.TABLE(J).BRAK, K, 1) NE ' '

THEN FLAG = '1'B;

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-justifies runway closure entry]

K = 0;
 FLAG = '0'B;

REPEAT WHILE (FLAG EQ '0'B) REPEAT UNTIL (K EQ 6);

K = K + 1;

IF SUBSTR (SURF_DATA.TABLE(J).CLOSED, K,1) NE ' '

THEN FLAG = '1'B;

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 LEFT JUSTIFY-OPEN DATA ENTRY

[This process left-justifies runway opening entry]

K = 0

FLAG = '0'B;

REPEAT WHILE (FLAG EQ '0'B) REPEAT UNTIL (K EQ 6);

K = K + 1;

IF SUBSTR(SURF_DATA.TABLE(J).OPEN,K,1) NE ' 'THEN FLAG = '1'B;ENDREPEAT

C = SUBSTR(SURF_DATA.TABLE(J).OPEN,K,7-K) SURF_DATA.TABLE(J).OPEN = SUBSTR(C,1,6);

END LEFT JUSTIFY_OPEN_DATA_ENTRY;PROCESS LEFT JUSTIFY REMARKS DATA ENTRY

[This process left-justifies remarks entry]

K = 0;

FLAG = '0'B

REPEAT WHILE (FLAG EQ '0'B) REPEAT UNTIL (K EQ 27);

K = K + 1;

IF SUBSTR(SURF_DATA.TABLE(J).REMARKS,K,1)NE ' 'THEN FLAG = '1'B;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 SUPDATEINOUT (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+1).TIME LT CNVRT_SRF.TABLE(L).TIMETHEN

TEMP1 = SURF_DATA.TABLE(L).TIME;
 TEMP2 = SURF_DATA.TABLE(L).RWY;
 TEMP3 = SURF_DATA.TABLE(L).SURF;
 TEMP4 = SURF_DATA.TABLE(L).BRAK;
 TEMP5 = SURF_DATA.TABLE(L).CLOSED;
 TEMP6 = SURF_DATA.TABLE(L).OPEN;
 TEMP7 = SURF_DATA.TABLE(L).REMARKS;
 CTMP1 = CNVRT_SRF.TABLE(L).TIME;
 SURF_DATA.TABLE(L).TIME = SURF_DATA.TABLE(L+1).TIME;
 SURF_DATA.TABLE(L).RWY = SURF_DATA.TABLE(L+1).RWY;
 SURF_DATA.TABLE(L).SURF = SURF_DATA.TABLE(L+1).SURF;
 SURF_DATA.TABLE(L).BRAK = SURF_DATA.TABLE(L+1).BRAK;
 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).RWY = TEMP2;
 SURF_DATA.TABLE(L+1).SURF = TEMP3;
 SURF_DATA.TABLE(L+1).BRAK = TEMP4;
 SURF_DATA.TABLE(L+1).CLOSED = TEMP5;

```

SURF_DATA.TABLE(L+1).OPEN = TEMP6;
SURF_DATA.TABLE(L+1).REMARKS = TEMP7;
CHVRT SRF.TABLE(L+1).TIME = CTEMP1;
L = L - 1;

ELSE
L = 10;

ENDREPEAT;

ENDLOOP;

END SUPDATE;
```

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 CMVRT_EQP LIKE CMVTEQP

[this structure is similar to CMVTEQP 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]BITS FENCE [32 bit variable as prescribed by DMS manual, initialized to string of (32) '1's]ENDSTRUCTURE;

2-229

ROUTINE ELOG

INOUT (EQPLOG, CNVTEQP, RSTATUS);

[This routine invokes equipment planning log screen]

REPEAT UNTIL (RSTATUS NE PF12);

EQUIP_DATA = EQPLOG;

CNVRT_EQP = CNVTEQP;

CALL ESCREEN;

INOUT (EQUIP_DATA, CNVRT_EQP, RSTATUS);

[This routine controls equipment planning log screen]

ENDREPEAT;

IF SUBSTR (EQUIP_DATA.MSG, 1, 12) EQ 'DATA ENTERED'

THEN

EQPLOG = EQUIP_DATA;

CNVTEQP = CNVRT_EQP;

END ELOG;

ROUTINE ESCREENINOUT (EQUIP_DATA, CNVRT_EQP, RSTATUS);

[This routine controls equipment planning log screen]

CHR PNAME [character variable of length 8 containing name of DMS panel initialized to 'MYLOG',
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 DMS]STRUCTURE AUX_DATA LIKE EQUIP_DATAENDSTRUCTURE;

DM = FLDDEF; [set data masks to default intensity (normal)]

DM(76) = FLDHIGH; [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 PA1
THEN STOP;IF RSTATUS NE ENTER
THEN EQUIP_DATA = AUX_DATA;
ELSEDM = FLDDEF;
DM(76) = FLDHIGH;
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) = FLDHIGH;

    ELSE

      CALL EVALID;

      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) = FLDHIGH;

    ELSE

      CALL EUPDATE;

      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
      RTS times]

      EQUIP_DATA.MSG= 'DATA ENTERED at 'CONCATENATE GMT;
      AUX_DATA = EQUIP_DATA

      ENDREPEAT

  END ESCREEN;

```



```

PROCESS SET_UP_SCREEN_POINTERS (ELOG)
  [This process sets up screen pointers for DMS 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 ECHECKINOUT (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';
RWYFRR = '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 = INDEX (EQUIP_DATA.TABLE(J).RWY, ' ');  
  
IF (K EQ 0) OR (K EQ 1)  
  THEN C = EQUIP_DATA.TABLE(J).RWY;  
  ELSEIF K EQ 2  
    THEN C = ' ' CONCATENATE SUBSTR (EQUIP_DATA.TABLE(J).RWY, 1, 1) CONCATENATE  
      (EQUIP_DATA.TABLE(J).RWY, 3, 1);  
    ELSE C = ' ' CONCATENATE SUBSTR (EQUIP_DATA.TABLE(J).RWY, 1, 2);  
  
FLAG = '0'B;  
  
REPEAT WHILE (FLAG EQ '0'B); [K = 1 To 14]  
  IF RWYID(K) EQ C  
    THEN FLAG = '1'B  
  
ENDREPEAT;  
  
IF FLAG = '0'B;  
  THEN EQUIP_DATA.MSG = RWYERR;  
  
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 (CNVRT_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 (CNVRT_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 EVALIDINOUT (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]

\$FOUR = 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).RWY EQ (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 [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_RWY_DATA_ENTRY;

IF (EQUIP_DATA.TABLE(J).RWY 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).RWY 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).RWY NE (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).RWY NE (3) ' ') AND
(EQUIP_DATA.TABLE(J).EQUIPMENT NE (11) ' ')
AND ((EQUIP_DATA.TABLE(J).OTS EQ (4) ' ') 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 CNVRT_EQP.TABLE(J).OTS = 2500;

ELSE

PERFORM OTS_TIME_CHECK;

EXITIF [error detected]

PERFORM LEFT_ZERO_PADDING_ON_TIME_ENTRY;

2-239

```
IF EQUIP_DATA.MSG EQ 'DATA ENTERED;  
  THEN  
    CURSOR = CURSOR + 1;  
    IF EQUIP_DATA.TABLE(J).RTS EQ (4)''  
      THEN CNVRT.EQP.TABLE(J).RTS = 2500;  
    ELSE  
      PERFORM RTS_TIME_CHECK  
      EXITIF (error detected)  
      IF EQUIP_DATA.MSG EQ 'DATA  
        ENTERED'  
        THENIF  
          (EQUIP_DATA.  
            TABLE(J).OTS NE (4)  
            ' ') AND (EQUIP  
            DATA.TABLE(J). RTS  
            NE (4) '') AND  
            (CNVRT.EQP.TABLE(J).  
            OTS GT CNVRT.EQP.  
            TABLE(J).RTS)  
          THEN EQUIP_DATA.MSG =  
            'TIME FOR RTS MUST BE  
            BLANK OR LATER THAN OTS';  
        ELSE  
          CURSOR = CURSOR + 1;  
          PERFORM LEFT  
            JUSTIFY REMARKS  
            DATA_ENTRY;  
        ENDREPEAT;  
      IF EQUIP_DATA.MSG = 'DATA ENTERED'  
        THEN  
          CURSOR = 61;  
        END EVALID;
```



```

PROCESS RIGHT_JUSTIFY_RWY_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_RWY_DATA_ENTRY;

PROCESS RIGHT_JUSTIFY_EQUIPMENT_DATA_ENTRY
  [This process right-justifies equipment type entry]

  K = 0;
  FLAG = '0'B;

  REPEAT WHILE (FLAG EQ '0'B) REPEAT UNTIL (K EQ 11);
    K = K + 1;

    IF SUBSTR (EQUIP_DATA.TABLE(J).EQUIPMENT, K, 1) NE ' '
      THEN FLAG = '1'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;

```

2-241

```
PROCESS OTS_TIME_CHECK
[This process checks for validity of OTS time entry]
    HOUR = FLOOR (FLOAT(CNVRT_EQP.TABLE(J).OTS)/100.0);
    IF HOUR GT 23
        THEN EQUIP_DATA.MSG = 'HOUR MUST NOT EXCEED 23';
    ELSE MIN = CNVRT_EQP.TABLE(J).OT - HOUR * 100;
        IF MIN GT 59
            THEN EQUIP_DATA.MSG = 'MINUTES MUST NOT EXCEED 59'
    END OTS_TIME_CHECK;

PROCESS LEFT_ZERO_PADDING_ON_OTS_TIME_ENTRY;
[This process pads OTS time entry with leading zeroes]
    C = F(FLOAT(CNVRT_EQP.TABLE(J).OTS), $FOUR);
    K = 0;
    FILL = '-';
    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;
    EQUIP_DATA.TABLE(J).OTS = FILL CONCATENATE SUBSTR (C, K, 5-K);
END LEFT_ZERO_PADDING_ON_OTS_TIME_ENTRY;
```

2-242

```
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 = F(FLOAT(CNVRT_EQP.TABLE(J).RTS), $FOUR);
    K = 0;
    FILL = '';
    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;
    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'B;

REPEAT WHILE (FLAG EQ '0'B) REPEAT UNTIL (K EQ 27);

K = K + 1;

IF SUBSTR(SURF_DATA.TABLE(J).REMARKS, K, 1) NE ' '

THEN FLAG = '1'B;

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 EUPDATEINOUT (EQUIP_DATA, CHVRT_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 12);IF (CHVRT_EQP.TABLE(L+1).OTS LT CHVRT_EQP.TABLE(L).OTS)OR ((CHVRT_EQP.TABLE(L+1).OTS EQ CHVRT_EQP.TABLE(L).OTS) AND (CHVRT_EQP.TABLE(L+1).RTS LT CHVRT_EQP.TABLE(L).RTS))THENTEMP1 = EQUIP_DATA.TABLE(L).RWY;TEMP2 = EQUIP_DATA.TABLE(L).EQUIPMENT;TEMP3 = EQUIP_DATA.TABLE(L).OTS;TEMP4 = EQUIP_DATA.TABLE(L).RTS;TEMP5 = EQUIP_DATA.TABLE(L).REMARKS;CTEMP1 = CHVRT_EQP.TABLE(L).OTS;CTEMP2 = CHVRT_EQP.TABLE(L).RTS;EQUIP_DATA.TABLE(L).RWY = DATA.TABLE(L+1).RWY;EQUIP_DATA.TABLE(L).EQUIPMENT = EQUIP_DATA.TABLE(L+1).EQUIPMENT;EQUIP_DATA.TABLE(L).OTS = EQUIP_DATA.TABLE(L+1).OTS;EQUIP_DATA.TABLE(L).RTS = EQUIP_DATA.TABLE(L+1).RTS;EQUIP_DATA.TABLE(L).REMARKS = EQUIP_DATA.TABLE(L+1).REMARKS;CHVRT_EQP.TABLE(L).OTS = CHVRT_EQP.TABLE(L+1).OTS;CHVRT_EQP.TABLE(L).RTS = CHVRT_EQP.TABLE(L+1).RTS;EQUIP_DATA.TABLE(L+1).RWY = TEMP1;EQUIP_DATA.TABLE(L+1).EQUIPMENT = TEMP2;EQUIP_DATA.TABLE(L+1).OTS = TEMP3;EQUIP_DATA.TABLE(L+1).RTS = TEMP4;EQUIP_DATA.TABLE(L+1).REMARKS = TEMP5;CHVRT_EQP.TABLE(L+1).OTS = CTEMP1;CHVRT_EQP.TABLE(L+1).RTS = CTEMP2;

L = L - 1;

ELSE L = 12;ENDREPEAT;ENDLOOP;END EUPDATE;

2.8 Demand Planning Log Screen

The processing for the Demand Planning Log Screen is described on pages 2-246 to 2-263.

LOCAL VARIABLES

STRUCTURE OAG_DATA LIKE OAGLOG

[This structure is similar to OAGLOG used as a working area within screen routine]

ENDSTRUCTURE;

STRUCTURE CNVRT_OAG LIKE CNVTOAG

[This structure is similar to CNVTOAG 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 GNT [pointer for GNT data field]

PTR TTLARR [pointer for total arrival demand data field]

PTR TTLDEP [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]
BITS FENCE [32 bit variable as prescribed for DMS manual, initialized to string of (32) '1'B]
ENDSTRUCTURE;

2-248

```
ROUTINE GLOG
  IN (OAGDEM);
  INOUT (OAGLOG, CNVTOAG, RSTATUS);
    [This routine invokes demand planning log screen]
  REPEAT UNTIL (RSTATUS NE PF12);
    OAG_DATA = OAGLOG;
    CNVRT_OAG = CNVTOAG;
    CALL GSCREEN;
    IN (OAGDEM);
    INOUT (OAG_DATA, CNVRT_OAG, RSTATUS);
      [This routine controls demand planning log screen]
  ENDREPEAT;
  IF SUBSTR (OAG_DATA.MSG, 1, 12) EQ 'DATA ENTERED'
    THEN
      OAGLOG = OAG_DATA;
      CNVTOAG = CNVRT_OAG;
END GLOG;
```

ROUTINE GSCREENIN (OAGDEM);INOUT (OAG_DATA, CNVRT_OAG, RSTATUS);

[This routine controls demand planning log screen]

CHR PNAME

[character variable of length 8 containing the name of DMS 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 DMS]

STRUCTUREAUX_DATA LIKE OAG_DATAENDSTRUCTURE;

DM = FLDDEF [set data masks to default intensity (normal)]

DM(47) = FLDHIGH; [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 (RSTATUS 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).GMT = ADDR(OAG_DATA.TABLE(HR).GMT);
OAG_LOADLIST.TABLE(L).TTLARR = ADDR(OAG_DATA.TABLE(HR).TTLARR);
OAG_LOADLIST.TABLE(L).TTLDEF = ADDR(OAG_DATA.TABLE(HR).TTLDEF);
OAG_LOADLIST.TABLE(L).KUBBS = ADDR(OAG_DATA.TABLE(HR).KUBBS);
OAG_LOADLIST.TABLE(L).CGT = ADDR(OAG_DATA.TABLE(HR).CGT);
OAG_LOADLIST.TABLE(L).VAINS = ADDR(OAG_DATA.TABLE(HR).VAINS);
OAG_LOADLIST.TABLE(L).FARMN = ADDR(OAG_DATA.TABLE(HR).FARMN);
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 = FLDEF;
        DM(47) = FLDHIGH;
        CALL GCHECK;
        IN (OAGDM, INDEX)
        INOUT (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 GVALID;

IN (INDEX);

INOUT (OAG_DATA, CNVRT_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
    DN(CURSOR) = FLDHIGH;
    CNVRT_OAG.SCROLL = 0;

  ELSE
    OAG_DATA.SCROLL = (2) ' ';
    OAG_DATA.MSG = 'DATA ENTERED AT' CONCATENATE GNT;
    AUX_DATA = OAG_DATA;

  ENDREPEAT;

END GSCREEN;
```

ROUTINE GCHECKIN (OAGDEM, INDEX)INOUT (OAG DATA, 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';

OAG DATA.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]

OAG_DATA.MSG = ERR1;

RETURN;IF X(OAG_DATA.INITIAL) NE 0THEN OAG_DATA.MSG = ERR5;ELSEIF OAG_DATA.INITIAL NE (2) ' 'THEN

OAG DATA = OAG DEM, BY NAME;

OAG_DATA.INITIAL = (2) ' ';

OAG_DATA.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]
  HR = INDEX + L;
  IF HR GT 23
    THEN HR = HR - 24;
    CURSOR = CURSOR + 2;
    PERFORM TTLARR_DATA_FIELD_ERROR_CHECK;
    EXITIF [error detected]
    CURSOR = CURSOR + 1;
    PERFORM TTLDEP_DATA_FIELD_ERROR_CHECK;
    EXITIF [error detected]
    PERFORM KUBBS_DATA_FIELD_ERROR_CHECK;
    EXITIF [error detected]
    CURSOR = CURSOR + 1;
    PERFORM CGT_DATA_FIELD_ERROR_CHECK;
    EXITIF [error detected]
    PERFORM VAINS_DATA_FIELD_ERROR_CHECK;
    EXITIF [error detected]
    CURSOR = CURSOR + 1;
    PERFORM FARMH_DATA_FIELD_ERROR_CHECK;
    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 (CMVRY_OAG.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).TTLDEP) EDIT (CNVRT_OAG.TABLE(HR).TTLDEP);

IF VERIFY ('-', OAG_DATA.TABLE(HR).TTLDEP) EQ 0THEN OAG_DATA.MSG = ERR2;ELSEIF VERIFY ('.', OAG_DATA.TABLE(HR).TTLDEP) EQTHEN 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 0THEN OAG_DATA.MSG = ERR2;ELSEIF VERIFY ('.', OAG_DATA.TABLE(HR).TTLARR) EQ 0THEN 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 (CNVRT_OAG.TABLE(HR).KUBBS);

IF VERIFY ('-', OAG_DATA.TABLE(HR).KUBBS) EQ 0THEN OAG_DATA.MSG = ERR2;ELSEIF VERIFY ('.', OAG_DATA.TABLE(HR).KUBBS) EQ 0THEN 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 (CNVRT_OAG.TABLE(HR).CGT);

IF VERIFY ('-', OAG_DATA.TABLE(HR).CGT) EQ 0THEN OAG_DATA.MSG = ERR2;ELSEIF VERIFY ('.', OAG_DATA.TABLE(HR).CGT) EQ 0THEN 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 (OAG_DATA.TABLE(HR).VAINS) EDIT (CNVRT_OAG.TABLE(HR).VAINS)

