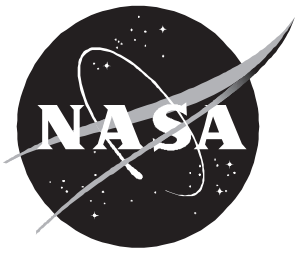
NASA / CR-1998-207657



Aircraft / Air Traffic Management Functional Analysis Model, Version 2.0, Technical Description

Melvin Etheridge, Joana Plugge, and Nusrat Retina
Logistics Management Institute, McLean, Virginia

April 1998

The NASA STI Program Office . . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA Scientific and Technical Information (STI) Program Office plays a key part in helping NASA maintain this important role.

The NASA STI Program Office is operated by Langley Research Center, the lead center for NASA's scientific and technical information. The NASA STI Program Office provides access to the NASA STI Database, the largest collection of aeronautical and space science STI in the world. The Program Office is also NASA's institutional mechanism for disseminating the results of its research and development activities. These results are published by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counter-part or peer-reviewed formal professional papers, but having less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services that help round out the STI Program Office's diverse offerings include creating custom thesauri, building customized databases, organizing and publishing research results . . . even providing videos.

For more information about the NASA STI Program Office, see the following:

- Access the NASA STI Program Home Page at ***http://www.sti.nasa.gov***

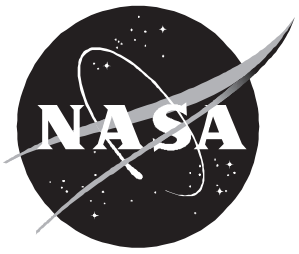
- Email your question via the Internet to help@sti.nasa.gov

- Fax your question to the NASA Access Help Desk at (301) 621-0134

- Phone the NASA Access Help Desk at (301) 621-0390

- Write to:
  NASA Access Help Desk
  NASA Center for AeroSpace Information
  7121 Standard Drive
  Hanover, MD 21076-1320

NASA / CR-1998-207657



Aircraft / Air Traffic Management Functional Analysis Model, Version 2.0, Technical Description

Melvin Etheridge, Joana Plugge, and Nusrat Retina
Logistics Management Institute, McLean, Virginia

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

Prepared for Langley Research Center
under Contract NAS2-14361

April 1998

Available from the following:

NASA Center for AeroSpace Information (CASI)
7121 Standard Drive
Hanover, MD 21076-1320
(301) 621-0390

National Technical Information Service (NTIS)
5285 Port Royal Road
Springfield, VA 22161-2171
(703) 487-4650

Contents

FIGURES

TABLES

Chapter 1
Introduction

The Aircraft/Air Traffic Management Functional Analysis Model, Version 2.0 (FAM 2.0), is a discrete event simulation model designed to support analysis of alternative concepts in air traffic management and control. FAM 2.0 was developed by the Logistics Management Institute (LMI) under task order NS703 of the National Aeronautics and Space Administration (NASA) contract number NAS2-14361. This document provides a technical description of FAM 2.0 and its computer files to enable the modeler and programmer to make enhancements or modifications to the model. Those interested in a guide for using the model in analysis should consult the companion document, *Aircraft/Air Traffic Management Functional Analysis Model, Version 2.0 Users Manual.*

PURPOSE OF THE MODEL

FAM 2.0 is designed to be used by personnel at NASA, the Federal Aviation Administration (FAA), and other organizations and institutions. Those who analyze and decide among competing programs for modernizing air traffic management may find that FAM 2.0 is a useful tool. We intend the model to be usable with little or no instruction by individuals who are unfamiliar with either the model or the host simulation environment. The intended user is the analyst, not the modeler.

FAM 2.0 is designed to provide quantitative time and queuing information about

- personnel work/task loads,
- equipment demand/utilization, and
- communications channel saturation.

This information is for

- aircraft,
- air traffic management and control, and
- airline operations centers.

FAM 2.0 provides users the flexibility to define the simulation scenario to address the particular issue or question under analysis. Baseline simulation scenarios come with the model, representing several different 3-hour periods of all flight operations by the Denver Air Route Traffic Control Center (ARTCC), Denver Terminal

Radar Approach Control (TRACON), and the Denver International and Colorado Springs Municipal Airports. Users can modify the baseline scenario or load an entirely new scenario if desired. Permissible user modifications include

- adding or deleting scenario events,
- changing the model's behavior when an event occurs,
- changing the characteristics of simulation objects (i.e., aircraft and ARTCC sectors), and
- Defining new simulation objects (i.e., aircraft and ARTCC sectors).

These modifications are made to simple text files. Generally, users makes change once to the appropriate file in the baseline scenario and the model applies that change wherever appropriate in the simulation. Similarly, entirely new files in the appropriate format can be loaded at simulation initialization to replace corresponding parts of the baseline.

FAM 2.0 was developed in the MODSIM III simulation environment hosted on an HP-UNIX platform. Since MODSIM III generates an executable (.exe) file, FAM 2.0 can run on any HP-UNIX platform. It is available from LMI, McLean, Virginia.

Chapter 2
Description of Simulation Events

FAM 2.0 is a discrete event simulation model centered around the events associated with a given simulation scenario. Currently, the model replicates the operations of Denver ARTCC, Denver TRACON, and the Denver and Colorado Springs Airports. Users have the option of modifying some part(s) of the baseline simulation and/or entering an entirely new scenario.

The simulation has two types of events:

- *A priori events*, events that are known in advance for each flight, and
- *random events,* events that occur randomly during a flight.

An example of an a priori event would be a handoff of an aircraft from one controller to another. Random events include both routine and unusual or emergency events that occur randomly, such as a request for a change of flight level.

Each of these primary events, both a priori and random, has a fixed set of associated sub-events. Continuing the previous example, a handoff from one controller to another might be broken down into the following associated sub-events:

- Request from losing to gaining controller to take control
- Acceptance of control from gaining controller to losing controller
- Instructions from losing controller to aircraft to contact gaining controller
- Aircraft “rogers” acknowledgment.

A request for change of flight level might have these associated sub-events:

- Aircraft contacts controller
- Controller “rogers”
- Aircraft requests new flight level
- Controller clears aircraft to climb/descend to new flight level or denies request
- Aircraft acknowledges.

There are, then, two levels of events: (1) the primary events and (2) for each primary event, a set of associated sub-events. To differentiate between the two, hereinafter **we note a primary event with an upper case 'e' ("Event") and an associated sub-event with a lower case 'e' ("event").**

During the simulation run, whenever a FAM 2.0 primary Event occurs, the model executes the set of associated events. Each of the associated events carries with it personnel task loadings, equipment requirements, and communications channel demands all in units of time.