IF VERIFY ('.', OAG_DATA.TABLE(HR).VAINS) EQ 0THEN OAG_DATA.MSG = ERR2;ELSEIF VERIFY ('-', OAG_DATA.TABLE(HR).VAINS) EQ 0THEN OAG_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 (OAG_DATA.TABLE(HR).FARM) EDIT (CNVRT_OAG.TABLE(HR).FARM)

IF VERIFY ('.', OAG_DATA.TABLE(HR).FARM) EQ 0THEN OAG_DATA.MSG = ERR2;ELSEIF VERIFY ('-', OAG_DATA.TABLE(HR).FARM) EQ 0THEN OAG_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 (OAG_DATA.TABLE(HR).NORTH) EDIT (CNVET_OAG.TABLE(HR).NORTH)

IF VERIFY ('.', OAG_DATA.TABLE(HR).NORTH) EQ 0THEN OAG_DATA.MSG = ERR2;ELSEIF VERIFY ('-', OAG_DATA.TABLE(HR).NORTH) EQ 0THEN OAG_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 (OAG_DATA.TABLE(HR).EAST) EDIT (CNVET_OAG.TABLE(HR).EAST)

IF VERIFY ('.', OAG_DATA.TABLE(HR).EAST) EQ 0THEN OAG_DATA.MSG = ERR2;ELSEIF VERIFY ('-', OAG_DATA.TABLE(HR).EAST) EQ 0THEN OAG_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 (CNVRT_OAG.TABLE(HR).SOUTH);

IF VERIFY ('.', OAG_DATA.TABLE(HR).SOUTH) EQ 0THEN OAG_DATA.MSG = ERR2;ELSEIF VERIFY ('-', OAG_DATA.TABLE(HR).SOUTH) EQ 0THEN 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 (CNVRT_OAG.TABLE(HR).WEST)

IF VERIFY ('.', OAG_DATA.TABLE(HR).WEST) EQ 0THEN OAG_DATA.MSG = ERR2;ELSEIF VERIFY ('-', OAG_DATA.TABLE(HR).WEST) EQ 0THEN OAG_DATA.MSG = ERR3;END WEST DATA FIELD ERROR CHECK;

ROUTINE CVALIDIN (INDEX);INOUT (OAG_DATA, CNVRT_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';

DMNDERR = 'NUMBER OF AIRCRAFT MUST NOT EXCEED 99';

TTLEERR = '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 + 1

IF HR GT 23THEN HR = HR - 24;

SUM = 0.0;

CURSOR = CURSOR + 4;

IF CNVRT_OAG.TABLE(HR).KUBBS GT 99.0THEN OAG_DATA.MSG = DMNDERR;ELSE

OAG_DATA.TABLE(HR).KUBBS = SUBSTR(F(CNVRT_OAG.TABLE(HR). KUBBS,\$THREE),1,3);

SUM = SUM + CNVRT_OAG.TABLE(HR).KUBBS;

CURSOR = CURSOR + 1;

IF CNVRT_OAG.TABLE(HR).CGT GT 99.0

THEN OAG_DATA.MSG = DMNDERR;

ELSE

OAG_DATA.TABLE(HR).CGT = SUBSTR(F(CNVRT_OAG.TABLE(HR).CGT, \$THREE),1,3);
 SUM = SUM + CNVRT_OAG.TABLE(HR).CGT;
 CURSOR = CURSOR + 1;

IF CNVRT_OAG.TABLE(HR).VAINS GT 99.0

THEN OAG_DATA.MSG = DMNDERR;

ELSE

OAG_DATA.TABLE(HR).VAINS = SUBSTR(F(CNVRT_OAG.TABLE(HR).VAINS,
 \$THREE),1,3);
 SUM = SUM + CNVRT_OAG.TABLE(HR).VAINS;
 CURSOR = CURSOR + 1;

IF CNVRT_OAG.TABLE(HR).FARMH GT 99.0

THEN OAG_DATA.MSG = DMNDERR;

ELSE

OAG_DATA.TABLE(HR).FARMH = SUBSTR(F(CNVRT_OAG.TABLE(HR).
 FARMH,\$THREE),1,3);
 SUM = SUM + CNVRT_OAG.TABLE(HR).FARMH;
 CURSOR = CURSOR - 5;

IF FLOOR(CNVRT_OAG.TABLE(HR).TTLARR) NE SUM
 [check total against sum of individual demand values]

THEN OAG_DATA.MSG = ERR5;

ELSE

OAG_DATA.TABLE(HR).TTLARR = SUBSTR(F(CNVRT_OAG.
 TABLE(HR).TTLARR, \$THREE),1,3);
 SUM = 0.0;
 CURSOR = CURSOR + 6;

IF CNVRT_OAG.TABLE(HR).NORTH GT 99.0

THEN OAG_DATA.MSG = DMNDERR;

ELSE

OAG_DATA.TABLE(HR).NORTH = SUBSTR(F(CNVRT
OAG.TABLE(HR).NORTH,\$THREE),1,3);
SUM = SUM + CNVRT OAG.TABLE(HR).NORTH;
CURSOR = CURSOR + 1;

IF CNVRT_OAG.TABLE(HR).EAST GT 99.0

THEN OAG_DATA.MSG = DMNDERR;

ELSE

OAG_DATA.TABLE(HR).EAST
= SUBSTR(F(CNVRT_OAG.TABLE(HR).EAST,
\$THREE),1,3);
SUM = SUM + CNVRT
OAG.TABLE(HR).EAST;
CURSOR = CURSOR + 1;

IF CNVRT_OAG.TABLE(HR).SOUTH GT
99.0

THEN OAG_DATA.MSG = DMNDERR;

ELSE

OAG_DATA.TABLE(HR).SOUTH
= SUBSTR(F(CNVRT_OAG.
TABLE(HR).SOUTH,\$THREE),1,
3);
SUM = SUM + CNVRT
OAG.TABLE(HR).SOUTH;
CURSOR = CURSOR + 1;

IF CNVRT_OAG.TABLE(HR).
WEST GT 99.0

THEN OAG_DATA.MSG
= DMNDERR;

2-263

```
ENDREPEAT;  
IF OAG_DATA.MSG EQ 'DATA ENTERED'  
    THEN    CURSOR = 2;  
END GVALID;
```

ELSE

```
OAG DATA.TABLE  
(HR).WEST =  
SUBSTR(F(CNVRT  
OAG.TABLE  
(HR).WEST,  
$THREE),1,3);  
CURSOR =  
CURSOR - 8;  
  
IF  
FLOOR(CNVRT  
OAG.TABLE(HR).TT  
LDEP) NE SUM  
    THEN OAG  
    DATA.MSG  
    = TILERR;  
    ELSE  
    CURSOR=  
    CURSOR+8;  
    OAG  
    DAT/.TABLE(  
    HR).TTLDEP  
    =  
    SUBSTR(F(CN  
    VRT  
    OAG.TABLE(H  
    R).TTLDEP,$  
    THREE),1,3)  
    ;
```


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 CNVAT 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 BRK [pointer for braking condition data field]

PTR RVR [pointer for RVR 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 CEIL [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 DMS manual, initialized to string of (32) '1's]

ENDSTRUCTURE;

```
ROUTINE ARPT
  IN (CNVTPEM);
  INOUT (APTSTAT, MIDFLAG, CNVTAPT, RSTATUS, I);
  {This routine invokes airport status screen for both current and forecast environment}
  ARPT_DATA = APTSTAT;
  MIDDATA = MIDFLAG;
  CNVRT_APT = CNVTAPT;
  REPEAT UNTIL (RSTATUS NE PF12);
    ARPT_DATA(I) = APTSTAT(I);
    CNVRT_APT(I) = CNVTAPT(I);
    MIDDATA(I) = MIDFLAG(I);
    REPEAT UNTIL (RSTATUS NE PF3);
      I = SWITCH(I);
    CALL ASCREEN;
    IN (CNVTPEM);
    INOUT (ARPT_DATA(I), MIDDATA(I), CNVRT_APT(I), RSTATUS);
    {this routine controls airport status screen}
  ENDREPEAT;
ENDREPEAT;
LOOP; [J = 1 To 2]
  IF SUBSTR(ARPT_DATA(J).MSG, 1, 12) EQ 'DATA ENTERED'
    THEN
      APTSTAT(J) = ARPT_DATA(J);
      MIDFLAG(J) = MIDDATA(J);
      CNVTAPT(J) = CNVRT_APT(J);
  ENDLOOP;
END ARPT;
```

ROUTINE ASCREENIN (CNVTPRM);INOUT (ARPT_DATA(I), MIDDATA(I), CNVRT_APT(I), RSTATUS);
[This routine controls airport status screen]CHR PNAME [character variable of length 8 containing the name of DMS 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 DMS]STRUCTURE AUX_DATA LIKE ARPT_DATA(I)ENDSTRUCTURE;STRUCTURE AUX_CNVRT LIKE CNVRT_APT(I)ENDSTRUCTURE;CHAR AUX_MID;

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) = FLDHIGH; [set first data field to high intensity]

DM(163) = FLDHIGH; [set last data field to high intensity]

AUX_DATA = ARPT_DATA(I);

AUX_MID = MIDFLAG(I);

AUX_CNVT = CNVRT_APT(I);

PERFORM SET_UP_SCREEN_POINTERS_(ARPT);

REPEAT UNTIL (RSTATUS NE ENTER);

PERFORM DISPLAY_PANEL;

IF RSTATUS EQ PA1

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).MSG NE 'DATA ENTERED'

THEN

DM(CURSOR) = FLDHIGH;

ELSE

CALL AVALID;

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

DN(CURSOR) = FLDHIGH;

ELSE

CALL AUPDATE;

IN (CNVTPRM);

INOUT (ARPT_DATA(I), MIDDATA(I);

[This routine performs local updates on screen]

ARPT_DATA(I).MSG = 'DATA ENTERED AT 'CONCATENATE GMT;

AUX_DATA = ARPT_DATA(I);

AUX_MID = MIDDATA(I);

CNVRT_AUX = CNVRT_APT(I);

ENDREPEAT;

END ASCREEN;

PROCESS SET_UP_SCREEN_POINTERS (ARPT)

[This process sets up screen pointers for DMS 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.DEP);
AIRPORT_LOADLIST.RUNWAY(J).TOWER.DEP = ADDR(ARPT_DATA(I).RUNWAY(J).TOWER.DEP);
AIRPORT_LOADLIST.RUNWAY(J).SURF = ADDR(ARPT_DATA(I).RUNWAY(J).SURF);
AIRPORT_LOADLIST.RUNWAY(J).BRAK = ADDR(ARPT_DATA(I).RUNWAY(J).BRAK);
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).CRSS = ADDR(ARPT_DATA(I).RUNWAY(J).CRSS);
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.DEP = ADDR(ARPT_DATA(I).RUNWAY(J).CLOSED.DEP);

ENDLOOP;

AIRPORT_LOADLIST.MSG = ADDR(ARPT_DATA(I).MSG);

END SET_UP_SCREEN_POINTERS (ARPT);

ROUTINE ACHECKINOUT (ARPT_DATA(I),MIDDATA(I),CNVRT_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 VIS_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_FIELD_ERROR_CHECK;
EXITIF [error detected]
CURSOR = 6;

IF X(MIDDATA(I)) NE 0
  THEN ARPT_DATA(I).MSG = ERR5;
  ELSE
    REPEAT WHILE (ARPT_DATA(I).MSG EQ 'DATA ENTERED'); [J = 1 to 12]
    CURSOR = CURSOR + 1;

    IF X(ARPT_DATA(I).RUNWAY(J).TOWER.ARR) NE 0
      THEN ARPT_DATA(I).MSG = ERR5;
      ELSE
        CURSOR = CURSOR + 1;

        IF X(ARPT_DATA(I).RUNWAY(J).TOWER.DEP) NE 0
          THEN ARPT_DATA(I).MSG = ERR5;
          ELSE
            CURSOR = CURSOR + 1;

            IF X(ARPT_DATA(I).RUNWAY(J).SURF) NE 0
              THEN ARPT_DATA(I).MSG = ERR5;
              ELSE
                CURSOR = CURSOR + 1;

                IF X(ARPT_DATA(I).RUNWAY(J).BRK) NE 0
                  THEN ARPT_DATA(I).MSG = ERR5;
                  ELSE CURSOR = CURSOR + 9;

            ENDREPEAT;

        IF ARPT_DATA(I).MSG EQ 'DATA ENTERED'
          THEN CURSOR = 2;

    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 (CNVRT_DATA(I).WX.CEIL);
IF CNVRT_DATA(I).WX.CEIL LT 0.0
THEN ARPT_DATA(I).MSG = ERR2;
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 (CNVRT_DATA(I).WX.VIS)
IF CNVRT_DATA(I).WX.VIS LT 0.0
THEN ARPT_DATA(I).MSG = ERR2;
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).WIND.DIR) EDIT (CNVRT_DATA(I).WIND.DIR);
IF CNVRT_DATA(I).WIND.DIR LT 0.0
THEN ARPT_DATA(I).MSG = ERR2;
ELSEIF VERIFY('.', ARPT_DATA(I).WIND.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 (ARPT_DATA(I).WIND.VEL) EDIT (CNVRT_DATA(I).WIND.VEL);
  IF CNVRT_DATA(I).WIND.VEL LT 0.0
    THEN ARPT_DATA(I).MSG = ERR2;
  ELSEIF VERIFY ('.', ARPT_DATA(I).WIND.VEL) EQ 0
    THEN ARPT_DATA(I).MSG = ERR3;
END WIND_VEL_DATA_FIELD_ERROR_CHECK;

```

ROUTINE AVALIDINOUT (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 100.0THEN 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 10.0THEN 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 360.0THEN 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),1,2);

```

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) ' ') AND (ARPT_DATA(I).RUNWAY(J).BRK NE (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 AVALID;

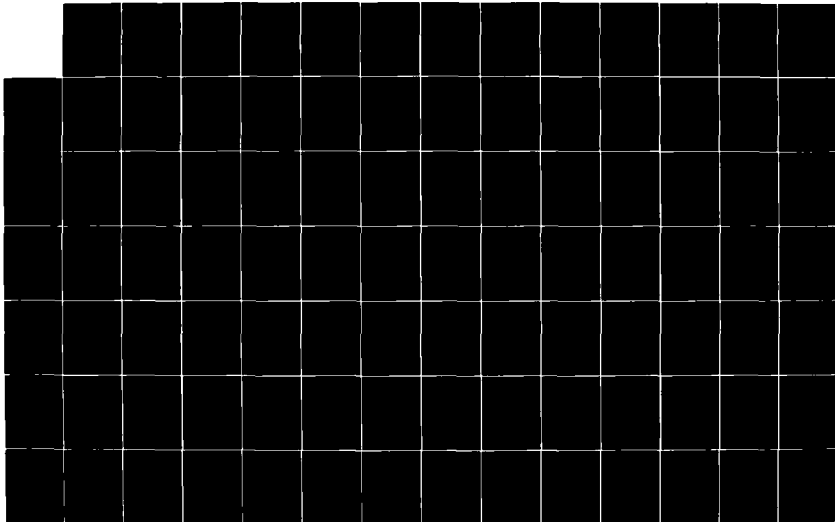
```

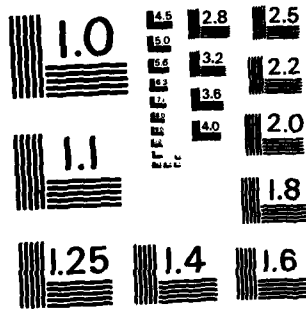
AU-A127 828

SOFTWARE DESCRIPTION FOR THE O'HARE RUNWAY
CONFIGURATION MANAGEMENT SYSTE... (U) MITRE CORP MCLEAN
VA METREK DIV 5 KAYOUSSI OCT 82 MTR-82W125-VOL-2
FAA-EM-82-28-VOL-2 DTFA01-81-C-10003 F/G 17/7

UNCLASSIFIED

4/5
NL





ROUTINE AUPDATE

IN (CNVTPEM);

INOUT (ARPT_DATA(I), MIDDATA(I), CNVRT_APT(I));
[This routine performs local updates on screen]

CALL WIND;

INOUT (ARPT_DATA(I), CNVRT_APT(I));
[This routine computes crosswind and tailwind components of prevailing wind and sets up
corresponding screen data fields]

CALL CLOSING;

IN (CNVTPEM);

INOUT (ARPT_DATA(I), CNVRT_APT(I));
[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.

2-280

LOCAL VARIABLES

STRUCTURE RWY_DATA(2) LIKE RWYEQP

[This structure is similar to RWYEQP 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)]

STRUCTURE RWY_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 MM [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 RVR [pointer for RVR 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 DMS manual, initialized to string of (32) '1'B]

ENDSTRUCTURE;

ROUTINE RWYINOUT (RWYEQP, RSTATUS, I);

[This routine invokes runway equipment status screen for both current and forecast environment]

RWY_DATA = RWYEQP;

REPEAT UNTIL (RSTATUS NE PF12);

RWY_DATA(I) = RWYEQP(I);

I = SWITCH(I);

[switch between two screens]

REPEAT UNTIL (RSTATUS NE PF4);

I = SWITCH(I);

CALL RSCREEN;INOUT (RWY_DATA(I), RSTATUS);

[This routine controls runway equipment status screen]

ENDREPEAT;ENDREPEAT;LOOP; [J = 1 to 2]IF SUBSTR(RWY_DATA(J).MSG, 1, 12) EQ 'DATA ENTERED'THEN RWYEQP(J) = RWY_DATA(J);ENDLOOP;END RWY;

ROUTINE RSCREENINOUT (RWY_DATA(I), RSTATUS);

[This routine controls runway equipment status screen]

CHAR PNAME [character variable of length 8 containing name of DMS 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(158) [8 bit variable of data mask used in DMS]STRUCTURE AUX_DATA LIKE RWY_DATA(I)ENDSTRUCTURE;

CURSOR = 3; [set cursor to position 3; first data field used by user]

DM = FLDDEF; [set data fields to default intensity (normal)]

DM(1) = FLDHIGH; [set first data field to high intensity]

DM(158) = FLDHIGH; [set last data field to high intensity]

AUX_DATA = RWY_DATA(I);

PERFORM SET_UP_SCREEN_POINTERS (RWY);REPEAT UNTIL (RSTATUS NE ENTER);PERFORM DISPLAY_PANEL;IF RSTATUS EQ PA1THEN stop;IF RSTATUS NE ENTERTHEN RWY_DATA(I) = AUX_DATA;ELSE

```
LOOP; [J = 2 To 157]
    DM(J) = FLDDEF;
ENDLOOP;
CALL RCHECK;
INOUT (RWY_DATA(I), 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 RWY_DATA(I).MSG NE 'DATA ENTERED'
    THEN DM(CURSOR) = FLDHIGH;
    ELSE
        CALL RUPDATE;
        INOUT (RWY_DATA(I);
            [This routine performs local updates on screen]
        RWY_DATA(I).MSG = 'DATA ENTERED AT ' CONCATENATE GMT;
        AUX_DATA = RWY_DATA(I);
    ENDREPEAT;
END RSCREEN;
```