There can be more than one set of associated events for each a priori Event type. The associated event sets could vary according to the equipment installed on the aircraft or available to the controller. An example could be the use of data link to provide certain communications. The situation could exist where some aircraft had a data link and others did not. Communications with controllers would primarily use the data link, if installed. The model would use different sets of associated events in the simulation for aircraft with and without data link.

There are two sources of primary Events. The a priori Events are contained in a text file, which is read by the model at the start of the simulation. Random events are generated by a random event generator inside the model. During the simulation, when an Event occurs, whether from the a priori Event file or originated by the random event generator, the model then executes the appropriate sets of associated events.

With this approach, users only need to change a particular set of associated events once before running the model in order to have the change occur throughout the simulation. If, for example, controller handoffs of aircraft were done automatically via a data link, reducing pilot and controller task loading associated with the handoff, an analyst would make the appropriate changes in the event task loads and (possibly) eliminate the "aircraft changes communications frequency" event.

If desired, users can add or eliminate some a priori Events entirely. In the case of adding Events, users must copy a text file of associated events into the appropriate directory. Details are in Chapter 3.

Two sets of a priori Events reside in the baseline scenarios. One contains the Events associated with flights as they actually occurred. The flights used the current point-to-point system of air navigation based on ground-based navigational radios. The flights were conducted under positive FAA control by ground-based controllers using conventional voice communications radios. The other set of events contains the wind-corrected great circle flight (so-called "free flight") tracks for the same flights. This enables the user to compare current navigation procedures and free-flight procedures.

In addition to modifying the events associated with one or more primary Events, and/or changing the primary Event file, users can, if desired, also read in an entirely new primary Event text file. This would be appropriate if a user wishes to analyze a different set of flights.

Chapter 3
Technical Design

This chapter contains the technical design for the Functional Analysis Model (FAM 2.0). Sections detail each object and included elements, methods, and files. The environment, MODSIM III, is an asynchronous, object-oriented, simulation language on UNIX and other platforms. MODSIM uses the C++ programming language to generate an executable file, making the model extremely portable.

This chapter contains computer commands and code listings of file contents. For clarity, we use the following convention:

- Computer commands are in this font. Embedded file names, paths, and flags set by the user are in italics and enclosed in < >.
- Code listings of file contents are in this font. Again, embedded file names, *paths, and flags set by the user within a file listing are in italics and enclosed in* < >.

You should substitute the appropriate entry for your simulation when you encounter italicized text enclosed in <.> in a program listing.

The following are the two main methods in MODSIM:

- **ASK Methods.** These are methods or functions that are members of objects and are executed synchronously, i.e., in the same way as any function or subroutine is executed in C/C++
- **TELL Methods.** These are methods that are scheduled for execution at a future time. Control comes back immediately to the calling method before the TELL Method is actually executed.

The TELL Methods allow for asynchronous execution of events. As an event is generated, it is placed in an execution queue for execution at its scheduled execution time, even if there are previously generated events scheduled for execution at a later time. In other words, MODSIM executes events in the order of their execution time, not in the order in which the events are generated. The user (or programmer) simply sets up the event queue and, upon the internal StartSimulation() command, the MODSIM environment and program takes over the execution of all events as necessary. MODSIM also has the advantage of being able to generate and remove objects from the simulation during the simulation run. This capability enables FAM 2.0 to only have aircraft objects in the simulation for those flights that are actually active at that point in the simulation.

PROGRAM DESCRIPTION

FAM 2.0 executes from the UNIX command line and at execution requires two mandatory input files:

- Scenario file
- Output file name.

For more detail about these input files, see the companion document, *Aircraft/Air Traffic Management Functional Analysis Model, Version 2.0 Users Manual*.

PROGRAM MODULES

The FAM 2.0 program is composed of a number of program modules. All MODSIM programs run from a main module. The FAM 2.0 main module code listing is in Figure 3-1.

Figure 3-1. FAM 2.0 Main Module Code Listing

```
//MAIN MODULE fam

BEGIN
{scenario file and output file are user input}
processCommandLine (scenarioFile, outputFile);
readScenarioFile (scenarioFile);
processScenarioFile;

IF there are no ERRORS
NEW (outStream);
ASK outStream TO Open (outputFile, Output);

{finalReport object is responsible for printing the final report}
NEW (finalReport);

{simulationEnd is a user input}
TELL finalReport TO print IN simulationEnd;

StartSimulation;
END IF;
END MODULE.
```

Table 3-1 contains the descriptions of the other FAM 2.0 program modules.

Table 3-1. Program Module Descriptions

Module	Description
DaircraftMod.mod	Definition of aircraftObj and related methods
DairportContMod.mod	Definition of airportcontrollerObj and related methods
DAOCMod.mod	Definition of AOCObj[1] and related methods
DassocevtMod.mod	Definition of assocevtObj and related methods
DchannelMod.mod	Definition of channelObj and related methods
DerrorMod.mod	Definition of famerrorObj and related methods
DfamStreamMod.mod	Definition of famstreamObj and related methods
DfinalReportMod.mod	Definition of finalReportObj and related methods
DglobalMod.mod	Definition of various global objects and variables
DloadMod.mod	Definition of loadObj and modeTypeloadObj and related methods
DoutputStatMod.mod	Definition of outputStatObj and related methods
DprimaryEvtMod.mod	Definition of primaryEvtObj and related methods
DprocedureMod.mod	Definition of procedures
DsectorMod.mod	Definition of sectorObj and related methods
DsimMod.mod	Definition of simObj and related methods
DstatMod.mod	Definition of iStatObj and rStatObj and related methods
DTRACONContMod.mod	Definition of TRACONControllerObj and related methods
DtriggerMod.mod	Definition of trigEvtObj and related methods
IaircraftMod.mod	Implementation of aircraftObj and related methods
IairportContMod.mod	Implementation of airportcontrollerObj and related methods
IAOCMod.mod	Implementation of AOCObj and related methods
IassocevtMod.mod	Implementation of assocevtObj and related methods
IchannelMod.mod	Implementation of channelObj and related methods
IerrorMod.mod	Implementation of famerrorObj and related methods
IfamStreamMod.mod	Implementation of famstreamObj and related methods
IfinalReportMod.mod	Implementation of finalReportObj and related methods
IglobalMod.mod	Implementation of various global objects and variables
IloadMod.mod	Implementation of loadObj and modeTypeloadObj and related methods
IoutputStatMod.mod	Implementation of outputStatObj and related methods
IprimaryEvtMod.mod	Implementation of primaryEvtObj and related methods
IprocedureMod.mod	Implementation of procedures
IsectorMod.mod	Implementation of sectorObj and related methods
IsimMod.mod	Implementation of simObj and related methods
IstatMod.mod	Implementation of iStatObj and rStatObj and related methods
ITRACONContMod.mod	Implementation of TRACONControllerObj and related methods
ItriggerMod.mod	Implementation of trigEvtObj and related methods
Mairport.mod	Implementation of main module

Note: Airline Operations Center (AOC).