```
PROCESS SET UP SCREEN POINTERS (RWY)
[This process sets up screen pointers for DMS use]

RWY_LOADLIST.TIME = ADDR(RWY_DATA(I).TIME);

LOOP;      [J = 1 To 12]
RWY_LOADLIST.RUNWAY(J).CATII = ADDR(RWY_DATA(I).RUNWAY(J).CATII);

RWY_LOADLIST.RUNWAY(J).LOC = ADDR(RWY_DATA(I).RUNWAY(J).LOC);
RWY_LOADLIST.RUNWAY(J).GS = ADDR(RWY_DATA(I).RUNWAY(J).GS);
RWY_LOADLIST.RUNWAY(J).OM = ADDR(RWY_DATA(I).RUNWAY(J).OM);
RWY_LOADLIST.RUNWAY(J).MM = ADDR(RWY_DATA(I).RUNWAY(J).MM);
RWY_LOADLIST.RUNWAY(J).IM = ADDR(RWY_DATA(I).RUNWAY(J).IM);
RWY_LOADLIST.RUNWAY(J).RAIL = ADDR(RWY_DATA(I).RUNWAY(J).RAIL);
RWY_LOADLIST.RUNWAY(J).ALS = ADDR(RWY_DATA(I).RUNWAY(J).ALS);
RWY_LOADLIST.RUNWAY(J).RVR = ADDR(RWY_DATA(I).RUNWAY(J).RVR);
RWY_LOADLIST.RUNWAY(J).HIREL = ADDR(RWY_DATA(I).RUNWAY(J).HIREL);
RWY_LOADLIST.RUNWAY(J).CL = ADDR(RWY_DATA(I).RUNWAY(J).CL);
RWY_LOADLIST.RUNWAY(J).TDZ = ADDR(RWY_DATA(I).RUNWAY(J).TDZ);
RWY_LOADLIST.RUNWAY(J).NDB_VOR = ADDR(RWY_DATA(I).RUNWAY(J).NDB_VOR);

ENDLOOP;

RWY_LOADLIST.MSG = ADDR(RWY_DATA(I).MSG);

END SET_UP_SCREEN_POINTERS (RWY);
```

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).MM) 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).IM) 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).RVR) 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).TDZ
) NE 0

THEN RWY
DATA(I).MSG =
ERR5;

2-287

```

                                ELSE
                                CURSOR =
                                CURSOR + 1;
IF (J NE 1) AND (J NE
4) AND (J NE 8) AND
(J NE 10) AND X(RWY
DATA(I).
RUNWAY(J).NDB_VOR)
NE 0

                                THEN RWY_DATA(I).MSG = ERR5

ENDREPEAT;
IF RWY_DATA(I).MSG = 'DATA ENTERED'
THEN CURSOR = 3; [if no errors detected return cursor to top]
END RCHECK;
```

ROUTINE RUPDATEINOUT (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).OM NE (2) ' 'THEN RWY_DATA(I).RUNWAY(J).CATII = ' ';IF RWY_DATA(I).RUNWAY(J).MM NE (2) ' 'THEN RWY_DATA(I).RUNWAY(J).CATII = 'X ';IF RWY_DATA(I).RUNWAY(J).IM NE (2) ' 'THEN RWY_DATA(I).RUNWAY(J).CAT II = 'X ';IF RWY_DATA(I).RUNWAY(J).RVR 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).HIRL NE (2) ' 'THEN RWY_DATA(I).RUNWAY(J).CATII = 'X ';

```
IF RWY_DATA(I).RUNWAY(J).CL NE (2) ' '  
  THEN RWY_DATA(I).RUNWAY(J).CATII = 'X '  
IF RWY_DATA(I).RUNWAY(J).TDZ NE (2) ' '  
  THEN RWY_DATA(I).RUNWAY(J).CATII = 'X '  
ENDLOOP;  
END RUPDATE;
```

2.11 Demand Profile Screen

The Demand Profile Screen is described in pages 2-291 to 2-309.

[***LOCAL VARIABLES***]

STRUCTURE DMND_DATA(2) LIKE DEMAND

[This structure is similar to DEMAND used as a working area within screen routine]

ENDSTRUCTURE;

STRUCTURE CNVRT DEM(2) LIKE CNVDEM

[This structure is similar to CNVDEM 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 DMND_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 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 FARMH [pointer for FARMH arrival demand data field]

GROUP DEP

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 MSG [pointer for screen message data field]

BITS FENCE [32 bit variable as prescribed by DNS manual, initialized to string of (32) '1's]

ENDSTRUCTURE;

```

ROUTINE DMND
  IN (CNVTOAG);
  INOUT (DEMAND, CNVTDEM, RSTATUS, 1);
  [This routine invokes demand profile screen for both current and forecast environments]

  DMND_DATA = DEMAND;
  CNVRT_DEM = CNVTDEM;

  REPEAT UNTIL (RSTATUS NE PF12);
  DMND_DATA(I) = DEMAND(I);
  CNVRT_DEM(I) = CNVTDEM(I);
  I = SWITCH(I);

  REPEAT UNTIL (RSTATUS NE PF5);
  I = SWITCH(I); [switch between two screens]

  CALL DSCREEN;
  IN (CNVTOAG);
  INOUT (DMND_DATA(I), CNVRT_DEM(I), RSTATUS, 1);
  [This routine controls demand profile status screen]

  ENDREPEAT;
  ENDREPEAT;
  LOOP; [J = 1 to 2]
  IF SUBSTR(DMND_DATA(J).MSG,1,12) EQ 'DATA ENTERED'
  THEN
    DEMAND(J) = DMND_DATA(J);
    CNVTDEM(J) = CNVRT_DEM(J);

  ENDLOOP;
END DMND;

```


ROUTINE DSCREENIN (CNVT0AG);INOUT (DMND_DATA(I), CNVRT_DEN(I), RSTATUS, I);
[This routine controls demand profile screen]CHR 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]BITS DM(15) [8 bit variable of data masks used in DMS]FLT OFFSET(2)STRUCTURE AUX_DATA LIKE DMND_DATA(I)ENDSTRUCTURE;STRUCTURE AUX_CNVT LIKE CNVRT_DEN(I)ENDSTRUCTURE;

CURSOR = 4; [set cursor to position 4; first data field used by user]

DM = FLDDEF; [set data fields to default intensity (normal)]

DM(1) = FLDHIGH; [set first data field to high intensity]

DM(15) = [set last data field to high intensity]

RETRIEVE = (2) ' ';

AUX_DATA = DMND_DATA(I);

AUX_CNVT = CNVRT_DEN(I);

Get STRING(CMT) EDIT (CFROM); [get current time]

CFROM = MOD(CFROM + OFFSET(I), 2400.0);

CTO = MOD (CFROM + 100.0, 240.0);

FROM = TRANSLATE (SUBSTR(F(CFROM,\$FROM),1,4),'0',' ');

```

PERFORM SET_UP_SCREEN_POINTERS(DMND);
REPEAT UNTIL (RSTATUS NE ENTER);
    PERFORM DISPLAY_PANEL;
    IF RSTATUS EQ PA1
        THEN stop;
    IF RSTATUS NE ENTER
        THEN DMND_DATA(I) = AUX_DATA;
        CNVRT_DEM(I) = AUX_CNVT;
    ELSE
        LOOP; {J = 4 To 12}
        DM(J) = FLDDEF;
    ENDLOOP;
    CALL DCHECK;
    IN (CNVTOAG);
    INOUT (DMND_DATA(I), RETRIEVE, CNVRT_DEM(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 DMND_DATA(I).MSG NE 'DATA ENTERED'
        THEN
            DM(CURSOR) = FLDHIGH;
            ELSE
                CALL DVALID;
                IN (CNVTOAG);

```

ENDREPEAT;
END DSCREEN;

INOUT (DMSD_DATA(I), RETRIEVE, CNVRT_DEN(I),
 CFROM, CTO, 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 DMSD_DATA(I).MSG NE 'DATA ENTERED'

THEN DM(CURSOR) = FLDHIGH;

ELSE

DMSD_DATA(I).MSG = 'DATA ENTERED AT CONCATENATE
 GHT';

AUX_DATA = DMSD_DATA(I);
 AUX_DATA = CNVRT_DEN(I);

```
PROCESS SET_UP_SCREEN_POINTERS (DMND);  
    [This process sets up screen pointers for DMS 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.KUBBS = ADDR(DMND_DATA(I).ARR.KUBBS);  
    DMND_LOADLIST.ARR.CGT = ADDR(DMND_DATA(I).ARR.CGT);  
    DMND_LOADLIST.ARR.VAINS = ADDR(DMND_DATA(I).ARR.VAINS);  
    DMND_LOADLIST.ARR.FARMM = ADDR(DMND_DATA(I).ARR.FARMM);  
    DMND_LOADLIST.DEP.TOTAL = ADDR(DMND_DATA(I).DEP.TOTAL);  
    DMND_LOADLIST.DEP.NORTH = ADDR(DMND_DATA(I).DEP.NORTH);  
    DMND_LOADLIST.DEP.EAST = ADDR(DMND_DATA(I).DEP.EAST);  
    DMND_LOADLIST.DEP.SOUTH = ADDR(DMND_DATA(I).DEP.SOUTH);  
    DMND_LOADLIST.DEP.WEST = ADDR(DMND_DATA(I).DEP.WEST);  
    DMND_LOADLIST.MSG = ADDR(DMND_DATA(I).MSG);  
  
END SET_UP_SCREEN_POINTERS (DMND);
```

ROUTINE DCHECKIN (CNVTOAG);INOUT (DMND_DATA(I), RETRIEVE, CNVRT_DEM(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]

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]

DMND_DATA(I).MSG = ERR1;

RETURN;

DMND_DATA(I).MSG = 'DATA ENTERED';

CURSOR = 4;

IF X (RETRIEVE) NE 0THEN DMND_DATA(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]
DWND DATA(I).ARR.PLANT = (4) ' ';
CNVET_DEN(I).ARR.PLANT = 0.;
CURSOR = 7;
PERFORM ARR_OGT_DATA_FIELD_ERROR_CHECK;
EXITIF [error detected]
CURSOR = 8;
PERFORM ARR_VAINS_DATA_FIELD_ERROR_CHECK;
EXITIF [error detected]
CURSOR = 9;
PERFORM ARR_FARMM_DATA_FIELD_ERROR_CHECK;
EXITIF [error detected]
DWND DATA(I).ARR.MKE_A = (4) ' ';
CNVET_DEN(I).ARR.MKE_A = 0.;
CURSOR = 10;
PERFORM DEP_TOTAL_DATA_FIELD_ERROR_CHECK;
EXITIF [error detected]
CURSOR = 11;
PERFORM DEP_NORTH_DATA_FIELD_ERROR_CHECK;
EXITIF [error detected]
CURSOR = 12;
```

```
PERFORM DEP_EAST_DATA_FIELD_ERROR_CHECK;  
EXITIF [error detected]  
CURSOR = 13;  
PERFORM DEP_SOUTH_DATA_FIELD_ERROR_CHECK;  
EXITIF [error detected]  
CURSOR = 14;  
PERFORM DEP_WEST_DATA_FIELD_ERROR_CHECK;  
EXITIF [error detected]  
DMND_DATA(1).DEP.MKE_D = (4) ' ';  
CNVRT_DEM(1).DEP.MKE_D = 0.;  
  
IF DMND_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(I).ARR.TOTAL) EDIT (CONVRT_DEM(I).ARR.TOTAL);
IF CONVRT_DEM(I).ARR.TOTAL LT 0.
THEN DMND_DATA(I).MSG = ERR2;
ELSEIF VERIFY ('.', DMND_DATA(I).ARR.TOTAL) EQ 0
THEN DMND_DATA(I).MSG = ERR3;
END ARR_TOTAL_DATA_FIELD_ERROR_CHECK;

```

```

PROCESS ARR_KUBBS_DATA_FIELD_ERROR_CHECK
[This process checks for errors on KUBBS data field]
Get STRING (DMND_DATA(I).ARR.KUBBS) EDIT (CONVRT_DEM(I).ARR.KUBBS);
IF CONVRT_DEM(I).ARR.KUBBS LT 0.
THEN DMND_DATA(I).MSG = ERR2;
ELSEIF VERIFY ('.', DMND_DATA(I).ARR.KUBBS) EQ 0
THEN DMND_DATA(I).MSG = ERR3;
END ARR_KUBBS_DATA_FIELD_ERROR_CHECK;

```



```
PROCESS ARR_CGT_DATA_FIELD_ERROR_CHECK  
  [This process checks for errors on CGT data field]  
  Get STRING (DMND_DATA(I).ARR.CGT) EDIT (CNVRT_DEM(I).ARR.CGT);  
  IF CONVRT_DEM(I).ARR.CGT LT 0.  
    THEN DMND_DATA(I).MSG = ERR2;  
    ELSEIF VERIFY ('.', DMND_DATA(I).ARR.CGT) EQ 0  
      THEN DMND_DATA(I).MSG = ERR3;  
END ARR_CGT_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 (CNVRT_DEM(I).ARR.VAINS);  
  IF CONVRT_DEM(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 (DMND_DATA(I).ARR.FARM) EDIT (CONVRT_DEM(I).ARR.FARM);  
  
  IF CONVRT_DEM(I).ARR.FARM LT 0.  
    THEN DMND_DATA(I).MSG = ERR2;  
    ELSEIF VERIFY ('.', DMND_DATA(I).ARR.FARM) EQ 0  
      THEN DMND_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 (DMND_DATA(I).DEP.TOTAL) EDIT (CONVRT_DEM(I).DEP.TOTAL);  
  
  IF CONVRT_DEM(I).DEP.TOTAL LT 0.  
    THEN DMND_DATA(I).MSG = ERR2;  
    ELSEIF VERIFY ('.', DMND_DATA(I).DEP.TOTAL) EQ 0  
      THEN DMND_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 (DMND_DATA(I).DEP.NORTH) EDIT (CNVRT_DEM(I).DEP.NORTH);
  IF CNVRT_DEM(I).DEP.NORTH LT 0.
    THEN DMND_DATA(I).MSG = ERR2;
    ELSEIF VERIFY ('.', DMND_DATA(I).DEP.NORTH) EQ 0
      THEN DMND_DATA(I).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 (DMND_DATA(I).DEP.EAST) EDIT (CNVRT_DEM(I).DEP.EAST);
  IF CNVRT_DEM(I).DEP.EAST LT 0.
    THEN DMND_DATA(I).MSG = ERR2;
    ELSEIF VERIFY ('.', DMND_DATA(I).DEP.EAST) EQ 0
      THEN DMND_DATA(I).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 (DMND_DATA(I).DEP.SOUTH) EDIT (CNVRT_DEM(I).DEP.SOUTH);

IF CNVRT_DEM(I).DEP.SOUTH LT 0.
    THEN DMND_DATA(I).MSG = ERR2;
    ELSEIF VERIFY ('.', DMND_DATA(I).DEP.SOUTH) EQ 0
        THEN DMND_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 (DMND_DATA(I).DEP.WEST) EDIT (CNVRT_DEM(I).DEP.WEST);

IF CNVRT_DEM(I).DEP.WEST LT 0.
    THEN DMND_DATA(I).MSG = ERR2;
    ELSEIF VERIFY ('.', DMND_DATA(I).DEP.WEST) EQ 0
        THEN DMND_DATA(I).MSG = ERR3;
END DEP_WEST_DATA_FIELD_ERROR_CHECK;
```

ROUTINE DVALIDIN (CNVTOAG);INOUT (DMND_DATA(I).RETRIEVE,CNVRT_DEM(I),CFROM,CTO,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;

DMNDERR = 'NUMBER OF AIRCRAFT MUST NOT EXCEED 99';

TTLERR = 'TOTAL DOES NOT EQUAL SUM OF INDIVIDUAL ENTRIES';

IF RETRIEVE NE (2) ' ';THEN PERFORM RETRIEVE_DEMAND_DATA_FROM_DEMAND_LOG;

CURSOR = 6;

IF CNVRT_DEM(I).ARR.KUBBS GT 99.0THEN DMND_DATA(I).MSG = DMNDERR;ELSE

DMND_DATA(I).ARR.KUBBS = SUBSTR(F(CNVRT_DEM(I).ARR.KUBBS,\$THREE),1,3);

CURSOR = 7;

IF CNVRT_DEM(I).ARR.CGT GT 99.0THEN DMND_DATA(I).MSG = DMNDERR;ELSE

DMND_DATA(I).ARR.CGT = SUBSTR(F(CNVRT_DEM(I).ARR.CGT,\$THREE),1,3)

CURSOR = 8;

IF CNVRT_DEM(I).ARR.VAINS GT 99.0THEN DMND_DATA(I).MSG = DMNDERR;ELSE

DMND_DATA(I).ARR.VAINS = SUBSTR(F(CNVRT_DEM(I).ARR.VAINS,\$THREE),1,3);

```
CURSOR = 9;
IF CNVRT_DEM(1).ARR.FARM GT 99.0
  THEN DMND_DATA(1).MSG = DMNDERR;
  ELSE
    DMND_DATA(1).ARR.FARM = SUBSTR(F(CNVRT_DEM(1).ARR.
    FARM,$THREE),1,3)
    CURSOR = 11;
  IF CNVRT_DEM(1).DEP.NORTH GT 99.0
    THEN DMND_DATA(1).MSG = DMNDERR;
    ELSE
      DMND_DATA(1).DEP.NORTH = SUBSTR(F(CNVRT_DEM(1).DEP.
      NORTH,$THREE),1,3);
      CURSOR = 12;
    IF CNVRT_DEM(1).DEP.EAST GT 99.0
      THEN DMND_DATA(1).MSG = DMNDERR;
      ELSE DMND_DATA(1).ARR.FARM = SUBSTR(F(CNVRT
      DEM(1).ARR.FARM,$THREE),1,3);
      CURSOR = 13;
    IF CNVRT_DEM(1).DEP.SOUTH GT 99.0
      THEN DMND_DATA(1).MSG = DMNDERR;
      ELSE
        DMND_DATA(1).DEP.SOUTH = SUBSTR(F(CNVRT
        DEM(1).DEP.SOUTH,$THREE),1,3);
        CURSOR = 14;
        IF CNVRT_DEM(1).DEP.WEST GT 99.0
```

```

      THEN      DMND_DATA(I).MSG = DMNDERR;

    ELSE
      DMND_DATA(I).DEP.WEST = SUBSTR(P(CNVRT
      DEM(I).DEP.WEST,$THREE),1,3);
      CURSOR = 5;

      IF FLOOR(CNVRT DEM(I).ARR.TOTAL) NE
      FLOOR(CNVRT DEM(I).ARR.KUBBS +
      CNVRT DEM(I).ARR.CGT + CNVRT
      DEM(I).ARR.VAINS + CNVRT
      DEM(I).ARR.FARM)

      THEN      DMND_DATA(I).MSG =
      TTLEERR;

      ELSE
      DMND_DATA(I).ARR.TOTAL =
      SUBSTR(P(CNVRT
      DEM(I).ARR.TOTAL,$THREE),1,3);

      CURSOR = 10;
      IF FLOOR(CNVRT
      DEM(I).DEP.TOTAL) NE
      FLOOR(CNVRT
      DEM(I).DEP.NORTH + CNVRT
      DEM(I).DEP.EAST + CNVRT
      DEM(I).DEP.SOUTH + CNVRT
      DEM(I).DEP.WEST)

      THEN DMND_DATA(I).MSG =
      TTLEERR;

      ELSE DMND
      DATA(I).DEP.TOTAL =
      SUBSTR(P(CNVRT
      DEM(I).DEP.TOTAL,$THR
      EE),1,3);

    IF DMND_DATA(I).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.KUBBS = FLOAT(FLOOR((1.0 - ALPHA)*CNVTOAG.TABLE (INDEX).KUBBS + ALPHA * CNVTOAG.TABLE
  (NEXT).KUBBS + 0.5));

  CNVRT.DEM(I).ARR.CGT = FLOAT(FLOOR((1.0 - ALPHA)*CNVTOAG.TABLE(INDEX).CGT + ALPHA *
  CNVTOAG.TABLE(NEXT).CGT + .5));

  CNVRT.DEM(I).ARR.VAINS = FLOAT(FLOOR((1.0 - ALPHA)*CNVTOAG.TABLE(INDEX).VAINS + ALPHA *
  CNVTOAG.TABLE(NEXT).VAINS + .5));

  CNVRT.DEM(I).ARR.FARMH = FLOAT(FLOOR((1.0 - ALPHA)*CNVTOAG.TABLE(INDEX).FARMH + ALPHA *
  CNVTOAG.TABLE(NEXT).FARMH + .5));

  CNVRT.DEM(I).DEP.NORTH = FLOAT(FLOOR((1.0 - ALPHA)*CNVTOAG.TABLE(INDEX).NORTH + ALPHA *
  CNVTOAG.TABLE(NEXT).NORTH + .5));

  CNVRT.DEM(I).DEP.EAST = FLOAT(FLOOR((1.0 - ALPHA)*CNVTOAG.TABLE(INDEX).EAST + ALPHA *
  CNVTOAG.TABLE(NEXT).EAST + .5));

  CNVRT.DEM(I).DEP.SOUTH = FLOAT(FLOOR((1.0 - ALPHA)*CNVTOAG.TABLE(INDEX).SOUTH + ALPHA *
  CNVTOAG.TABLE(NEXT).SOUTH + .5));

  CNVRT.DEM(I).DEP.WEST = FLOAT(FLOOR((1.0 - ALPHA)*CNVTOAG.TABLE(INDEX).WEST + ALPHA *
  CNVTOAG.TABLE(NEXT).WEST + .5));

  CNVRT_DEM(I).ARR.TOTAL = CNVRT_DEM(I).ARR.KUBBS + CNVRT_DEM(I).ARR.CGT + CNVRT_DEM(I).ARR.VAINS + CNVRT
  DEM(I).ARR.FARMH;

  CNVRT_DEM(I).DEP.TOTAL = CNVRT_DEM(I).DEP.NORTH + CNVRT_DEM(I).DEP.
  EAST + CNVRT_DEM(I).DEP.SOUTH + CNVRT_DEM(I).DEP.WEST;

  RETRIEVE = (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]

FLT 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 HMS manual, initialized to string of (32) '1's]

ENDSTRUCTURE;

ROUTINE ORDERIN (PRCARR, INFORM, CNFGREQ, RWYEQP, MIDFLAG);INOUT (CONFLST, CONFIND, RSTATUS, I);

[This routine invokes ordered list of configurations screen for both current and forecast environments]

CALL OSETUP;IN (PRCARR, INFORM, CNFGREQ, RWYEQP, MIDFLAG)INOUT (CONFLST, LIST, COUNT);

[This routine sets up information on ordered list of configurations screen]

CDATA = CONFIND;

MSG_DATA(1) = CONFLIST(1).MSG;

MSG_DATA(2) = CONFLIST(2).MSG;

REPEAT UNTIL (RSTATUS NE PF12);

CDATA(I) = CONFIND(I);

CONFLIST(I).MSG = MSG_DATA(I);

I = SWITCH(I); [switch between two screens]

REPEAT UNTIL (RSTATUS NE PF6);CALL OSCREEN;

```

      IN (CONFLIST(I), LIST(I), COUNT(I));
      INOUT (CDATA(I), RSTATUS);
           [This routine controls ordered list of configurations screen]
      ENDREPEAT;
      ENDREPEAT;
      LOOP; [J = 1 To 2]
          IF SUBSTR(CONFLIST(J).MSG,1,12) EQ 'DATA ENTERED'
              THEN CONFIND(J) = CDATA(J);
          ENDLOOP;
      END ORDER;

```

2-314

ROUTINE OSETUPIN (PRCARR, INFORM, CNFGRQ, RWYEQP, MIDFLAG, COUNT);INOUT (CONPLST, 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).TOTARR + PRCARR(K).TOTDEP = 0.0THEN ATOTPRC = .5; [default value of arrival percentage is .5 if no demand is specified]ELSE ATOTPRC = PRCARR(K).TOTARR / (PRCARR(K).TOTARR + PRCARR(K).TOTDEP);

CONPLST(K).TOT_ARR = SUBSTR(F(ATOTPRC * 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 999THEN

PERFORM SCREEN_PARAMETERS_SET UP;
[set up parameters on screen]

PERFORM FLAG_SETTING;
[set up appropriate flags]

ENDLOOP;

CONFLST(K).NUMBER = SUBSTR(F(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) = CNFGRQ(P).ARR_RWY(1);

CONFLST(K).CONFIG(COUNT(K)).ARR(2) = CNFGRQ(P).ARR_RWY(2);

CONFLST(K).CONFIG(COUNT(K)).ARR(3) = CNFGRQ(P).ARR_RWY(3);

CONFLST(K).CONFIG(COUNT(K)).DEP(1) = CNFGRQ(P).DEP_RWY(1);

CONFLST(K).CONFIG(COUNT(K)).DEP(2) = CNFGRQ(P).DEP_RWY(2);

CONFLST(K).CONFIG(COUNT(K)).DEP(3) = CNFGRQ(P).DEP_RWY(3);

CONFLST(K).CONFIG(COUNT(K)).DEP(4) = CNFGRQ(P).DEP_RWY(4);

CONFLST(K).CONFIG(COUNT(K)).CAPACITY = SUBSTR(P/LIST(K).CONF(N).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) ' 'THEN HIRLIND = HIRLIND CONCATENATE '1' B;ELSE HIRLIND = HIRLIND CONCATENATE '0' B;ENDLOOP;HIRLIND = HIRLIND CONCATENATE HIRLIND;

MESSAGE = (27) ' ';

IF PRCARR(K).CONF(P).BNPRCNT LT 0. [if airport is saturated]THEN

MESSAGE = 'SATURATED';

FLAG = '1'B;

IF MIDFLAG(K) NE (2) ' ' AND ((CNFGRQ(P).ID AND MININD) GT 0)THEN IF FLAG EQ '1'BTHEN MESSAGE = MESSAGE CONCATENATE, 'MIDWAY';ELSE

MESSAGE = 'MIDWAY';

FLAG = '1'B;

IF (CNFGRQ(P).ID AND HIRLIND) NE (24) '0'BTHEN IF FLAG EQ '1'BTHEN MESSAGE = MESSAGE CONCATENATE 'DAY ONLY';ELSE MESSAGE = 'DAY ONLY';END FLAG_SETTING;

2-319

ROUTINE OSORT

INOUT (LIST(K));
[This routine sorts configurations list on capacity; Shell's method is used]

H(1) = 36;
H(2) = 18;
H(3) = 9;
H(4) = 5;
H(5) = 3;
H(6) = 1;

LOOP; [M = 1 to 6]

LOOP; [J = (H(M)+1) to 73]

I = J - H(M);

IF LIST(K).CONF(J).CAPACITY GT LIST(K).CONF(I).CAPACITY

THEN

TEMP1 = LIST(K).CONF(I).CAPACITY;
TEMP2 = LIST(K).CONF(I).INDEX;
LIST(K).CONF(I).CAPACITY = LIST(K).CONF(I+H(M)).CAPACITY;
LIST(K).CONF(I).INDEX = LIST(K).CONF(I+H(M)).INDEX;
LIST(K).CONF(I+H(M)).CAPACITY = TEMP1;
LIST(K).CONF(I+H(M)).INDEX = TEMP2;
I = I - H(M);

IF I GT 0

THEN

REPEAT WHILE ((I GT 0) AND (LIST(K).CONF(I+H(M)).CAPACITY GT LIST(K).
CONF(I).CAPACITY));

TEMP1 = LIST(K).CONF(I).CAPACITY;
TEMP2 = LIST(K).CONF(I).INDEX;
LIST(K).CONF(I).CAPACITY = LIST(K).CONF(I+H(M)).CAPACITY;
LIST(K).CONF(I).INDEX = LIST(K).CONF(I+H(M)).INDEX;
LIST(K).CONF(I+H(M)).CAPACITY = TEMP1;
LIST(K).CONF(I+H(M)).INDEX = TEMP2;
I = I - H(M);

ENDREPEAT;

ENDLOOP;

ENDLOOP

END OSORT;

ROUTINE OSCREENIN (CONGLST(I), LIST(I), COUNT(I));INOUT (CDATA(I), RSTATUS);

[This routine controls ordered list of configurations screen]

CHR PNAME [character variable of length 8 containing name of DMS 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 DMS]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 CONFLST structure that are blank]CHR SELECT [length 1]CHR RANK [length 2]CHR ARR(3) [length 3]CHR DEP(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 (RSTATUS NE ENTER);PERFORM SCREEN_SCROLL;

2-322

```
PERFORM DISPLAY_PANEL;

IF RSTATUS EQ PAL
  THEN stop;
IF RSTATUS NE ENTER
  THEN;
  ELSE
    DM = FLDEF;
    DM(1) = FLDHIGH;
    DM(115) = FLDHIGH;
    CALL OCHECK;
    IN (CNVT_SCROLL, L, M, COUNT(I));
    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]
    IF CONFLST(I).MSG NE 'DATA ENTERED'
      THEN
        DM(CURSOR) = FLDHIGH;
        CNVT_SCROLL = 0;
      ELSE
        CALL OVALID;
        IN (L, M, COUNT(I));
```

2-323

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

DM(CURSOR) = FLDHIGH;

CNVT_SCROLL = 0;

CALL OVALID;

IN (LIST(I), L, M, CONFLST(I));

INOUT (CDATA(I));

{This routine locally updates configuration index parameter}

CONFLST(I).MSG = 'DATA ENTERED AT ' CONCATENATE GMT;

CONFLST(I).SCROLL = (4) ' ';

LOOP; [P = 1 to 73]

CONFLST(I).CONF(P).SELECT = ' ';

ENDLOOP;

ENDREPEAT

END OSCREEN;

PROCESS SET UP SCREEN PERMANENT POINTERS (ORDER)

[This process sets up screen pointers for permanent variables for DMS use]

```

ORDER_LOADLIST.TIME = ADDR(CONFLST(I).TIME);
ORDER_LOADLIST.TOT ARR = ADDR(CONFLST(I).TOT ARR);
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;

CNVT_SCROLL = 0;

DM = FLDDEF;

DM(115) = FLDHIGH;

[set up other parameters]

CONFLST(I).SCROLL = (4) ' ';

LOOP; [P = 1 to 73]

CONFLST(I).CONF(P).SELECT = ' ';

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);
 INDEX = L;
 NEXT = L + DELTA - 1;

ELSE

M = NEXT;
 L = INDEX;

K = 0;

IF (M LT 10) AND COUNT GT 9

THEN REPEAT WHILE (K LT 10-M);

K = K + 1;
 DH(11*K-6) = FLDDARK;

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).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 = 1 to M]

    K = K + 1;

    IF CDATA(I) EQ LIST(I).CONF(J).INDEX
        THEN
            LOOP; [P = 1 to 11]

                DW(11*K + P - 7) = FLDHIGH; [highlight operating configuration]

            ENDLOOP;

            ORDER_LOADLIST.CONF(K).RANK = ADDR(CONFLST(I).CONF(J).RANK);
            ORDER_LOADLIST.CONF(K).SELECT = ADDR(CONFLST(I).CONF(J).SELECT);
            ORDER_LOADLIST.CONF(K).ARR(1) = ADDR(CONFLST(I).CONF(J).ARR(1));
            ORDER_LOADLIST.CONF(K).ARR(2) = ADDR(CONFLST(I).CONF(J).ARR(2));
            ORDER_LOADLIST.CONF(K).ARR(3) = ADDR(CONFLST(I).CONF(J).ARR(3));
            ORDER_LOADLIST.CONF(K).DEP(1) = ADDR(CONFLST(I).CONF(J).DEP(1));
            ORDER_LOADLIST.CONF(K).DEP(2) = ADDR(CONFLST(I).CONF(J).DEP(2));
            ORDER_LOADLIST.CONF(K).DEP(3) = ADDR(CONFLST(I).CONF(J).DEP(3));
            ORDER_LOADLIST.CONF(K).DEP(4) = ADDR(CONFLST(I).CONF(J).DEP(4));
            ORDER_LOADLIST.CONF(K).CAPACITY = ADDR(CONFLST(I).CONF(J).CAPACITY);
            ORDER_LOADLIST.CONF(K).REMARKS = ADDR(CONFLST(I).CONF(J).REMARKS);

        ENDREPEAT;

```

REPEAT WHILE (K LT 10);

K = K + 1;

DM(11*K - 6) = FLD DARK;

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).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 OCHECKIN (CNVT_SCROLL, L, M, COUNT(I));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 0THEN CONFLST(I).MSG = ERR3;ELSEIF (L EQ 1) AND (COUNT GE 10)THEN K = 10 - M;ELSE K = 0;REPEAT WHILE (CONFLST(I).MSG EQ 'DATA ENTERED'); [J = 1 to M]

CURSOR = 11*K + 5;

K = K + 1;

```
      IF VERIFY(CONFLST(I).CONF(J).SELECT,X_OR_BLANK)NE 0
        THEN CONFLST(I).MSG = ERR5;
      ENDREPEAT;
    IF CONFLST(I).MSG EQ 'DATA ENTERED'
      THEN CURSOR = 4;
  END OCHECK;
```

ROUTINE OVALIDIN (L, M, 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]

ERRSEL = 'SELECT ONLY ONE CONFIGURATION';

CONFLST(I).MSG = 'DATA ENTERED';

FLAG = 0;

IF (L EQ 1) AND (COUNT GE 10)THEN K = 10 - M;ELSE K = 0;REPEAT WHILE (FLAG LT 2); [J = 1 to M]

CURSOR = 11*K + 5;

K = K + 1;

IF (CONFLST(I).CONF(J).SELECT NE ' ')THEN FLAG = FLAG + 1;ENDREPEATIF FLAG LT 2THEN CURSOR = 4;ELSE CONFLST(I).MSG = ERRSEL;END OVALID;

2-331

ROUTINE OUPDATE

IN (LIST(I), L, M, CONFLIST(I));

INOUT (CDATA(I));

[This routine updates configuration index parameter locally]

FLAG = '0'B;

REPEAT WHILE (FLAG EQ '0'B); [J = L to M]

IF CONFLST(I).CONF(J).SELECT NE ' '

THEN

CDATA(I) = LIST(I).CONF(J).INDEX;

CONFLST(I).CONF(J).SELECT = ' ';

FLAG = '1'B;

ENDREPEAT;

END OUPDATE;

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 CNVRT_QUELEN(4) [This variable is similar to CNVTQLN used as a working area within screen routine]

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

GROUP LINE(4)

PTR DEPRUN [pointer for departure runway data field]

PTR QL [pointer for queue length data field]

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

ENDSTRUCTURE;

2-334

```
ROUTINE QUEUE

  INOUT (QUELEN, CNVTQLN, RSTATUS);
    [This routine invokes current departure queue screen]
  REPEAT UNTIL (RSTATUS NE PF12);
    QUELEN_DATA = QUELEN;
    CNVRT_QUELEN = CNVTQLN;
    REPEAT UNTIL (RSTATUS NE PF7);
      CALL QSCREEN;
      INOUT (QUELEN_DATA, CNVRT_QUELEN, RSTATUS);
        [This routine controls current departure queue screen]
      ENDREPEAT;
    ENDREPEAT;
  IF SUBSTR(QUELEN_DATA.MSG,1,12) EQ 'DATA ENTERED'
    THEN
      QUELEN = QUELEN_DATA;
      CNVTQLN = CNVRT_QUELEN;
END QUEUE;
```

ROUTINE QSCREENINOUT (QUELEN_DATA, CMVRT_QUELEN, RSTATUS);

[This routine controls current departure queue screen]

CHR PNAME [character variable of length 8 containing name of DMS panel initialized to 'QLENGTH',
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 DMS]STRUCTURE AUX_DATA LIKE QUELEN_DATAENDSTRUCTURE;

CURSOR = 2;

AUX_DATA = QUELEN_DATA;

DM = FLDDDEF; [set data fields to default intensity (normal)]

DM(9) = FLDDHIGH; [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) = FLDDDEF;

DEPCOUNT = DEPCOUNT + 1;

ELSE DM(2*J) = FLDDARK; [darken screen if there are no departure runways]ENDLOOP;REPEAT UNTIL (RSTATUS NE ENTER);PERFORM DISPLAY_PANEL;

```

IF RSTATUS EQ PA1
    THEN stop;
IF RSTATUS NE ENTER
    THEN QUELEN_DATA = AUX_DATA;
    ELSE
        LOOP; [J = 1 to 2 * DEPCOUNT]
            DN(J) = FLDDEF;
        ENDLOOP;
        CALL QCHECK;
        IN (DEPCOUNT);
        INOUT (QUELEN_DATA, CNVRT_QUELEN, 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 QUELEN_DATA.MSG NE 'DATA ENTERED'
            THEN DN(CURSOR) = FLDHIGH;
            ELSE
                CALL QVALID;
                IN (DEPCOUNT);
                INOUT (QUELEN_DATA, CNVRT_QUELEN)
                [This routine right-justifies data on screen]
                QUELEN_DATA.MSG = 'DATA ENTERED AT 'CONCATENATE GMT;
                AUX_DATA = QUELEN_DATA;