GLOBAL LIST AND OVERALL MODEL DIAGRAM

Figure 3-2 shows the hierarchy of the significant program global lists for FAM 2.0. Table 3-2, which follows, contains a description of these lists.

Figure 3-2. Program Global List Hierarchy

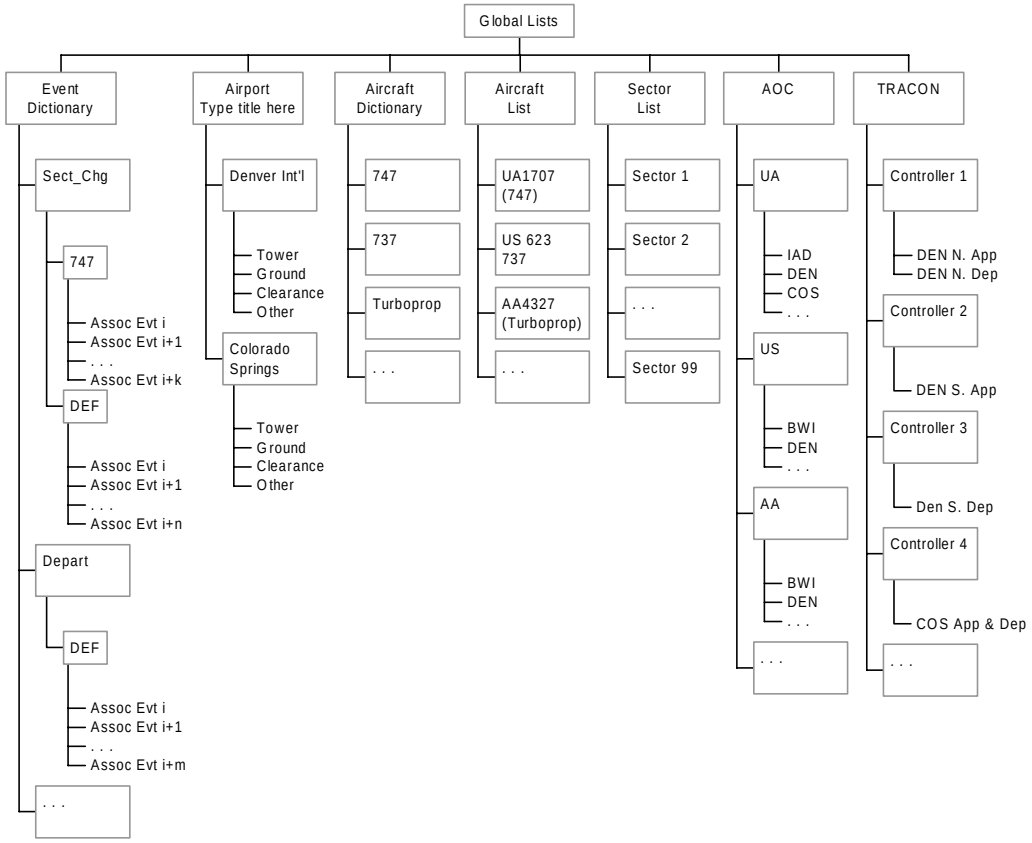


Table 3-2. Global List Descriptions

Global list	Description
EventDictionary	List of primary Events and their associated event list(s)
aircraftDictionary	List of aircraftObj; one for each type of aircraft (e.g., 747) with all of its loads
aircraftList	List of aircraftObj; one for each instance of aircraft (e.g., UA1707)
aircraftTypeList	List containing the names of each unique aircraft type
sectorDictionary	List of sectorObj; one for each type of sector (e.g., sector_A) with all of its loads
sectorList	List of sectorObj; one for each instance of sector (e.g., sector 54)
sectorTypeList	List containing the names of each unique sector type
AOCDictionary	List of AOCObj; one for each type of AOC (e.g., UA) with all of its loads

Table 3-2. Global List Descriptions (Continued)

Global list	Description
AOCList	List of AOCObj; one for each instance of sector (e.g., UABWI)
AOCTypeList	List containing the names of each unique AOC type
aptContDictionary	List of airportControllerObj; one for each type of airport controller (e.g., all_colorado) with all of its loads
aptContList	List of airportControllerObj; one for each instance of sector (e.g., colorado_tower)
aptContTypeList	List containing the names of each unique airport controller type
TRACONContDictionary	List of TRACONControllerObj; one for each type of TRACON controller (e.g., denver_area) with all of its loads
TRACONContList	List of TRACONControllerObj; one for each instance of sector (e.g., denver_approach)
TRACONContTypeList	List containing the names of each unique TRACON controller type
TRACONAptPosList	List of airports and positions for each TRACON controller
channelList	List of channels used for all simulation objects
activationList	List containing the activations times of all aircraft
eventTimeList	List containing the times of all events in the a priori event file
AOCNameList	List of AOC names
airportNameList	List of airport names
TRACONNameList	List of TRACON names
usedAssocEvtList	List to store old associated events for later cleanup
usedTriggerList	List to store old primary Events for later cleanup

MODEL PROCEDURES

FAM 2.0 has several procedures that read and validate user input files. Procedures are different from ASK and TELL Methods because they are not affiliated with any objects. Procedures are traditional subroutines that are called directly from the main module to process various user inputs.

The following list explains some key procedures for FAM 2.0. All procedures are contained in *IprocedureMod.mod* and the corresponding definitions are contained in *DprocedureMod.mod*. A partial list of the FAM 2.0 procedures follows. These procedures are key in creating and searching lists.