        ENDREPEAT;
    END QSCREEN;

```

```

PROCESS SET_UP_SCREEN_POINTERS_(QUEUE)
  {This process sets up screen pointers for DMS 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));
  ENDLLOOP;
  QUE_LOADLIST.MSG = ADDR(QUELEN_DATA.MSG);
END SET_UP_SCREEN_POINTERS_(QUEUE);

```

ROUTINE QCHECKIN (DEPCOUNT);INOUT (QUELEN_DATA, CNVRT_QUELEN, 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 'DATE ENTERED') [J = 1 to DEPCOUNT]

CURSOR = 2*J;

Get STRING (QUELEN_DATA.QL(J)) EDIT (CNVRT_QUELEN(J));

IF VERIFY('_', QUELEN_DATA.QL(J)) EQ 0THEN QUELEN_DATA.MSG = ERR2;ELSEIF VERIFY('.', QUELEN_DATA.QL(J)) EQ 0THEN QUELEN_DATA.MSG = ERR3;ENDREPEAT;IF QUELEN_DATA.MSG = 'DATA ENTERED'THEN CURSOR = 2;END QCHECK;

ROUTINE QVALID

IN (DEPCOUNT);

INOUT (QUELEN_DATA, CNVRT_QUELEN);
[This routine right-justifies on data on screen]

\$TWO = 2;

LOOP; [J = 1 To DEPCOUNT]

QUELEN_DATA.QL(J) = SUBSTR(F(FLOAT(CNVRT_QUELEN(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.

2-341

LOCAL VARIABLES

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 CTRANHR [pointer for current configuration's transition hour capacity data field]

PTR CFINCAP [pointer for current configuration's final capacity data field]

GROUP CONFIG(10)

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]

BITS FENCE [32 bit variable as prescribed by DMS 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 (PRECARR, CNFGCRQ, CNVTDEM, DEPMAT, FIXTRAV, TRANLST, INFORM, CONFIND, CNDTN, ELGBLTY, CNVTQLN, QUELEN);

INOUT (RSTATUS);
[This routine invokes ordered list of transitions screen]

CALL TRAN;

IN (PRECARR, CNFGCRQ, CNVTDEM, DEPMAT, FIXTRAV, INFORM, CONFIND, CNDTN, ELGBLTY, CNVTQLN, QUELEN);

INOUT (TRANLST)
[This routine performs transition computations and transition screen parameter set up]

TEMP1 = TRANLST.MSG;

TEMP2 = (4) ' ';

REPEAT UNTIL (RSTATUS NE PF12);

TRANLST.MSG = TEMP1;
TRANLST.SCROLL = TEMP2;

REPEAT UNTIL (RSTATUS NE PF8);

CALL TSCREEN;

IN (ELGBLTY(2));

INOUT (TRANLST, RSTATUS);
[This routine controls ordered list of transitions screen]

ENDREPEAT;

ENDREPEAT;

END TSETUP;

2-343

ROUTINE TSCREEN

IN (ELGBLY(2));

INOUT (TRANLST, RSTATUS);

[This routine controls ordered list of transitions screen]

CHR PNAME [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 masks used in DMS]

INT CNVT_SCROLL [integer value of scroll data field]

PERFORM SET_UP_SCREEN_PERMANENT_POINTERS (TSETUP);

PERFORM SCREEN_PROGRAM_INITIALIZATION;

IF ELGBLY(2).NUM EQ 0

THEN TRANLST.MSG = 'NO ELIGIBLE CONFIGURATIONS';

IF TRANLST.MSG EQ 'CURRENT CONFIGURATION IS INELIGIBLE'

THEN

COUNT = 0;

DELTA = 0;

NEXT = 0;

REPEAT UNTIL (RSTATUS NE ENTER);

PERFORM SCREEN_SCROLL;

PERFORM DISPLAY_PANEL;

IF RSTATUS EQ PA1

THEN stop;

2-344

```
IF RSTATUS EQ ENTER
  THEN
    DM = FLDDEF;
    DM(123) = FLDHIGH;
    LOOP; [J = 4 to 12]
      DM(J) = FLDHIGH;
    ENDLOOP;
    CALL TCHECK;
    INOUT (TRANLST, CHVT_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 TRANLST.MSG NE 'DATA ENTERED'
      THEN
        DM(CURSOR) = FLDHIGH;
        CHVT_SCROLL = 0;
      ELSE
        TRANLST.SCROLL = (4) ' ';
        TRANLST.MSG = 'DATA ENTERED AT ' CONCATENATE CMT;
    ENDREPEAT
END TSCREEN;
```

```

PROCESS SET_UP_SCREEN_PERMANENT_POINTERS_(TSETUP)
  [This process sets up screen pointers of permanent variables for DMS use]

  TRANS_LOADLIST.ARR(1) = ADDR(TRANLST.ARR(1));
  TRANS_LOADLIST.ARR(2) = ADDR(TRANLST.ARR(2));
  TRANS_LOADLIST.ARR(3) = ADDR(TRANLST.ARR(3));
  TRANS_LOADLIST.DEP(1) = ADDR(TRANLST.DEP(1));
  TRANS_LOADLIST.DEP(2) = ADDR(TRANLST.DEP(2));
  TRANS_LOADLIST.DEP(3) = ADDR(TRANLST.DEP(3));
  TRANS_LOADLIST.DEP(4) = ADDR(TRANLST.DEP(4));
  TRANS_LOADLIST.PCT_ARR = ADDR(TRANLST.PCT_ARR);
  TRANS_LOADLIST.NUM_ELIG = ADDR(TRANLST.NUM_ELIG);
  TRANS_LOADLIST.SCROLL = ADDR(TRANLST.SCROLL);
  TRANS_LOADLIST.CTRANHR = ADDR(TRANLST.CTRANHR);
  TRANS_LOADLIST.CPINCAP = ADDR(TRANLST.CPINCAP);
  TRANS_LOADLIST.MSG = ADDR(TRANLST.MSG);

  LOOP: [J = 1 to 10]

    TRANS_LOADLIST.CONFIG(J).RANK = ADDR(TRANLST.CONFIG(J).RANK);
    TRANS_LOADLIST.CONFIG(J).ARR(1) = ADDR(TRANLST.CONFIG(J).ARR(1));
    TRANS_LOADLIST.CONFIG(J).ARR(2) = ADDR(TRANLST.CONFIG(J).ARR(2));
    TRANS_LOADLIST.CONFIG(J).ARR(3) = ADDR(TRANLST.CONFIG(J).ARR(3));
    TRANS_LOADLIST.CONFIG(J).DEP(1) = ADDR(TRANLST.CONFIG(J).DEP(1));
    TRANS_LOADLIST.CONFIG(J).DEP(2) = ADDR(TRANLST.CONFIG(J).DEP(2));
    TRANS_LOADLIST.CONFIG(J).DEP(3) = ADDR(TRANLST.CONFIG(J).DEP(3));
    TRANS_LOADLIST.CONFIG(J).DEP(4) = ADDR(TRANLST.CONFIG(J).DEP(4));
    TRANS_LOADLIST.CONFIG(J).HOURLY = ADDR(TRANLST.CONFIG(J).HOURLY);
    TRANS_LOADLIST.CONFIG(J).PINCAP = ADDR(TRANLST.CONFIG(J).PINCAP);
    TRANS_LOADLIST.CONFIG(J).MINUTES = ADDR(TRANLST.CONFIG(J).MINUTES);

  ENDLLOOP;

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;

CNVT_SCROLL = 0;
TRANLST.MSG = (4) ' ';
COUNT = ELGBLTY.NUM;
DELTA = MIN(COUNT, 10);
NEXT = DELTA;
INDEX = 1;

END SCREEN_PROGRAM_INITIALIZATION;

2-347

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;

IF (M LT 10) AND (COUNT GT 9)

THEN

REPEAT WHILE (K LT 10 - M)

LOOP; [J = 1 to 11]

DM(11*K + J + 12) = FLDDARK;

ENDLOOP;

K = K + 1;

2-348

```
ENDREPEAT;
REPEAT WHILE      (M NE 0)); [J = L to M]
    K = K + 1;
    TRANS_LOADLIST.CONFIG(K).RANK = ADDR(TRANLST.CONFIG(J).RANK);
    TRANS_LOADLIST.CONFIG(K).ARR(1) = ADDR(TRANLST.CONFIG(J).ARR(1));
    TRANS_LOADLIST.CONFIG(K).ARR(2) = ADDR(TRANLST.CONFIG(J).ARR(2));
    TRANS_LOADLIST.CONFIG(K).ARR(3) = ADDR(TRANLST.CONFIG(J).ARR(3));
    TRANS_LOADLIST.CONFIG(K).DEP(1) = ADDR(TRANLST.CONFIG(J).DEP(1));
    TRANS_LOADLIST.CONFIG(K).DEP(2) = ADDR(TRANLST.CONFIG(J).DEP(2));
    TRANS_LOADLIST.CONFIG(K).DEP(3) = ADDR(TRANLST.CONFIG(J).DEP(3));
    TRANS_LOADLIST.CONFIG(K).DEP(4) = ADDR(TRANLST.CONFIG(J).DEP(4));
    TRANS_LOADLIST.CONFIG(K).HOURLY = ADDR(TRANLST.CONFIG(J).HOURLY);
    TRANS_LOADLIST.CONFIG(K).FINCAP = ADDR(TRANLST.CONFIG(J).FINCAP);
    TRANS_LOADLIST.CONFIG(K).MINUTES = ADDR(TRANLST.CONFIG(J).MINUTES);
ENDLOOP;
REPEAT WHILE (K LT 10);
    LOOP; [J = 1 to 11]
        DM(11*K + J + 12) = FLDDARK;
    ENDLOOP;
    K = K + 1;
ENDREPEAT;
END SCREEN_SCROLL;
```

ROUTINE TCHECK

INOUT (TRANLST, 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 appropriate screen message is issued advising user with corrections]

ERR1 = 'NUMERIC INPUT REQUIRED';
ERR3 = 'NO DECIMAL POINTS ALLOWED';

ON CONVERSION BEGIN;

TRANLST.MSG = ERR1;

RETURN;

TRANLST.MSG = 'DATA ENTERED';

Get STRING (TRANLST.SCROLL) EDIT (CNVT_SCROLL);

IF VERIFY ('.', TRANLST.SCROLL) EQ 0

THEN TRANLST.MSG = ERR3;

END TCHECK;

ROUTINE TRAN

IN (PRCARR, CNFCRQ, CNVTDEM, DEPMAT, FIXTRAV, INFORM, CONFIND, CNDTN, ELGBLTY, CNVTQLN, QUELEN);

INOUT (TRANLST);

[This routine performs transition computations and screen parameter set up]

Y = '0'B;
DEN = 0;
\$TWO = 2;
\$THREE = 3;
\$FOUR = 4;

IF (CNDTN(1) EQ 2) AND (CNDTN(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 PRCARR(2).TOTARR + PRCARR(2).TOTDEF EQ 0.

THEN ATOTPRC = 0.5;

ELSE ATOTPRC = PRCARR(2).TOTARR/(PRCARR(2).TOTARR + PRCARR(2).TOTDEF);

ATOTPRC = FLOAT(FLOOR(ATOTPRC*100. + .5));

CONFLST.PCT_ARR = SUBSTR(F(ATOTPRC,\$THREE),1,3);

CURCONF = CONFIND(1); [current configuration index]

FIX(1,*,*) = FIXTRAV(CURCONF,*,*);

TFLAG = '0'B;

IF SUBSTR(ELGBLTY(1).ID, CURCONF,1) EQ '0'B

THEN [if current configuration is currently eligible, then continue]

CC = 0;

```

CALL CONSET;
IN  (CMFGRQ(CURCONF));
OUT (CC);
    [This routine sets variable CONFIGDATA which signifies runways in a configuration]
CONFIGDATA(1,*) = CC;
IF  (X EQ 2) AND  (SUBSTR(ELGSLTY(2).ID,CURCONF,1) EQ '1'B)
    THEN [if transition is from IFR to VFR and current configuration will be ineligible in
    VFR conditions]
        X = 1;
        Y = '1'B;
        TFLAG = '1'B;
XP = 1;
CALL DEMSET;
IN  (X, XP, CURCONF, INFORM, PECAER, CNVTDEM, CONFIGDATA, CMFGRQ);
OUT (DEM);
    [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 VARNUM1 = 10;
    ELSEIF CONFIGDATA(1,6) NE 0
        THEN VARNUM1 = 9;

```

```

      ELSE VARNUM1 = 8;
INT = 0;
LOOP; [R = 7 to 10] [set up variable INT time required for queue flush out for each departure
runway based on departure queue length]
  IF DEN(1,R) NE 0
    THEN INT(R-6) = (3600./DEN(1,R) * CNVTQLN (R_6));
ENDLOOP;
ROWMAX = 0;
LOOP; [G = 1 to 7] [set up number of runways in current configuration]
  IF CONFIGDATA(1,G) NE 0
    THEN ROWMAX = ROWMAX + 1;
ENDLOOP;
LOOP; [Q = 1 to 73]
  IF (SUBSTR(ELGSLTY(2).ID,Q,1) EQ '0'B) AND (Q NE CURCONF)
    THEN [if configuration Q is eligible in forecast environment and it is not same as
current configuration]
      FINCONF = Q; [final configuration index]
      CC = 0;
      CALL CONSET;
      IN (CNFGCQ(FINCONF));
      OUT (CC)
      [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 VARNUM2 = 10;
  ELSEIF CONFIGDATA(2,6) NE 0
    THEN VARNUM2 = 9;
    ELSE VARNUM2 = 8;
COLUMNMAX = 0;
LOOP:  [G = 1 to 7] [set up numbers of runways in final configuration]
  IF CONFIGDATA (2,G) NE 0
    THEN COLUMNMAX = COLUMNMAX + 1;
ENDLOOP;
IF TFLAG = '1'B [if transition is from IFR to VFR and current configuration
will be ineligible in forecast conditions]
  THEN
    XP = 2;
    CALL DEMSET;
    IN (X, XP, FINCONF, INFORM, PRICARR, CNVTDEM, CONFIGDATA,
CONFGRQ);
    OUT (DEM);
    [This routine computes demand values for each fix
    pertaining to final configuration]
    FIX(2,*,*) = FIXTRAV(FINCONF,*,*);
    CALL SPTRAN;

```

IN (DEM, VARNUM1, VARNUM2, INT, FIX);

OUT (TRAMTIME, TRAMCAP);

[special routine to compute transition capacity, by passing LP algorithm]

ELSEIF SUBSTR(ELGHLT(X).ID,Q,1) EQ '0'B

THEN [if transition is taking place in same environment: that is VFR to VFR or IFR to IFR and configuration Q is eligible]

XP = 2;

CALL DEMSET;

IN (X, XP, FINCONF, INFORM, PREARR, CNVTDEM, CONFIGDATA, CNFGRQ);

OUT (DEM)

[This routine computes demand values for each fix pertaining to final configuration]

CALL TDEP;

IN (CONDITION(X), CONFIGDATA, INT, DEPMAT);

OUT (MATDEP, TRAVTIM)

[This routine prepares dependence matrix]

LOOP: [I = 4 to 7] [if departure runways are same take average demand]

IF CONFIGDATA(1,I) EQ CONFIGDATA(2,I)

THEN

DEM(1,I+3) = (DEM(1,I+3) + DEM(2,I+3))/STWO;
DEM(2,I+3) = DEM(1,I+3);

ENDLOOP;

CALL CALC;

IN (ROWMAX, COLUMNMAX, VARNUM1, VARNUM2, CONFIGDATA,
MATDEF, DEM, TRAVTIM, FIX);

OUT (TRANSTIME, 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 IFR to VFR]

XP = 2;

XX = 2;

CALL DEMSET;

IN (XX, XP, FINCONF, INFORM, PRECARR, CNVTDEM,
CONFIGDATA, CNFGCRQ);

OUT (DEM);
[This routine computes demand values for
each fix pertaining to final configuration]

FIX(2,*,*) = FIXTRAV(FINCONF,*,*);

CALL SPTRAN;

IN (DEM, VARNUM1, VARNUM2, INT, FIX);

OUT (TRANSTIME, TRANCAP);
[This is a special routine to compute
transition capacity, bypassing LP algorithm]

ELSEIF SUBSTR(ELGBLTY(2), ID, CURCONF,1) EQ '0'B

THEN

XX = 2;

XP = 1;

CALL DEMSET;

IN (XX, XP, CURCONF, INFORM, PECARR,
CNVIDEM, CONFIGDATA, CNFGRQ)

OUT (DEM);

[This routine computes demand values
for each fix pertaining to current
configuration]

XX = 2;

XP = 2;

CALL DEMSET;

IN (XX, XP, FIMCONF, INFORM, PECARR,
CNVIDEM, CONFIGDATA);

OUT (DEM);

[This routine computes demand values
for each fix pertaining to final
configuration]

FIX(2,*,*) = FIXTRAV(FIMCONF,*,*);

CALL TDEP;

IN (CONDITION(X), CONFIGDATA,
INT, DEPMAT);

OUT (MATDEP, TRAVTIM);

[This routine prepares
dependence matrix]

LOOP [I = 4 to 7] [if departure
runways are same take average
demand]

IF CONFIGDATA (1,I) EQ CONFIGDATA(2,I)

THEN

DEM(1,I+3) = (DEM(1,I+3) +
DEM(2,I+3))/2;
DEM(2,I+3) = DEM(1,I+3);

ENDLOOPCALL CALC;

IN (ROWMAX, COLUMNMAX, VARNUM1, VARNUM2,
CONFIGDATA, MATDEP, DEM, TRAVTIM, FIX);

OUT (TRANSTIME, TRANCAP);
[This routine performs LP computation
and determines transition duration and
capacity]

ELSE

XP = 2;
XX = 2;

CALL DEMSET;

IN (XX, XP, FINCONF, INFORM, PECARR,
CNVIDEN, CONFIGDATA, CNFGREQ);

OUT (DEM);
[This routine demand values for each
fix pertaining to final configuration]

FIX (2, *, *) = FIXTRAV(FINCONF, *, *);

CALL SPTRAN;

IN (DEM, VARNUM1, VARNUM2, INT, FIX);

OUT (TRANSTIME, TRANCAP);
[This is a special routine to compute
transition capacity, bypassing LP
algorithm]


```

CNVT_LIST(Q).INDEX = Q;
CNVT_LIST(Q).MINUTES = TRANSTIME;

CNVT_LIST(Q).FINC = INFORM(2).CONF(FINCONF).
CAPACITY;

TOTCAP = INFORM(2).CONF(FINCONF).CAPACITY;
TOTCAP = TOTCAP * (60 - TRANSTIME)/60;
LIST(Q).CAP = TOTCAP + TRANCAP;
LIST(Q).INDEX = Q;