Procedure name: *checkAOCName*
Input: name
Output: TRUE/FALSE
Description: Check Airline Operations Center (AOC) name to ensure it exists in *AOClist*

- Procedure name: checkAirportName
  Input: name
  Output: TRUE/FALSE
  Description: Check airport name to ensure it exists in *airportlist*

- Procedure name: checkTraconName
  Input: name
  Output: TRUE/FALSE
  Description: Check TRACON name to ensure it exists in *TRACONlist*

- Procedure name: *lookupAircraftDictionary*
  Input: aircraft type
  Output: aircraftObj
  Description: Match the aircraft type (e.g., 747) and retrieve the corresponding aircraftObj from aircraftDictionary list containing all its loads

- Procedure name: *lookupAircraftList*
  Input: airline, flight number
  Output: aircraftObj
  Description: Lookup airline and flight number and retrieve the aircraftObj with matching airline and flight number (e.g., UA1707)

- Procedure name: *lookupSectorDictionary*
  Input: sectorType
  Output: sectorObj
  Description: Match the sector type (e.g., sectorA) and retrieve the corresponding sectorObj from sectorDictionary list containing all its loads

- Procedure name: *lookupSectorList*
  Input: sectorID
  Output: sectorObj
  Description: Lookup sector ID and retrieve the sectorObj with matching sector identifier (e.g., sector54)

- Procedure name: *readEventDicFile*
  Input: event dictionary file name
  Output: (none)
  Description: Reads the event dictionary as specified in the file name (e.g., *event.dic*)

- Procedure name: *allocateGlobalLists*
  Input: (none)
  Output: (none)
  Description: Allocates all global lists for the model

- Procedure name: *deallocateGlobalLists*
  Input: (none)
  Output: (none)
  Description: Deallocates all global lists for the model

- Procedure name: *initializeGlobalLists*
  Input: (none)
  Output: (none)
  Description: Initialize each global list with appropriate objects

- Procedure name: *printGlobalLists*
  Input: (none)
  Output: (none)
  Description: Prints information about members of each global list

- Procedure name: *readTrigEvtFile*
  Input: a priori event file name (e.g., *trig.evt*)
  Output: (none)
  Description: Reads event file line by line. For each vector or row in the event file:

- Create a trigevtObj and call it trigger

  - Schedule it for execution, (e.g., TELL trigger TO processTrigger IN *trigger.time*)

- Procedure name: *addChannelToChannelList*
  Input: channelPos, channelValue
  Output: (none)
  Description: Adds channel to global channel list

- Procedure name: *getPrimaryEvtPtr*
  Input: eventName
  Output: primaryevtObj
  Description: Searches the event dictionary for an exact match for primary Event and returns a pointer to the event

- Procedure name: *doesPrimaryExist*
  Input: event name
  Output: TRUE/FALSE
  Description: Returns whether the primary Event exists or not in the event dictionary

- Procedure name: *lookupAssociatedEvtList*
  Input: primaryEvtName, ac1Type, ac2Type, sector1Type, sector2Type
  Output: QueueObj
  Description: Given a primary event name, aircraft 1 type, aircraft 2 type, sector 1 type and sector 2 type, lookup the corresponding associated event list

- Procedure name: *readTypeFile*
  Input: file name, type
  Output: type file name
  Description: For each type specified in the file, create a typeObj in the corresponding global list (e.g., *aircraftTypeList*, *sectorTypeList*, etc.)

- Procedure name: *buildLoad*
  Input: *loadList*, mode, Type, Load, pEvtName, aEvtName, modeName, typeName, maxIndex
  Output: (none)
  Description: When reading the load file for a particular object (e.g., *747.ac*), build the load for various personnel/radio/equipment for this activity [specified by primary event (*pEvtName*), associated event (*aEvtName*)] for each mode (*modeName*) and mate-type (*typeName*). Add the loads to the input load list (*loadList*).

- Procedure name: *doesTypeExist*
  Input: typeName, simObjType
  Output: TRUE/FALSE
  Description: Given a type name (e.g., sectorA) and simObj type (e.g., sector), check whether this type exists in sectorType list

- Procedure name: *lookupSectorType*
  Input: sector ID
  Output: sector type
  Description: Given a sector ID (e.g., sector54), lookup its type (SectorC)

- Procedure name: *lookupAircraftType*
  Input: airline, flight number
  Output: aircraft type
  Description: Given airline and flight number (e.g., UA1707), lookup its type (777)

- Procedure name: *readSectorDicFile*
  Input: file
  Output: (none)
  Description: Read the sector dictionary file (e.g., *sector.dic*) and set up the *sectorList* (global list)

- Procedure name: *lookupAOCDictionary*
  Input: AOCType
  Output: AOCObj
  Description: Match the AOC type (e.g., UA) and retrieve the corresponding AOCObj from AOCDictionary list containing all its loads

- Procedure name: *lookupAOCList*
  Input: AOC name
  Output: AOCObj
  Description: Lookup AOC name and retrieve the AOCObj with matching AOC identifier (e.g., UABWI)

- Procedure name: *lookupAOCType*
  Input: AOC name
  Output: AOC type
  Description: Given AOC name (UABWI), lookup its type (UA)

- Procedure name: *readAOCDicFile*
  Input: AOC dictionary files
  Output: (none)
  Description: Read the AOC dictionary file (e.g., *AOC.dic*) and set up the AOC list (global list)

- Procedure name: *lookupAptContDictionary*
  Input: airport controller type
  Output: airportControllerObj
  Description: Given an airport controller type (e.g., tower or all_colorado), return the airportControllerObj block from the airport dictionary containing all its loads

- Procedure name: *lookupAptContList*
  Input: airport name, controller name
  Output: airportControllerObj
  Description: Given an airport name (e.g., colorado) and a controller name (e.g., ground), return the airportControllerObj block from the airport list

- Procedure name: *lookupAptContType*
  Input: airport name, controller name
  Output: type name
  Description: Given an airport name (e.g., colorado) and a controller name (e.g., ground), return its type (e.g., all_colorado)

- Procedure name: *readAirportDicFile*
  Input: airport dictionary file
  Output: (none)

Description: Read the airport dictionary file (e.g., *airport.dic*) and set up the airport list (global list)

- Procedure name: *lookupTraconController*
  Input: TRACON name, airport name, position name
  Output: TRACONControllerObj
  Description: Given a TRACON name (e.g., denver_TRACON), an airport name (colorado), a position (e.g., final), return the corresponding TRACONControllerObj block (e.g., controller_THREE)

- Procedure name: *readTraconDicFile (file)*
  Input: TRACON dictionary file name
  Output: (none)
  Description: Read the TRACON dictionary file (e.g., *TRACON.dic*) and set up the TRACON controller list (global list)

- Procedure name: *lookupTraconContDictionary*
  Input: type name
  Output: TRACONControllerObj
  Description: Given a TRACON controller type (e.g., denver_area), return the TRACONControllerObj block from the TRACON dictionary containing its loads

- Procedure name: *lookupTraconContList*
  Input: TRACON name, controller name
  Output: TRACONControllerObj
  Description: Given a TRACON name (e.g., denver_TRACON) and a controller name (e.g., controller_THREE), return the TRACONControllerObj block from the TRACON list (global list)

- Procedure name: *readRandomEvtFile*
  Input: random event file name
  Output: (none)
  Description: Read and process the list of random events (e.g., file *rand.evt*)

- Procedure name: *readScenarioFile*
  Input: scenario file
  Output: (none)
  Description: Read the scenario file (*scenario.sc*)

- Procedure name: *processScenarioFile*
  Input: (none)
  Output: (none)
  Description: Process and trap errors in the scenario file

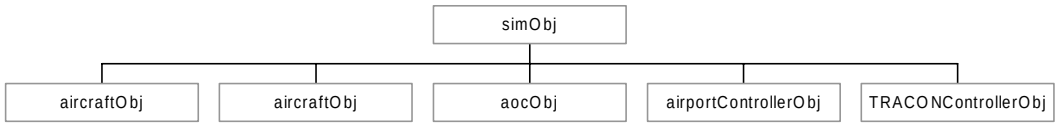
- Procedure name: *processCommandLine*
  Input: scenario file, output file
  Output: (none)
  Description: Parse the command line for FAM 2.0 execution at the UNIX prompt

- Procedure name: *getDeactivationTime*
  Input: airline, flight number
  Output: (none)
  Description: Get the deactivation time for a particular flight, given its airline and flight number, from the aircraft list

SIMULATION OBJECTS

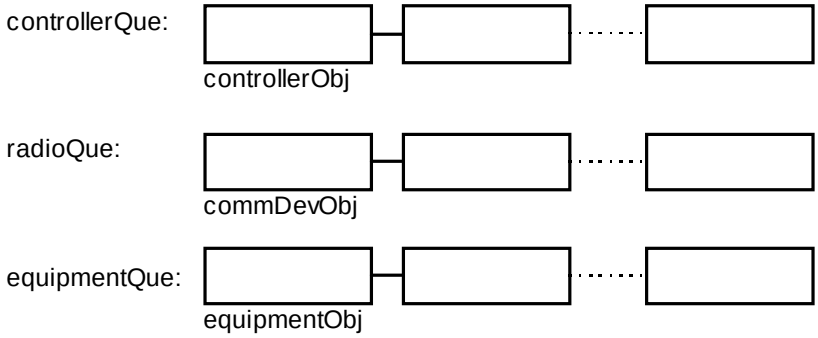
A simulation object represents a person (pilot, controller, dispatcher), communications device (radio), or equipment system (radar, computer, etc.). A simulation object can also be a collection of other simulation objects, such as an aircraft, which contains simulation objects representing pilots, radios, and equipment. Simulation objects perform activities and/or collect simulation statistical data. The loads for performing the events for each personnel, radio, and equipment are contained in the load file. Examples of simulation objects are aircraft, sector, AOC, airport controller and TRACON controller. Figure 3-3 diagrams the various kinds of simulation objects.

Figure 3-3. Simulation Object Class



A simulation object contains sets (queues) of controllers, radios, and equipment depicted in Figure 3-4.

Figure 3-4. Simulation Object Contents



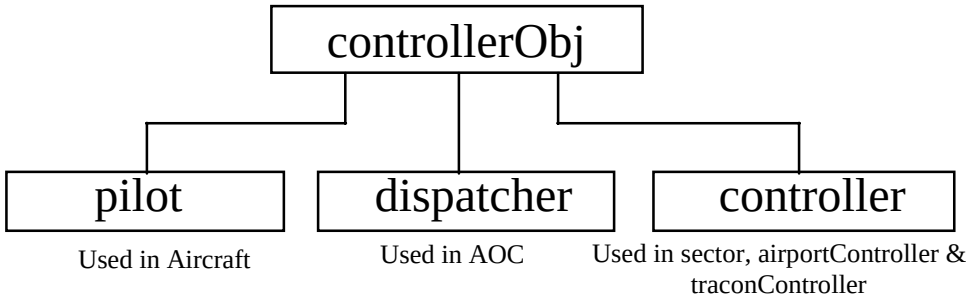
simObjType	(e.g., AIRCRAFT, SECTOR, AOC, AIRPORT, TRACON)
typeName	(e.g., 747, SECTOR_A, UA, TOWER, DENVER_TRA)
loadList	Contains loads for a particular object (e.g., aircraftObj,

. . .

Controller Object (*controllerObj*)

There are three types of controller objects, shown in Figure 3-5.

Figure 3-5. Types of Controller Objects



System Object (*sysObj*)

System objects can be either communications device objects (*commDevObj*) or equipment system objects (*equipmentObj*), as shown in Figure 3-6.

Figure 3-6. Types of System Objects

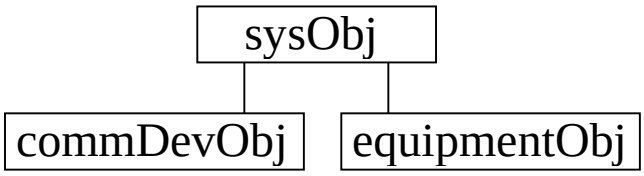
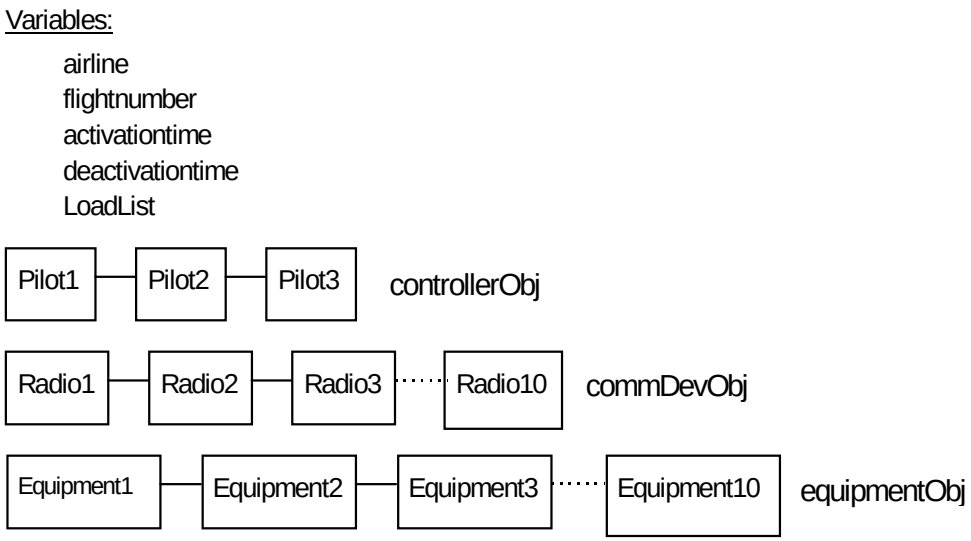


Table 3-10 on page 3-34 contains details on statistical counters for the *controllerObj* and *sysObj*.

Aircraft Object (*aircraftObj*)

The aircraft object (*aircraftObj*) is a type of simulation object (*simObj*). In addition to the data fields and methods inherited from *simObj*, *aircraftObj* has airline, flight number, activation time, and deactivation time. Figure 3-7 is a diagram of an aircraft object.