```

ELSE

```

LIST(Q).CAP = -1.0;
LIST(Q).INDEX = 999;
CNVT_LIST(Q).INDEX = 0;
CNVT_LIST(Q).MINUTES = 0;
CNVT_LIST(Q).FINC = 0;

```

ENDLOOP;CALL OSORT;IMOUT (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;

```

```

TRANLST.CONFIG(COUNT).RANK = SUBSTR(F(FLOAT(COUNT), $THREE),1,3)
TRANLST.CONFIG(COUNT).ARR(1) = CNFGRQ(P).ARR_RWY(1);
TRANLST.CONFIG(COUNT).ARR(2) = CNFGRQ(P).ARR_RWY(2);
TRANLST.CONFIG(COUNT).ARR(3) = CNFGRQ(P).ARR_RWY(3);
TRANLST.CONFIG(COUNT).DEP(1) = CNFGRQ(P).DEP_RWY(1);
TRANLST.CONFIG(COUNT).DEP(2) = CNFGRQ(P).DEP_RWY(2);
TRANLST.CONFIG(COUNT).DEP(3) = CNFGRQ(P).DEP_RWY(3);
TRANLST.CONFIG(COUNT).DEP(4) = CNFGRQ(P).DEP_RWY(4);
TRANLST.CONFIG(COUNT).HOURLY = F(LIST(N).CAP,$FOUR);
TRANLST.CONFIG(COUNT).FIMCAP = F(CNVT LIST(P).FIMC,$FOUR);
TRANLST.CONFIG(COUNT).MINUTES = SUBSTR(F(CNVT_LIST(P).MINUTES, $TWO),1,2);

ENDLOOP;

IF SUBSTR(ELGBLTY(2).ID, CURCONF,1) EQ '0'B
    THEN COUNT = COUNT + 1;

TRANLST.NUM_ELIG = SUBSTR(F(FLOAT(COUNT,$THREE),1,3);

E = ELGBLTY(2).NUM;
ELGBLTY(2).NUM = COUNT;

ELSE [current configuration is ineligible]

TRANLST.MSG = SUBSTR(TRANLST.MSG, 1, 20) CONCATENATE ' ***CURRENT CONFIGURATION IS
INELIGIBLE***';

TRANLST.NUM_ELIG = SUBSTR(F(FLOAT(ELGBLTY(2).NUM),$THREE),1,3);

TRANLST.ARR(1) = CNFGRQ(CURCONF).ARR_RWY(1);
TRANLST.ARR(2) = CNFGRQ(CURCONF).ARR_RWY(2);
TRANLST.ARR(3) = CNFGRQ(CURCONF).ARR_RWY(3);
TRANLST.DEP(1) = CNFGRQ(CURCONF).DEP_RWY(1);
TRANLST.DEP(2) = CNFGRQ(CURCONF).DEP_RWY(2);
TRANLST.DEP(3) = CNFGRQ(CURCONF).DEP_RWY(3);
TRANLST.DEP(4) = CNFGRQ(CURCONF).DEP_RWY(4);

IF SUBSTR(ELGBLTY(1).ID,CURCONF,1) EQ '0'B

```

```
      THEN TRANLST.CTRANHR = F(INFORM(1).CONF(CURCONF).CAPACITY,$FOUR);  
      ELSE TRANLST.CTRANHR = (5)'b';  
IF SUBSTR(ELGBLTY(2).ID.CURCONF,1) EQ '0'B  
      THEN TRANLST.CFINCAP = F(INFORM(2).CONF(CURCONF).CAPACITY,$FOUR);  
      ELSE TRANLST.CFINCAP = (5)' ';  
TRANLST.SCROLL = (4)' ';  
END TRAN;
```

ROUTINE CONSETIN (CNFGRQ (Q));OUT (CC);

[This routine sets variable CONFIGDATA which signifies runways in a configuration]

CC = 0;

K1 = 0;

K2 = 3;

LOOP; [P = 1 to 12]IF SUBSTR(CNFGRQ(Q).ID,P,1) EQ '1'BTHEN

K1 = K1 + 1;

CC(K1) = P;

IF SUBSTR(CNFGRQ(Q).ID,P+12,1) EQ '1'BTHEN

K2 = K2 + 1;

CC(K2) = P;

ENDLOOP;END CONSET;

ROUTINE DEMSETIN (X, XP, INDEX, INFORM, PRCARR, CNVTDEM, CONFIGDATA, CNFGRQ);OUT (DEM);

[This routine computes demand at fixes for current and forecast configurations]

DDEM(1) = CNVTDEM(X).DEP.NORTH;
 DDEM(2) = CNVTDEM(X).DEP.EAST;
 DDEM(3) = CNVTDEM(X).DEP.SOUTH;
 DDEM(4) = CNVTDEM(X).DEP.WEST;
 DDEM(5) = CNVTDEM(X).DEP.MKE_D;

DISDEM = 0;

K2 = 3;

LOOP; [P = 1 to 12]IF SUBSTR(CNFGRQ(INDEX).ID,P+12,1) EQ '1'BTHEN

K2 = K2 + 1;

LOOP; [J = 1 to 5]IF CNFGRQ(INDEX).DEP(J) EQ PTHEN DISDEM (K2 - 3) = DISDEM (K2 - 3) + DDEM(J);ENDLOOP;ENDLOOP;

ARRCAP = INFORM(X).CONF(INDEX).NARRCAP + INFORM(X).CONF(INDEX).SARRCAP;
 DEPCAP = INFORM(X).CONF(INDEX).SDEPCAP + INFORM(X).CONF(INDEX).NDEPCAP;

IF PRCARR(X).TOTARR + PRCARR(X).TOTDEP EQ 0THEN ATOTPRC = .5;ELSE ATOTPRC = PRCARR(X).TOTARR/(PRCARR(X).TOTARR + PRCARR(X).TOTDEP);

```

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);
DEM(XP,1) = ARRCAP*CNVTDEM(X).ARR.KUBBS/PRCARR(X).TOTARR;
DEM(XP,2) = ARRCAP*CNVTDEM(X).ARR.PLANT/PRCARR(X).TOTARR;
DEM(XP,3) = ARRCAP*CNVTDEM(X).ARR.CGT/PRCARR(X).TOTARR;
DEM(XP,4) = ARRCAP*CNVTDEM(X).ARR.VAINS/PRCARR(X).TOTARR;
DEM(XP,5) = ARRCAP*CNVTDEM(X).ARR.FARM/PRCARR(X).TOTARR;
DEM(XP,6) = ARRCAP*CNVTDEM(X).ARR.MKE_A/PRCARR(X).TOTARR;
IF PRCARR(X).TOTDEP GT 0
  THEN
    REPEAT WHILE CONFIGDATA(XP, K+3) NE 0 [K = 1 To 4]
      DEM(XP, K+6) = DEPCAP*DISDEM(*)/PRCARR(X).TOTDEP;
    ENDREPEAT;
END DEMSET;

```

ROUTINE TDEPIN (X, CONFIGDATA, INT, DEPMAT);OUT (MATDEP, TRAVTIM);

[This routine computes dependence matrix]

MATDEP = 0;

LOOP [G = 1 To 7]IF CONFIGDATA(1, G) NE 0THENLOOP; [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 0THEN

MATDEP(G,H) = DEPMAT(X).SECT(JJ).MATRIC(CONFIGDATA(1,G), CONFIGDATA(2,H));

ENDLOOP;ENDLOOP;

2-365

```
TRAVTIM = 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 MATDEP(3 + R, RR) NE 0
        THEN
          MATDEP(3 + R, RR) = MATDEP(3 + R, RR) + INT(R);
          QFLAG = 1;
      ENDLOOP;
    IF QFLAG EQ 0
      THEN TRAVTIM(R) = INT(R);
    ENDLOOP;
  END TDEP;
```


2-366

ROUTINE SPYRAM

IN (DEM, VARNUM1, VARNUM2, INT, FIX);

OUT (TRANSTIME, TRAMCAP);

[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: [M = 1 to 3]

LOOP: [L = 1 to 6]

IF FIXTRAV(1,M,L) GT TEMP

THEN

TEMP = FIXTRAV(1,M,L);
RUNIND = M;
FIXIND = L;

ENDLOOP;

ENDLOOP;

CURTIME = TEMP;

LOOP: [M = 1 To 6]

IF M NE FIXIND

THENIF CURTIME EQ FIXTRAV (1, RUNIND, M)

```

      THEN CURTIME = CURTIME + SAFESEP; [contribution of current configuration to transition
                                         duration]

ENDLOOP;

FIXTRAV (1, RUNIND, FIXIND) = CURTIME;

TEMP = 0;

LOOP; [M = 1 to 4]
    TEMP = MAX(TEMP, INT(M));
ENDLOOP;

CURTIME = MAX(CURTIME, TEMP); [include effect of departure queues]

IF CURTIME GE 60.
    THEN CURTIME = 59;
    TEMP = 0;
    LOOP; [M = 1 to 3]
        LOOP; [L = 1 to 6]
            IF FIXTRAV(2, M, L) GT TEMP
                THEN
                    TEMP = FIXTRAV(2, M, L);
                    RUNIND = M;
                    FIXIND = L;
        ENDLOOP;
    ENDLOOP;
    FINTIME = TEMP;
    LOOP; [M = 1 to 6]

```

```

IF M NE FIXIND
THEN
  IF FINTIME EQ FIXTRAV(2,RUNIND,M)
  THEN FINTIME = FINTIME + SAFESKP; [contribution of current configuration to
                                     transition duration]
ENDLOOP;
FIXTRAV(2,RUNIND, FIXIND) = FINTIME;
TRANSTIME = MAX(FINTIME, CURTIME); [transition duration]
IF TRANSTIME EQ CURTIME [if transition duration is driven from current configuration]
THEN
  [compute variables for current configuration]
  VAR1 = 0;
  LOOP; [L = 1 to 6]
    LOOP; [L = 1 to 6]
      IF FIXTRAV (1,M,L) NE 0
      THEN VAR1(L) = FIXTRAV(1,M,L);
    ENDLOOP;
  ENDLOOP;
  LOOP; [M = 7 to VARNUM1]
    VAR1(M) = CURTIME
  ENDLOOP;
  TRANCAP = 0;
  LOOP; [T = 1 to VARNUM1]

```

```

      TRANCAP = TRANCAP + DEM1(T)*VARI(T)/60.; [compute transition capacity]
    ENDLOOP;
  ELSE [if transition duration is driven from final configuration]
    VARI = 0;
    LOOP; [M = 1 to 3]
      LOOP; [L = 1 to 6]
        IF FIXTRAV(1,M,L) NE 0
          THEN VARI(L) = FIXTRAV(1,M,L);
        ENDLOOP;
      ENDLOOP;
    LOOP; [M = 7 to VARNUM1] [compute variables for current configuration]
      VARI(M) = CURTIME;
    ENDLOOP;
    COUNT = 1;
    LOOP; [M = 1 to 3]
      LOOP [L = 1 to 6]
        IF FIXTRAV(2,M,L) NE 0
          THEN
            IF FIXTRAV(2,M,L) GT CURTIME
              THEN F(COUNT) = L;
            COUNT = COUNT + 1;

```

```
ENDLOOP;  
ENDLOOP;  
[compute transition capacity]  
TRANCAP = 0.;  
LOOP; T = 1 to VARNUM1  
    TRANCAP = TRANCAP + DEM1(T)*VAR1(T)/60.;  
ENDLOOP;  
CONST = FINTIME - CURTIME;  
LOOP; [T = 1 to VARNUM2]  
    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 SPTRAN;
```

ROUTINE CALC

IN (ROWMAX, COLUMNMAX, VARNUM1, VARNUM2, CONFIGDATA, DEPMAT, DEM, TRAVTIM, FIX);

OUT (TRANSTIME, TRANCAP);

[This routine computes LP solution using special algorithm, also transition duration is computed]

CURCONF = 1;

FINCONF = 2;

FIXTRAV = FIX;

DEPMAT = DEPMAT/60.; [conversion from seconds to minute]

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 (TRAVTIM, CONFIGDATA, DEPMAT, DEM);

OUT (TRANSTIME, TRANCAP);

[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 (TFLAG, TRANSTIME);

[This routine computes transition duration]

TRAVTIM = TRAVTIM/60.; [conversion from seconds to minutes]

IF SUM(TRAVTIM) NE 0 [modify transition duration with information on current departure
queues]

THEN

COMPAREX = 0;

```

      LOOP; [W = 1 to 4]
        IF TRAVTIM(W) GT COMPAREX
          THEN COMPAREX = TRAVTIM(W)
        ENDLOOP;
        IF COMPAREX GE 60.
          THEN COMPAREX = 59;
        IF TFLAG = 0;
          THENIF COMPAREX GT TRANSTIME
            THEN TRANSTIME = COMPAREX;
          ELSEIF COMPAREX GT (TRANSTIME - SAFESEP)
            THEN TRANSTIME = COMPAREX;
        IF TRANSTIME GE 60;
          THEN TRANSTIME = 59;
      IF (CONFIGDATA(1,1) NE CONFIGDATA(2,1)) OR (CONFIGDATA(1,2) NE CONFIGDATA(2,2)) OR
        (CONFIGDATA(1,3) NE CONFIGDATA(2,3))
      THEN
        EDM = 0;
        CALL EDP;
        IN (ROWMAX, COLUMNMAX, DEPMAT, CONFIGDATA, FIXTRAV);
        OUT (EDM);
        [This routine prepares expanded dependence matrix]
        [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 UB1(I) = TRANSTIME - MAXDELAY(J) - FIXTRAV(CURCONF,J,I)
    IF ABS(UB1(I)) LT .001
      THEN UB1(I) = 0
  ENDLOOP;
ENDLOOP;
IF CONFIGDATA(1,3) NE 0
  THEN I3 = 3;
```


AU-A127 828

SOFTWARE DESCRIPTION FOR THE O'HARE RUNWAY
CONFIGURATION MANAGEMENT SYSTEM (U) MITRE CORP MCLEAN
VA METREX DIV 3 KAYOUSSI OCT 82 MTR-82M125-VOL-2
FAA-EM-82-28-VOL-2 DTFA01-81-C-10003

F/G 17/7

NL

515

UNCLASSIFIED

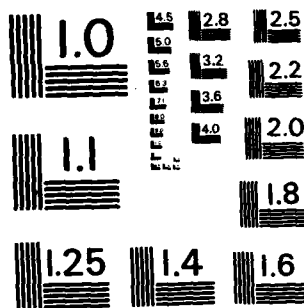
END

DATE

FILED

6 83

DT



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

```
ELSE I3 = 4;
LOOP; [I = 7 to VARNUM1]
    UB1(I) = TRANSTIME - MAXDELAY(I - I3) - TRAVTIN(I - 6)
ENDLOOP;
[determine initial solutions for LP variables]
LOOP; [M = 1 to VARNUM1]
    COMPARE = 0;
    LOOP; [N = 1 to VARNUM1]
        IF EDM(N,M) GT COMPARE
            THEN COMPARE = EDM(N,M);
    ENDLOOP;
    VAR2(M) = COMPARE;
ENDLOOP;
LOOP; [M = 1 to VARNUM1]
    COMPARE = TRANSTIME;
    LOOP [N = 1 to VARNUM2]
        IF (M EQ N) OR (EDM(M,N)) NE 0
            THEN
                INDEX = MIN((VAR2(N) - EDM(M,N)), UB1(N));
            IF INDEX LT COMPARE
                THEN COMPARE = INDEX;
    ENDLOOP;
```

VAR1(M) = 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 = "B;

LOOP; [I = 1 To VARNUM2]

    TEMP = TEMP CONCATENATE '0'B;

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 ALT1(I,J) EQ 1
      THEN
        COUNTER = COUNTER + 1;
        ALT1(I,J) = -1;
        LOOP; [K = 1 to VARNUM 2]
          IF ALT1(I,K) NE 0
            THEN
              COL(COUNTER) = COL(COUNTER) CONCATENATE '1'B
              ALT1(I,K) = -1;
            ELSE COL(COUNTER) = COL(COUNTER) CONCATENATE '0'B;
          ENDLOOP;
        LOOP; [K = 1 to VARNUM1]
          IF ALT1(K,J) NE 0
            THEN
              ROW(COUNTER) = ROW(COUNTER) CONCATENATE '1'B;
              ALT1(K,J) = -1;
            ELSE ROW(COUNTER) = ROW(COUNTER) CONCATENATE '0'B;
          ENDLOOP;
        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;

ENDLOOP; .

ST = "B

LOOP; [R = 1 to VARNUM1]

LOOP; [Q = 1 to VARNUM 2]

IF ALT(R,Q) EQ 1

THEN ST(R) = ST(R) CONCATENATE '1'B;

ELSE ST(R) = ST(R) CONCATENATE '0'B;

ENDLOOP;

ENDLOOP;

LOOP; [I = 1 to COUNTER]

ROWSTCK = 0;

COLCOUNT = 0;

COUNT = 0;

LOOP; [Q = 1 to VARNUM1]

IF SUBSTR(ROW(I),Q,1) EQ '1'B

THEN

COUNT = COUNT + 1;

ROWSTCK(COUNT) = Q;

ENDLOOP;

LOOP; [R = 1 to COUNT]

Q = ROWSTCK(R);

COLCOUNTER = 0;

LOOP; [S = 1 to VARNUM2]

IF ALT(Q,S) EQ 1

THEN COLCOUNTER = COLCOUNTER + 1;

ENDLOOP;

COLCOUNT(R) = COLCOUNTER;

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
        TTEMP1 = COLCOUNT(QQ-1);
        TTEMP2 = ROWSTCK(QQ-1);
        COLCOUNT(QQ-1) = COLCOUNT(QQ);
        ROWSTCK(QQ-1) = ROWSTCK(QQ)
        COLCOUNT(QQ) = TTEMP1;
        ROWSTCK(QQ) = TTEMP2;
        QQ = QQ - 1;
      ELSE SORT_FLAG = 1;
    ENDREPEAT;
  ENDLOOP;
  SUBCOUNT = COUNT;
  PSUDEN = DEM1;
  ALTROWSTCK = 0
  IND3 = 0;
  LOOP; [Q = 1 to COUNT]
    IF PSUDEN(ROWSTCK(Q)) NE 0
      THEN
        IND3 = IND3 + 1;
        ALTROWSTCK(IND3) = ROWSTCK(Q);
        QQ = Q;