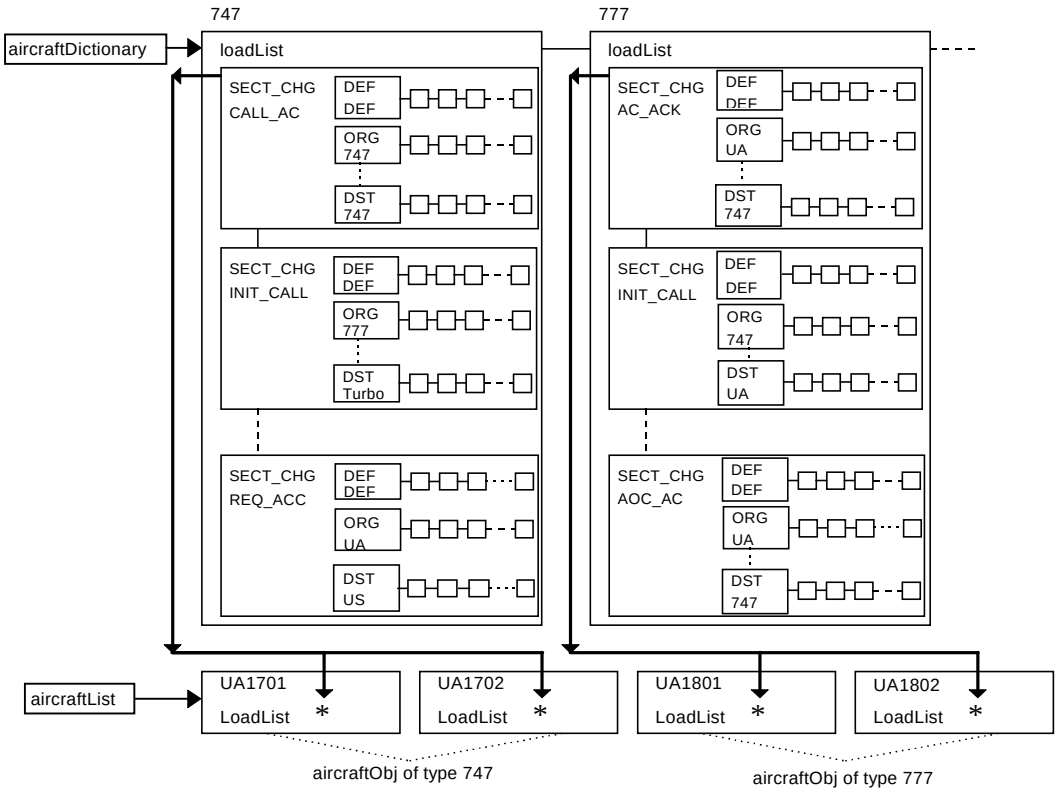
Figure 3-7. Aircraft Object Diagram



The aircraft dictionary (*aircraftDictionary*) is a list holding different types of aircraft. For example, if there are only two types of aircraft, 747 and 777, then this list would point to two aircraft objects. Each of these two aircraft objects will contain all the loads for various combinations of primary Event, associated event, mode, and mate-type.

The aircraft list (*aircraftList*) is a list holding all the instances of aircraft in the model. For example, if the model has four aircraft (UA1701, UA1702, UA1801, and UA1802), the list would point to four aircraft. The records in the *aircraftList* do not contain the actual loads; the *loadList* points to the corresponding aircraft type object in the *aircraftDictionary*. Figure 3-8 diagrams this relationship.

Figure 3-8. Relationship of Aircraft Files



Similarly, the sector, AOC, airport controller, and TRACON controller objects are defined this way. Each simulation object has its own dictionary and list:

- *sectorObj* has *sectorDictionary* and *sectorList*
- *aocObj* has *aocDictionary* and *aocList*
- *airportControllerObj* has *aptContDictionary* and *aptContList*
- *traconControllerObj* has *traconContDictionary* and *traconContList.*

Load Object (*loadObj*)

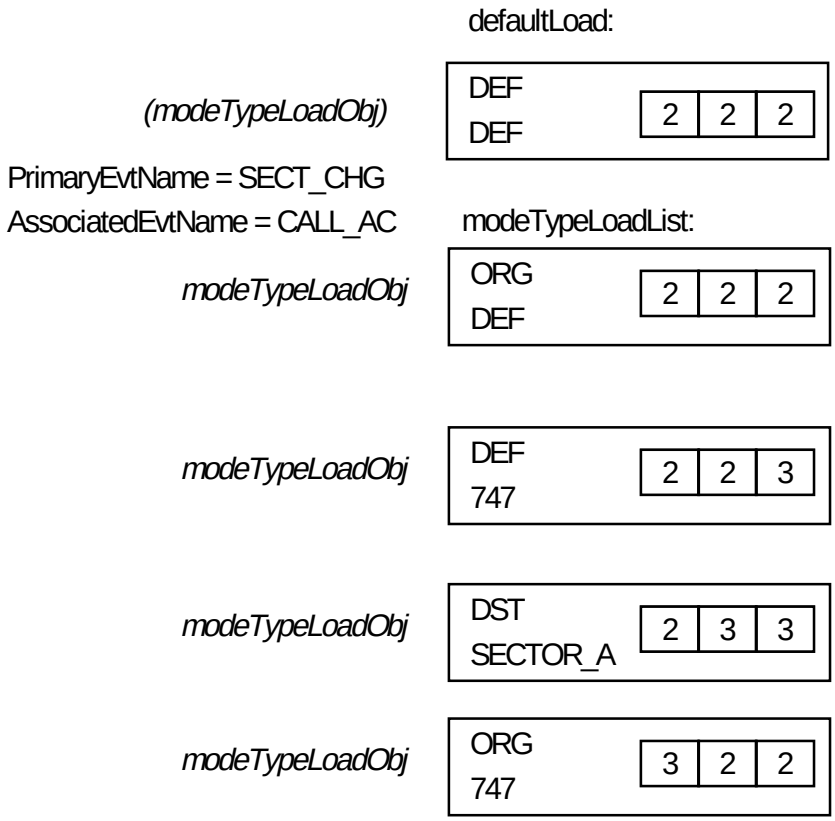
Task, communications, and equipment loads are implemented via a complex data structure. The data structure involves building and searching multiple linked lists. The complexity is due to the requirement of varying loads by mode (i.e., whether the object is the originating object [ORG] or destination object [DST] of the event) and mate-type (object type of mate) for a particular event.

For example:

```
//C1 = controller1; R1 = Radio1; EQ1 = Equipment1
//PRIMARY_EVT ASSOC_EVT  MODE    MATE-TYPE  C1 R1 EQ1
SECT_CHG      CALL_AC    ORG     747        2  2  2
SECT_CHG      CALL_AC    DST     SECTOR_A3  3  3
SECT_CHG      CALL_AC    DEF     747        2  2  3
SECT_CHG      CALL_AC    ORG     DEF        2  2  2
SECT_CHG      CALL_AC    DEF     DEF        2  2  2
```

The above load will be implemented via the data structure shown in Figure 3-9.

Figure 3-9. Load Object Data Structure



LOOKUP LOAD METHOD

The lookup load method (*lookupLoad* in *IsimMod.mod*) builds and searches the *loadObj* data structure. It requires five parameters in order to retrieve a load value for a particular activity. The parameters are primary Event name, associated event name, mode, mate-type, and an index. The first two are self-explanatory. The meanings of the other parameters are:

- Mode. If the simulation object participates in an activity as the
  - origin, then the mode is ORG;
  - destination, then the mode is DST; and
  - DEF may be used to specify either mode.
- Mate-type
  - If mode is ORG, the mate is DST, and the mate-type is its object type (e.g., 747, SECTOR_A, UA).
  - If mode is DST, the mate is ORG, and the mate-type is its object type (e.g., 747, SECTOR_A, UA).
- index. An integer value that references a particular controller/radio/equipment within the simulation object

The pseudocode for the *lookupLoad* method is shown in Figure 3-10.

Figure 3-10. LookupLoad Pseudocode

```
FOR each load in the load list
   Match the primary Event and associated event name
      Find the exact mode and mate-type
      If not found, find DEF for mode and exact mate-type
      If not found, find exact mode and DEF for mate-type
      If not found, find DEF for mode and DEF for mate-type
   If found, retrieve load value
   If not found, THEN it is an error
```

BUILD LOAD

The *modeTypeLoadObj* contains the mode, mate-type, and a list of load values. The build load procedure (*buildLoad* in *IprocedureMod.mod*) requires seven parameters in order to build a list of *modeTypeLoadObj* to hold the load values of a particular activity. See Figure 3-9. The parameters are

- the *loadList* from the simulation object,
- the *modeTypeLoad* object,
- the primary Event name,
- the associated event name,
- the mode,

- the mate-type, and
- the maximum number of loads for the activity.

Once the *modeTypeLoad* list is built, a load value along with its index can be inserted via method *setLoad* of the *modeTypeLoad* object.

The pseudocode for the *buildLoad* procedure is shown in Figure 3-11.

Figure 3-11. Pseudocode for buildLoad Procedure

```
If primary Event and associated event name already exist in
    the loadList
   If mode and mate-type are DEF
      Create a default modeTypeLoadObj for the loadObj
   ELSE
   Add a new modeTypeLoadObj to the loadObj
ELSE IF primary Event and associated event name do not exist
    in the loadList
   Create a new loadObj
   If mode and mate-type are DEF
      Create a default modeTypeLoadObj for the loadObj
   ELSE
      Add a new modeTypeLoadObj to the loadObj
```

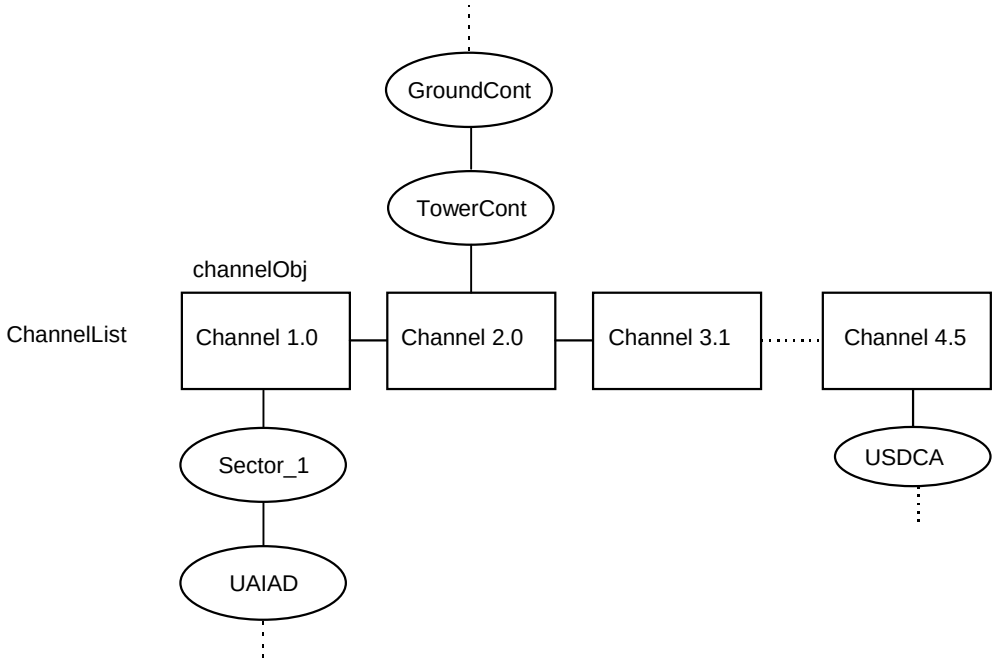
CHANNEL OBJECTS

Similar to simulation objects, channel objects gather statistics of communication (e.g., the communication time between a pilot and personnel of the tower controller) between simulation objects via communications devices such as radios. The channel object must be acquired as a resource object.

ChannelList

Figure 3-12 is an example of activities of simulation objects request channels at various times during simulation. In this example, Channel 1.0 is used by Sector 1 and UAIAD (an AOC). If Channel 1.0 is in use by one of these objects, it cannot be simultaneously used by the other. In this event, a communication involving the other object is queued until Channel 1.0 is free.

Figure 3-12. Communications Example



Lock Channel

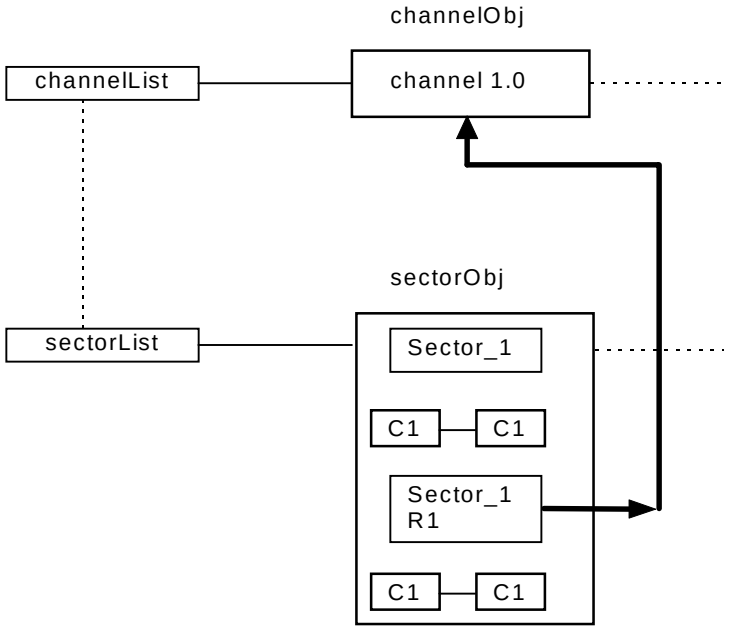
The lock channel method (*lockChannel* in module *IsimMod.mod*) is called from method *commenceEvt.* Once a channel is requested by an activity, it will be locked until it completely serves that activity. While a channel is locked, requests from other simulation objects for that channel will be queued. The pseudocode for method *lockChannel* is in Figure 3-13.