```

```

      LOOP; [R = QQ + 1 to COUNT]
        IF ST(ROWSTCK(Q)) EQ ST(ROWSTCK(R))
          THEN
            PSUDEN(ROWSTCK(Q)) = PSUDEN(ROWSTCK(Q)) + DEM1(ROWSTCK(R));
            PSUDEN(ROWSTCK(R)) = 0;
            SUBCOUNT = SUBCOUNT - 1;
          ENDLOOP;
        ENDLOOP;

      ROWSTCK = ALTROWSTCK;
      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;
        ENDLOOP;

      IF ST(ROWSTCK(1)) NE COL(1)

```

```
      THEN
        IND1 = IND1 + 1;
      LOOP; [Q = 1 to COUNT]
        X(IND1, Q) = ROWSTCK(Q);
      ENDLOOP;

SFLAG = 0;
REPEAT WHILE (SFLAG EQ 0); [R = 1 to IND1]
  ADD1 = 0;
  ADD2 = 0;
  NEWST = "B";
  REPEAT WHILE (X(R,Q) NE 0); [Q = 1 to IND2]
    ADD1 = ADD1 + PSUDEN(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.;
      UPSLCK2 = 9999.;
      MINSLCK = 9999;
    REPEAT WHILE (X(R,Q) NE 0); [Q = 1 to 10]
```

LOOP [W = 1 to VARNUM2]

IF SUBSTR(ST(X(R,Q),W,1) EQ '0'B

THEN

IF (EDM(X(R,Q),W) NE 0) OR X(R,Q) EQ W)

THEN

SLCK = VAR2(W) - VAR1(W) - EDM(X(R,Q),W);

MINSLCK = MIN(SLCK,MINSLCK);

ELSE

SLCK = UB2(W) - VAR2(W);

UPSLCK2 = MIN(SLCK,UPSLCK2);

ENDLOOP;

SLCK = UB1(X(R,Q) - VAR1(X(R,Q));

UPSLCK1 = MIN(SLCK,UPSLCK1);

ENDREPEAT;

ABSMIN = MIN(MINSLCK,UPSLCK1,UPSLCK2);

IF ABSMIN GT .001

THEN

FLAG = 1;

SFLAG = 1;

REPEAT WHILE (X(R,Q) NE 0) [Q = 1 to VARNUM1]

VAR1(X(R,Q)) = VAR1(X(R,Q)) + ABSMIN;

ENDREPEAT;

LOOP; [W = 1 to VARNUM2]

IF SUBSTR(NEWST,W,1) EQ '1'B

THEN VAR2(W) = VAR2(W) + ABSMIN;

ENDLOOP

ENDREPEAT;

ENDLOOP;

ENDREPEAT;

END LP_ALGORITHM_ITERATION;

ROUTINE DURIN (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 0THENLOOP [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(1);
      IF COMPARE2 GT COMPARE1
        THEN
          COMPARE1 = COMPARE2;
          FIX_INDICATOR1 = FIX_INDICATOR;
          RUN_INDICATOR1 = RUN_INDICATOR
        ENDREPEAT;
      CURTIME = COMPARE1;
    LOOP; [I = 4 to 7]
      IF CURTIME LT MAXDELAY(1)
        THEN CURTIME = MAXDELAY(1);
      ENDLOOP;
      [compute contribution of final configuration to transition duration]
      COMPARE1 = 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;
        ENDREPEAT;
      FINTIME = COMPARE1;
      [compute transition duration]
      IF CURTIME GT FINTIME
        THEN
          TRANSTIME = CURTIME;
          FIX_INDICATOR = FIX_INDICATOR1;
          RUN_INDICATOR = RUN_INDICATOR1;
          DUMMY = TRANSTIME_MAXDELAY(RUN_INDICATOR1);
          LOOP; [I = 1 to 6]
            IF (DUMMY EQ FIXTRAV(CURCONF,RUN_INDICATOR1,I)) AND I NE FIX_INDICATOR1
              THEN
                TRANSTIME = TRANSTIME + SAFESEP;
                TFLAG = 1;
              ENDLOOP;
            ENDLOOP;

```


ELSEIF FINTIME GT CURTIME

THEN

TRANSTIME = FINTIME;
DUMMY = TRANSTIME;

LOOP; [I = 1 to 6]

IF (DUMMY EQ FIXTRAV(FINCONF, RUN_INDICATOR1, I)) AND (I NE FIX_INDICATOR2)

THEN

TRANSTIME = TRANSTIME + SAFESEP;
TFLAG = 1;

ENDLOOP;

ELSE

TRANSTIME = CURTIME;
DUMMY = TRANSTIME - MAXDELAY(RUN_INDICATOR1);

LOOP; [I = 1 to 6]

IF (DUMMY EQ FIXTRAV(CURCONF, RUN_INDICATOR1, I)) AND (I NE FIX_INDICATOR1)

THEN

TRANSTIME = TRANSTIME + SAFESEP;
TFLAG = 1;

ENDLOOP;

DUMMY = TRANSTIME;

LOOP; [I = 1 to 6]

IF (DUMMY EQ FIXTRAV(FINCONF, RUN_INDICATOR2, I)) AND (I NE FIX_INDICATOR2)

THEN

TRANSTIME = TRANSTIME + SAFESEP;
TFLAG = 1;

ENDLOOP;

END DUB;

ROUTINE ADJSTIN (TRAVTIM, 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 COMPARETHEN COMPARE = DEPMAT(I,J);ENDLOOP;

MAXDELAY(I-3) = COMPARE;

ENDLOOP;

ITEMP = 0;

LOOP; [I = 1 to 4]

TEMP = TRAVTIM(I) + MAXDELAY(I);

ITEMP = MAX(ITEMP, TEMP);

ENDLOOP;

TRANSTIME = ITEMp;

IF TRANSTIME NE 0;

THEN

CAP = 0;

LOOP; [I = 1 to 10]

CAP = CAP + DEN(I, I);

ENDLOOP;

TRANCAP = CAP * (TRANSTIME/60);

END ADJUST;

ROUTINE EDPIN (ROWMAX, COLUMNMAX, DEPMAT, CONFIGDATA, FIXTRAV);OUT (EDM);

[This routine prepares expanded dependence matrix]

CURCONF = 1;

FINCONF = 2;

SAFESEP = 3;

[initialization]

IF CONFIGDATA (CURCONF,3) NE 0THEN I1 = 3; [number of arrival runways in current configuration]ELSE I1 = 2;IF CONFIGDATA(CURCONF,7) NE 0THEN I2 = 4;ELSEIF CONFIGDATA(CURCONF, 6) NE 0THEN I2 = 3;ELSE I2 = 2;IF CONFIGDATA(FINCONF, 3) NE 0THEN J1 = 3; [number of arrival runways in final configuration]ELSE J1 = 2;IF CONFIGDATA(FINCONF, 7) NE 0THEN J2 = 4;ELSEIF CONFIGDATA(FINCONF, 6) NE 0

```
      THEN J2 = 3;
      ELSE J2 = 2;

IF I1 EQ 2
      THEN I3 = 4;
      ELSE I3 = 3;

IF J1 EQ 2
      THEN J3 = 4;
      ELSE J3 = 3;

LOOP; [I = 1 to I1]
      INDEX1 = 1;
      LOOP; [K = 1 to 6]
          IF FIXTRAV(CURCONF, I, K) NE 0
              THEN
                  RECORD1 (INDEX1) = K;
                  INDEX1 = INDEX1 + 1;

      ENDLOOP;

      LOOP; [J = 1 to J1]
          INDEX2 = 1;
          LOOP; [K = 1 to 6]
              IF FIXTRAV(CINCONF, J, K) NE 0
                  THEN
                      RECORD2 (INDEX2) = K;
                      INDEX2 = INDEX2 + 1;
```

```

ENDLOOP;
LOOP; [L = 1 to INDEX1 - 1]
  LOOP; [M = 1 to INDEX2 - 1]
    EDM(RECORD1(L), RECORD2(M)) = DEPMAT(I, J);
  ENDLOOP;
ENDLOOP;
ENDLOOP;
LOOP; [L = I1 + 1 to ROWMAX]
  LOOP; [K = 1 to COLUMNMAX]
    IF K LE J1
      THEN;
        LOOP; [M = 1 to 6]
          IF FIXTRAV(FINCONF, K, M) NE 0
            THEN
              EDM(L+I3, M) = DEPMAT(3+L-I1, K);
          ENDLOOP;
        ENDLOOP;
      ENDLOOP;
    ENDLOOP;
  ENDLOOP;

```

```

      IF K LE I1
      THEN
        LOOP; [M = 1 to 6]
          IF FIXTRAV(CURCONF,K,M) NE 0
            THEN EDM(M,L + J3) = DEPMAT(K,L+3-J1);
          ENDLOOP;
        ENDLOOP;
      ENDLOOP;
    LOOP; [L = I1 + 1 to ROWMAX]
      LOOP; [N = J1 + 1 to COLUMNMAX]
        EDM(L + I3, N + J3) = DEPMAT(L+3-I1, N+3-J1);
      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 11]
        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 6+J2]
        IF (EDM(I,J) NE 0) AND (I NE J)
          THEN
            LOOP; [K = 1 to 11]
              IF FIXTRAV(CURCONF,K,I) NE 0
                THEN EDM(I,J) = EDM(I,J) + FIXTRAV(CURCONF,K,I);
              ENDLOOP;
            ENDLOOP;
          ENDLOOP;
        END EDP;
```



```

FUNCTION OBJFUN
  IN (DEM1, DEM2, TRANSTIME, VAR1, VAR2, FIXTRAV, VARNUM1, VARNUM2,
      CONFIGDATA, TRAVTIM);
  OUT (TRANCAP);
  [This routine computes value of 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) * DEM1(I) / $SIXTY;
            ENDLOOP;
          ELSE
            CAP = CAP + (TRAVTIM(I-6) + VAR1(I)) * DEM1(I) / $SIXTY;
          ENDLOOP;
        ENDLOOP;
      TRANCAP = CAP;
    END OBJFUN;

```

2.15 Configuration Information Screen

The Configuration Information Screen is presented on pages 2-398 to 2-420.

2-398

LOCAL VARIABLES

STRUCTURE CNFG_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 DEP(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]

GROUP DEPPTR DEM [pointer for total departure demand data field]PTR CAP [pointer for total departure capacity data field]GROUP NORTHPTR PCT_ARR [pointer for north percentage of arrivals data field]PTR SAT [pointer for north saturation data field]GROUP ARRPTR DEM [pointer for north arrival demand data field]PTR CAP [pointer for north arrival capacity data field]GROUP SOUTHPTR PCT_ARR [pointer for south percentage of arrivals data field]PTR SAT [pointer for south saturation data field]GROUP ARRPTR DEM [pointer for south arrival demand data field]PTR CAP [pointer for south arrival capacity data field]GROUP DEMPTR DEM [pointer for south departure demand data field]PTR CAP [pointer for south departure capacity data field]GROUP BALANCINGPTR AMOVE [pointer for arrival aircraft moved data field]PTR ACOMPLEX [pointer for complex data field]

2-400

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's]
ENDSTRUCTURE;

ROUTINE CNFG

IN (CONFIG, CNFGRQ, PRCARR, INFORM, MIDFLAG, RWYEQP);

INOUT (CONFIND, RSTATUS, I);

[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); [switch between two screens]

REPEAT UNTIL (RSTATUS NE PF9);

I = SWITCH(I);

CALL CSCREEN;

IN (CNFG_DATA(I), CINDEX(I), CNFGRQ, PRCARR(I), INFORM(I), MIDFLAG(I), RWYEQP(I));

INOUT (RSTATUS);

[This routine controls configuration information screen]

ENDREPEAT;

ENDREPEAT;

LOOP; [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 CSCREEN

IN (CNFG_DATA(I), CNFCRQ, PRCARR(I), PRCARR(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 CNFG_LIST(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 CHR [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 '0' B  
  THENIF INFORM(I).CONF(CINDEX(I)).INDEX LT 999  
    THEN  
      PERFORM OUTPUT_SET_UP_(TOTAL);  
      PERFORM OUTPUT_SET_UP_(NORTH);  
      PERFORM OUTPUT_SET_UP_(SOUTH);  
      PERFORM OUTPUT_SET_UP_(BALANCING_ARRIVALS);  
      PERFORM OUTPUT_SET_UP_(BALANCING_DEPARTURE);  
      PERFORM OUTPUT_SET_UP_(OTHERS);  
    ELSE  
      LOOP; [J = 29 to 62]  
        TM(J) = FLDDARK;  
      ENDLOOP;  
      LOOP; [J = 26 to 49]  
        DM(J) = FLDDARK;  
      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;
TM = FLDDEF;
DM = FLDDEF;
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(J).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 to 62]  
            TM(J) = FLDDARK;  
  
      ENDLOOP;  
  
      LOOP; [J = 26 to 50]  
            DM(J) = FLDDARK;  
  
      ENDLOOP;  
  
    ELSE  
  
      CALL CUPDATE;  
  
      INOUT (CNFG_DATA(I));  
            [This routine performs local updates on screen]  
  
      CNFG_DATA(I).MSG = 'DATA ENTERED AT ' CONCATENATE GMT;  
  
      AUX_DATA = CNFG_DATA(I), BY NAME;  
  
    ENDREPEAT;  
  
  END CSCREEN;
```

PROCESS INITIALIZATION;
[This process performs a number of necessary initializations]

```
$THREE = 3;  
FLAG = '0'B;
```

```
DM = FLDDDEF;  
TM = FLDDDEF;  
DM(1) = FLDDHIGH;  
DM(51) = FLDDHIGH;
```

CNFG_LIST = CNFGRQ, BY NAME;

CURSOR = 2;

LOOP; {J = 1 to 12}

```
IF SUBSTR(CNFGRC(CINDEX(I)).ID,J,1) EQ '1'B
```

```

THEN
  CNFG DATA(I).CONF.ARR(J) = 'X'

```

```
ELSE CNFG_DATA(I).CONF.ARR(J) = (2)' ';
```

```
IF SUBSTR(CINDEX(I)).ID,J+12,1) EQ '1'B;
```

```
THEN CNFG_DATA(I).CONF.DEP(J) = 'X ';
```

```
ELSE CNFG_DATA(1).CONF.DEF(J) = (2)' ';
```

ENDLOOP;

```
MIDIND = '110000000001000010010000'B;
MID(1).NUM = '100000000000000000000000';
MID(2).NUM = '01000000000000000000000000';
MID(3).NUM = '00000000000100000000000000';
MID(4).NUM = '0000000000000000000100000000';
MID(5).NUM = '00000000000000000000010000';
```

MID(1).CHR = '4R ARR';
MID(2).CHR = '4L ARR';
MID(3).CHR = '32L ARR';
MID(4).CHR = '14R DEP';
MID(5).CHR = '22L DEP';

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 DMS use]

  CONFIG_LOADLIST.TIME = ADDR(CNFG_DATA(I).TIME);

  LOOP;      [J = 1 to 12]

    CONFIG_LOADLIST.CONF.ARR(J) = ADDR(CNFG_DATA(I).CONF.ARR(J));
    CONFIG_LOADLIST.CONF.ARR(J) = ADDR(CNFG_DATA(I).CONF.DEP(J));

  ENDLLOOP;

  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_ARR = ADDR(CNFG_DATA(I).NORTH.PCT_ARR);
  CONFIG_LOADLIST.NORTH.ARR.DEM = ADDR(CNFG_DATA(I).NORTH.ARR.DEM);
  CONFIG_LOADLIST.NORTH.ARR.CAP = ADDR(CNFG_DATA(I).NORTH.ARR.CAP);
  CONFIG_LOADLIST.NORTH.DEP.DEM = ADDR(CNFG_DATA(I).NORTH.DEP.DEM);
  CONFIG_LOADLIST.NORTH.DEP.CAP = ADDR(CNFG_DATA(I).NORTH.DEP.CAP);
  CONFIG_LOADLIST.SOUTH.SAT = ADDR(CNFG_DATA(I).SOUTH.SAT);
  CONFIG_LOADLIST.SOUTH.PCT_ARR = ADDR(CNFG_DATA(I).SOUTH.PCT_ARR);
  CONFIG_LOADLIST.SOUTH.ARR.DEM = ADDR(CNFG_DATA(I).SOUTH.ARR.DEM);
  CONFIG_LOADLIST.SOUTH.ARR.CAP = ADDR(CNFG_DATA(I).SOUTH.ARR.CAP);
  CONFIG_LOADLIST.SOUTH.DEP.DEM = ADDR(CNFG_DATA(I).SOUTH.DEP.DEM);
  CONFIG_LOADLIST.SOUTH.DEP.CAP = ADDR(CNFG_DATA(I).SOUTH.DEP.CAP);
  CONFIG_LOADLIST.BALANCING.AMOVE = ADDR(CNFG_DATA(I).BALANCING.AMOVE);
  CONFIG_LOADLIST.BALANCING.ACOMPLEX = ADDR(CNFG_DATA(I).BALANCING.ACOMPLEX);
  CONFIG_LOADLIST.BALANCING.DMOVE = ADDR(CNFG_DATA(I).BALANCING.DMOVE);
  CONFIG_LOADLIST.BALANCING.DCOMPLEX = ADDR(CNFG_DATA(I).BALANCING.DCOMPLEX);
  CONFIG_LOADLIST.WMSG0 = ADDR(CNFG_DATA(I).WMSG0);
  CONFIG_LOADLIST.WMSG1 = ADDR(CNFG_DATA(I).WMSG1);
  CONFIG_LOADLIST.WMSG2 = ADDR(CNFG_DATA(I).WMSG2);
  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 PRCARR(I).CONF(CINDEX(I)).BNPRCNT LT 0.

THEN

CNFG_DATA(I).WMSG0 = (29)' ' CONCATENATE'*** SATURATED ***'

LOOP; {J = 57 to 62}

TH(J) = FLDDARK;

ENDLOOP;

LOOP; {J = 44 To 47}

DM(J) = FLDDARK;

ENDLOOP;

IF INFORM(I).CONF(CINDEX(I)).SATURATION EQ 0.

THEN CNFG_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);

CNFG_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);

CNFG_DATA(I).TOTAL.SAT = SUBSTR(C,1,1) CONCATENATE '.' CONCATENATE SUBSTR(C,2,2);

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

THEN ATOTPRC = .5;

2-410

```
      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)
  [this process sets up screen variable with north complex information]

  IF INFORM(I).CONF(CINDEX(I)).NSAT GE 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 .095
      THEN
        C = SUBSTR(F(100.0 * INFORM(I).CONF(CINDEX(I)).NSAT,$THREE),1,3);
        CNFG_DATA(I).NORTH.SAT = SUBSTR(C,1,1) CONCATENATE '.0' CONCATENATE SUBSTR(C,3,1);
      ELSE
        C = SUBSTR(F(100.*INFORM(I).CONF(CINDEX(I)).NSAT,$THREE),1,3);
        CNFG_DATA(I).NORTH.SAT = SUBSTR(C,1,1) CONCATENATE '.' CONCATENATE SUBSTR(C,2,2);
      ELSE CNFG_DATA(I).NORTH.SAT = (4) ' ';
    IF PRCARR(I).CONF(CINDEX(I)).BNPRCNT GE 0.0
      THEN
        CNFG_DATA(I).NORTH.PCT_ARR = SUBSTR(F(100.*PRCARR(I).CONF(CINDEX(I)).BNPRCNT,$THREE),1,3);
        CNFG_DATA(I).NORTH.ARR_DEM = SUBSTR(F(PRCARR(I).CONF(CINDEX(I)).BNARRDEM,$THREE),1,3);
        CNFG_DATA(I).NORTH.ARR_CAP = SUBSTR(F(INFORM(I).CONF(CINDEX(I)).NARRCAP,$THREE),1,3);
        CNFG_DATA(I).NORTH.DEP_DEM = SUBSTR(F(PRCARR(I).CONF(CINDEX(I)).BNDEPDEM,$THREE),1,3);
        CNFG_DATA(I).NORTH.DEP_CAP = SUBSTR(F(PRCARR(I).CONF(CINDEX(I)).BNDEPCAP,$THREE),1,3);
      ELSE
        CNFG_DATA(I).NORTH.PCT_ARR = SUBSTR(F(100.*PRCARR(I).CONF(CINDEX(I)).NPRCNT,$THREE),1,3);
        CNFG_DATA(I).NORTH.ARR_DEM = SUBSTR(F(PRCARR(I).CONF(CINDEX(I)).NARRDEM,$THREE),1,3);
        CNFG_DATA(I).NORTH.ARR_CAP = SUBSTR(F(INFORM(I).CONF(CINDEX(I)).SARRCAP,$THREE),1,3);
        CNFG_DATA(I).NORTH.DEP_DEM = SUBSTR(F(PRCARR(I).CONF(CINDEX(I)).NDEPDEM,$THREE),1,3);
        CNFG_DATA(I).NORTH.DEP_CAP = SUBSTR(F(INFORM(I).CONF(CINDEX(I)).NDEPCAP,$THREE),1,3);
    END 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.THENIF INFORM(I).CONF(CINDEX(I)).SSAT EQ 0.THEN CNFG_DATA(I).SOUTH.SAT = ' 0.0';ELSEIF INFORM(I).CONF(CINDEX(I)).SSAT LT .095THEN

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,3,1);ELSE

C = SUBSTR(F(100.*INFORM(I).CONF(CINDEX(I)).SSAT,\$THREE),1,3);

CNFG_DATA(I).SOUTH.SAT = SUBSTR(C,1,1) CONCATENATE '.' CONCATENATE SUBSTR(C,2,2);ELSE CNFG_DATA(I).SOUTH.SAT = (4) ' ';IF PRCARR(I).CONF(CINDEX(I)).BNPRCNT GE 0.0THEN

CNFG_DATA(I).SOUTH.PCT ARR = SUBSTR(F(100.*PRCARR(I).CONF(CINDEX(I)).BSPRCNT,\$THREE),1,3);

CNFG_DATA(I).SOUTH.ARR.DEM = SUBSTR(F(PRCARR(I).CONF(CINDEX(I)).BSARRDEM,\$THREE),1,3);

CNFG_DATA(I).SOUTH.ARR.CAP = SUBSTR(F(INFORM(I).CONF(CINDEX(I)).SARRCAP,\$THREE),1,3);

CNFG_DATA(I).SOUTH.DEP.DEM = SUBSTR(F(PRCARR(I).CONF(CINDEX(I)).BSDEPDEM,\$THREE),1,3);

CNFG_DATA(I).SOUTH.DEP.CAP = SUBSTR(F(PRCARR(I).CONF(CINDEX(I)).BSDEPCAP,\$THREE),1,3);

ELSE

CNFG_DATA(I).SOUTH.PCT ARR = SUBSTR(F(100.*PRCARR(I).CONF(CINDEX(I)).SPRCNT,\$THREE),1,3);

CNFG_DATA(I).SOUTH.ARR.DEM = SUBSTR(F(PRCARR(I).CONF(CINDEX(I)).SARRDEM,\$THREE),1,3);

CNFG_DATA(I).SOUTH.ARR.CAP = SUBSTR(F(INFORM(I).CONF(CINDEX(I)).SARRCAP,\$THREE),1,3);

CNFG_DATA(I).SOUTH.DEP.DEM = SUBSTR(F(PRCARR(I).CONF(CINDEX(I)).SDEPDEM,\$THREE),1,3);

CNFG_DATA(I).SOUTH.DEP.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 variable with arrival demand balancing information]
  IF PRCARR(I).CONF(CINDEX(I)).BNPRCNT GE 0.
    THENIF INFORM(I).CONF(CINDEX(I)).CHANGENARR GT 0
      THEN
        CHANGE = FLOAT(INFORM(I).CONF(CINDEX(I)).CHANGENARR);
        CNFG_DATA(I).BALANCING.ACOMPLEX = 'SOUTH';
        CNFG_DATA(I).BALANCING.AMOVE = SUBSTR(F(CHANGE,$THREE),1,3);
      ELSEIF INFORM(I).CONF(CINDEX(I)).CHANGENARR LT 0
        THEN
          CHANGE=FLOAT(INFORM(I).CONF(CINDEX(I)).CHANGENARR)
          CNFG_DATA(I).BALANCING.ACOMPLEX = 'NORTH';
          CNFG_DATA(I).BALANCING.AMOVE = SUBSTR(F(CHANGE,$THREE),1,3);
        ELSE
          CNFG_DATA(I).BALANCING.ACOMPLEX = (4)' ';
          CNFG_DATA(I).BALANCING.AMOVE = ' NO';
          TM(58) = FLDDARK;
          TM(59) = FLDDARK;
        ELSE
          TM(57) = FLDDARK;
          TM(58) = FLDDARK;
          TM(59) = FLDDARK;
          DM(44) = FLDDARK;
          DM(45) = FLDDARK;
      END OUTPUT_SET_UP_(BALANCING_ARRIVALS);

```

2-414

```
PROCESS OUTPUT_SET_UP_(BALANCING_DEPARTURES);
  [This process sets up screen variable with departure demand balancing information]

  IF PRCARR(I).CONF(CINDEX(I)).BNPRCNT GE 0.
    THENIF INFORM(I).CONF(CINDEX(I)).CHANGENDEP GT 0
      THEN
        CHANGE = FLOAT(INFORM(I).CONF(CINDEX(I)).CHANGENDEP);
        CNFG_DATA(I).BALANCING.DCOMPLEX = 'SOUTH';
        CNFG_DATA(I).BALANCING.DMOVE = SUBSTR(F(CHANGE,$THREE),1,3);
      ELSEIF INFORM(I).CONF(CINDEX(I)).CHANGENDEP LT 0
        THEN
          CHANGE=FLOAT(INFORM(I).CONF(CINDEX(I)).CHANGENDEP)
          CNFG_DATA(I).BALANCING.DCOMPLEX = 'NORTH';
          CNFG_DATA(I).BALANCING.DMOVE = SUBSTR(F(CHANGE,$THREE),1,3);
        ELSE
          CNFG_DATA(I).BALANCING.DCOMPLEX = (4)' ';
          CNFG_DATA(I).BALANCING.DMOVE = ' NO';
          TH(61) = FLDDARK;
          TH(62) = FLDDARK;
        ELSE
          TH(60) = FLDDARK;
          TH(61) = FLDDARK;
          TH(62) = FLDDARK;

          DM(46) = FLDDARK;
          DM(47) = FLDDARK;
      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 MIDFLAG(I) NE (2) ' '

THEN

LOOP; [R = 1 to 5]

IF (MID(R).NUM AND CNFGRQ(CINDEX(I)).ID) NE 0

THEN DUMMY1 = DUMMY1 CONCATENATE MID(R).CHR;

ENDLOOP;

IF DUMMY1 EQ ''

THEN CNFG_DATA(I).WMSG1 = (80) ' ';

ELSE CNFG_DATA(I).WMSG1 = DUMMY1 CONCATENATE 'COORDINATE WITH MIDWAY TRAFFIC';

HIRLIND = '';

LOOP; [N = 1 to 12]

IF RWYEQP(I).RUNWAY(N).HIRL NE (2) ' '

THEN HIRLIND = HIRLIND CONCATENATE '1'B;

ELSE HIRLIND = HIRLIND CONCATENATE '0'B;

ENDLOOP;

COMBIND = SUBSTR(CNFGRQ(CINDEX(I)).ID,1,12) CONCATENATE SUBSTR(CNFGRQ(CINDEX(I)).ID,13,12);

COMBIND = COMBIND AND HIRLIND;

DUMMY1 = '';

```

LOOP;    [N = 1 to 12]
          IF  SUBSTR(COMBIND,N,1) EQ '1'B
              THEN DUMMY1 = DUMMY1 CONCATENATE HIRL(N);
          ENDLOOP;
          IF  DUMMY1 NE ''
              THEN CNFG_DATA(I).WMSG2 = DUMMY1 CONCATENATE 'INELIGIBLE BETWEEN SUNSET & SUNRISE';
              ELSE CNFG_DATA(I).WMSG2 = (80)' ';
          END  OUTPUT_SET_UP_(OTHERS);

```

ROUTINE 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]

ERR5 = 'INPUT MUST BE X OR BLANK';

CID = (24) '0' B;

CNFG_DATA(I).MSG = 'DATA ENTERED'

B1 = "B;

B2 = '1'B;

B3 = (23)'0'B;

REPEAT WHILE (CNFG_DATA(I).MSG EQ 'DATA ENTERED') [J = 1 to 12]
[check screen inputs]

CURSOR = J + 1;

IF X(CNFG_DATA(I).CONF.ARR(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.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;  
        ENDREPEAT;  
    IF CNFG_DATA(I).MSG EQ 'DATA ENTERED'  
        THEN CURSOR = 2;  
END CCHECK;
```

ROUTINE CVALIDIN (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 LTHEN [ID not in list]

AGAIN = 0

INDEX = 0;

ELSEIF CID GT CNFG_LIST(INDEX).IDTHEN L = INDEX + 1;ELSEIF CID LT CNFG_LIST(INDEX).IDTHEN U = INDEX - 1;ELSE AGAIN = 0 [ID is equal to CNFG_LIST(INDEX)]ENDREPEAT;IF INDEX EQ 0THEN CNFG_DATA(I).MSG = ERRCNFG;ELSE CINDEX(I) = INDEX;END CVALID;

2-420

```
ROUTINE CUPDATE
  IN  CNFGRQ(CINDEX(1)).ID;
  INOUT CNFG_DATA(I);
      [This routine performs local updates on screen]
  LOOP;    [J = 1 to 12]
      IF SUBSTR(CNFGRQ(CINDEX(1)).ID,J,1) = '1'B
      THEN CNFG_DATA(I).CONF.ARR(J) = 'X';
      ELSE CNFG_DATA(I).CONF.ARR(J) = ' ';
      IF SUBSTR(CNFGRQ(CINDEX(1)).ID,J + 12, 1) = '1'B;
      THEN CNFG_DATA(I).CONF.DEP(J) = 'X';
      ELSE CNFG_DATA(I).CONF.DEP(J) = ' ';
  ENDLOOP;
END CUPDATE;
```

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 PARAM_DATA LIKE PARAM

[This structure is similar to PARAM used as a working area within screen routine]

ENDSTRUCTURE;

STRUCTURE CNVRT_PRM LIKE CNVTPRM

[This structure is similar to CNVTPRM 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 PARAM_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 CRSS

PTR ARR [pointer for arrival crosswind threshold data field]

PTR DEP [pointer for departure crosswind threshold data field]

GROUP TAIL

PTR ARR [pointer for arrival tailwind threshold data field]

PTR DEP [pointer for departure tailwind threshold 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;

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 DMS manual, initialized to string of (32) '1's]

ENDSTRUCTURE;

2-424

ROUTINE MENUPRM

INOUT (PARAM, CNVTPRM, RSTATUS);

OUT (TERM);

[This routine invokes menu screen and/or parameter screen]

PARAM_DATA = PARAM;

CNVRT_PRM = CNVTPRM;

I = 1;

REPEAT UNTIL (RSTATUS NE PF12);

IF I EQ 2

THEN PARAM_DATA = PARAM

I = SWITCH(I); [switch between two screens]

REPEAT UNTIL (RSTATUS NE PF11);

I = SWITCH(I);

IF I EQ 1

THEN CALL MSCREEN;

INOUT (RSTATUS);

OUT (TERM);

[This routine controls menu screen]

ELSE CALL PSCREEN;

INOUT (PARAM_DATA, CNVRT_PRM, RSTATUS);

[This routine controls parameter screen]

ENDREPEAT;

ENDREPEAT;

IF SUBSTR(PARAM_DATA.MSG,1, 12) EQ 'DATA ENTERED'

THEN

PARAM = PARAM_DATA;
CHVTFRM = CHVTFRM;

END MENUPRM;

2-425

ROUTINE MSCREEN

INOUT (RSTATUS);

OUT (TERM);
[This routine controls menu screen]

CHR PNAME [character variable of length 8 containing name of DMS 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 PA1

THEN stop;

ENDREPEAT;

END MSCREEN;

ROUTINE PSCREEN

INOUT (PARM_DATA, CNVRT_PRM, RSTATUS);
[This routine controls parameter screen]

CHR PNAME [character variable of length 8 containing name of DMS panel initialized to 'PARMOPT',
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 DMS]

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 PARM_DATA = AUX_DATA;

ELSE

DM = FLDDEF;

DM(5) = FLDHIGH;

CALL PCHECK;

INOUT (PARM_DATA, CNVRT_PRM, 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 PARM_DATA.MSG NE 'DATA ENTERED'

THEN DM(CURSOR) = FLDHIGH;

ELSE

CALL PVALID;

INOUT (PARM_DATA, CNVRT_PRM, 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 PARM_DATA.MSG NE 'DATA ENTERED'

THEN DM(CURSOR) = FLDHIGH;

ELSE

PARM_DATA.MSG = 'DATA ENTERED AT ' ' CONCATEMATE CNT;

AUX_DATA = PARM_DATA;

ENDREPEAT;

END PSCREEN;

PROCESS SET_UP_SCREEN_POINTERS (PARM)

[This process sets up screen pointers for DMS 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.DEP = ADDR(PARM_DATA.PARAMETER.CRSS.DEP);
PARM_LOADLIST.PARAMETER.TAIL.DEP = ADDR(PARM_DATA.PARAMETER.TAIL.DEP);
PARM_LOADLIST.MSG = ADDR(PARM_DATA.MSG);

END SET_UP_SCREEN_POINTERS (PARM)

ROUTINE PCHECKINOUT (PARM_DATA, CNVRT_PFM, 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_PFM.ARR.CRSS);

IF VERIFY('-', PARM_DATA.PARAMETER.ARR.CRSS) EQ 0THEN PARM_DATA.MSG = ERR2;ELSE

CURSOR = 2;

Get STRING (PARM_DATA.PARAMETER.DEP.CRSS) EDIT (CNVRT_PFM.DEP.CRSS);

IF VERIFY('-', PARM_DATA.PARAMETER.DEP.CRSS) EQ 0THEN PARM_DATA.MSG = ERR2;

2-431

```
ELSE
  CURSOR = 3;
  Get STRING (PARM_DATA.PARAMETER.ARR.TAIL) EDIT (CNVRT_PRM.ARR.TAIL);
  IF VERIFY ('-', PARM_DATA.PARAMETER.ARR.TAIL) EQ 0
    THEN PARM_DATA.MSG = ERR2;
    ELSE
      CURSOR = 4;
      Get STRING (PARM_DATA.PARAMETER.DEP.TAIL) EDIT (CNVRT_PRM.DEP.TAIL);
      IF VERIFY('-', PARM_DATA.PARAMETER.DEP.TAIL) EQ 0
        THEN PARM_DATA.MSG = ERR2;
        ELSE CURSOR = 1;
      END PCHECK;
  END PCHECK;
```

ROUTINE PVALID

INOUT (PARM_DATA, CNVRT_PRM, 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;

THRRERR1 = 'CROSSWIND THRESHOLD MUST NOT EXCEED 50 KNOTS';

THRRERR2 = 'TAILWIND THRESHOLD MUST NOT EXCEED 50 KNOTS';

CURSOR = 1;

IF CNVRT_PRM.ARR.CRSS GT 50.

THEN PARM_DATA.MSG = THRRERR1;

ELSEIF CNVRT_PRM.ARR.CRSS EQ 0.

THEN PARM_DATA.PARAMETER.ARR.CRSS = ' 0.0';

ELSE

C = SUBSTR(F(CNVRT_PRM.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_PRM.DEP.CRSS GT 50.

THEN PARM_DATA.MSG = THRRERR1;

ELSEIF CNVRT_PRM.DEP.CRSS EQ 0.

THEN PARM_DATA.PARAMETER.DEP.CRSS = ' 0.0';

ELSE

C = SUBSTR(F(CNVRT_PRM.DEP.CRSS*10.0,\$THREE),1,3);

2-433

```
PARM_DATA.PARAMETER.DEP.CRSS = SUBSTR(C,1,2) CONCATENATE '.' CONCATENATE
SUBSTR(C,3,1);

CURSOR = 3;

IF CNVRT_PFM.ARR.TAIL GT 50.
    THEN PARM_DATA.MSG = THRERR2;
    ELSEIF CNVRT_PFM.ARR.TAIL EQ 0.
        THEN PARM_DATA.PARAMETER.ARR.TAIL = ' 0.0';
        ELSE
            C = SUBSTR(F(CNVRT_PFM.ARR.TAIL*10.0,$THREE),1,3);
            PARM_DATA.PARAMETER.ARR.TAIL = SUBSTR(C,1,2) CONCATENATE
            '.' CONCATENATE SUBSTR(C,3,1);
            CURSOR = 4;
        IF CNVRT_PFM.DEP.TAIL GT 50.
            THEN PARM_DATA.MSG = THRERR2;
            ELSEIF CNVRT_PFM.DEP.TAIL EQ 0.
                THEN PARM_DATA.PARAMETER.DEP.TAIL = ' 0.0';
                ELSE
                    C = SUBSTR(F(CNVRT_PFM.DEP.TAIL*10.0,
                    $THREE),1,3);
                    PARM_DATA.PARAMETER.DEP.TAIL = SUBSTR(C,1,2)
                    CONCATENATE '.' CONCATENATE SUBSTR(C,3,1);
                ELSE
                    ELSE
                        CURSOR = 1;
                    END PVALID;
```

ATE
LMED
8