Figure 3-13. Method lockChannel Pseudocode

```
Find the first radio in the commQue that has load value
    greater than zero
Retrieve the channel value from the commDevObj
Find the channel object associated with this channel value
Lock the channel
```

Figure 3-14 shows the relationship between *simObj* and *channelObj*.

Figure 3-14. Relationship Between simObj and channelObj



Unlock Channel

The unlock channel method (*unlockChannel* in module *IsimMod.mod*) is called from method *commenceEvt*. When a channel finishes serving an activity, it will be released to serve other activities. The pseudocode is shown in Figure 3-15.

Figure 3-15. Method unlockChannel Pseudocode

```
Unlock the channel
Update statistics for the channel
```

EVENT PROCESSING

FAM 2.0 processes events dynamically during the simulation run. It reads the primary Events from the a priori event file (named *trig.evt*), which contains a set of primary Event vectors (records). Before simulation start, FAM 2.0 retrieves the associated event list for each primary Event. Table 3-3 contains a short sample *trig.evt* file with three primary Event vectors.

Table 3-3. Sample trig.evt File

EVENT	1 TIME	2 AL	3 FN	4 AC_T	5 ALT_AL	6 ALT_FN	7 SCT1	8 SCT2	9 ARPT	10 TRC	11 AOC	12 ACCHNL
ACTIVATE_AC	1000	UA	1707	747	NULL	0	1	0	NULL	NULL	NULL	0.0
SECT_CHG	1000	UA	1707	NULL	NULL	0	1	2	DEN	DEN	NULL	0.0
DEACTIVATE_AC	4000	UA	1707	NULL	NULL	0	2	0	NULL	NULL	NULL	0.0

Notes:

1. TIME=Simulation time of primary Event initiation.
2. AL=Airline of primary (#1) aircraft.
3. FN=Flight number of primary (#1) aircraft.
4. AC_T=Type of primary (#1) aircraft.
5. ALT_AL=Airline of secondary (#2) aircraft.
6. ALT_FN=Flight number of secondary (#2) aircraft.
7. SCT1=Identification number of primary (losing) sector.
8. SCT2=Identification number of secondary (gaining) sector.
9. ARPT=Name of airport.
10. TRC=Name of TRACON.
11. AOC=Name of AOC.
12. ACCHNL=Communications channel for aircraft-to-aircraft communications.

EVENT DICTIONARY

The event dictionary contains the file names of the sets of associated events for each primary Event (a priori or random). There can be more than one associated event list for each primary Event, since the associated events can vary with the types of aircraft and sectors participating in the primary Event. The dictionary also introduces priority of the event. The priority determines the rank of service priority when events are queued up for processing at various simulation servers or object, like sectors, AOCs, and airport and TRACON controllers.

Table 3-4 contains a sample event dictionary showing the file format.

In the event dictionary, *NULL* indicates that the field (column) does not apply to that Event. *DEF* (default) indicates that the associated event file for that record (row) should be used in all cases unless the type of the objects involved in the actual event are listed in another record. For example, referring to Table 3-4, there are five sector change (*SECT_CHG*) event records, the first five rows of the table. The first record applies to 747 aircraft where both sectors are type *SECTOR_A.* Similarly, the second record applies to 747 aircraft where the Sector 1 type is *SECTOR_A* and the Sector 2 type is *SECTOR_B.* The third and fourth records apply when the aircraft is a 777 and the sectors are the types shown. The fifth record, with DEF, applies to all cases not covered by the first four records.

Table 3-4. Sample Event Dictionary File

EVENT	1 AC1TYP	2 AC2TYP	3 SCT1TYP	4 SCT2TYP	5 ASCEVT_FILE	6 PRIORITY
SECT_CHG	747	NULL	SECTOR_A	SECTOR_A	sector_chg747.evt	5.0
SECT_CHG	747	NULL	SECTOR_A	SECTOR_B	sector_chg747.evt	5.0
SECT_CHG	777	NULL	SECTOR_B	SECTOR_B	sector_chg777.evt	5.0
SECT CHG	777	NULL	SECTOR_B	SECTOR_A	sector_chg777.evt	5.0
SECT_CHG	DEF	NULL	DEF	DEF	sector_chgdef.evt	5.0
DEPART	DEF	NULL	NULL	NULL	departure.evt	5.0
APPROACH	DEF	NULL	NULL	NULL	approach.evt	5.0
ALT_CHG	DEF	NULL	NULL	NULL	altitude_change.evt	10.0
CONFLICT	DEF	NULL	NULL	NULL	conflict.evt	10.0
CATCH_FIRE	DEF	NULL	NULL	NULL	catch_fire.evt	15.0

Notes:
1. Type of primary aircraft (1).
2. Type of second aircraft (2).
3. Type of primary or losing sector (1).
4. Type of secondary or gaining sector (2).
5. Associated event file name.
6. Event priority.

In operation, FAM 2.0 first looks for a record with a match in the appropriate type fields. If it finds one, it uses the associated event in that record. If no match is found, it will look for DEF and use that associated event file. If no default row is found, then the FAM 2.0 will generate an error and stops. For example, if the aircraft type 747 is the primary aircraft participating in an event, FAM 2.0 will use the associated event if it finds a record with 747 in AC1TYP. If FAM 2.0 finds no exact match between 747 and AC1TYP, then it will search for DEF under AC1TYP. If it finds no such record FAM 2.0 generates an error message and stops.

LOOKUP ASSOCIATED EVENT LIST (*LOOKUPASSOCIATEDEVTLIST*)

The lookup associated event list method (*lookupAssociatedEvtList*) requires five parameters in order to retrieve an associated event list. The parameters are

- primary Event name,
- Aircraft 1 type,
- Aircraft 2 type,
- Sector 1 type, and
- Sector 2 type.

The pseudocode is shown in Figure 3-16.

Figure 3-16. Method lookupAssociatedEvtList Pseudocode

```
FOR each primary Event in the eventDictionary
      IF primary Event exists
         FOR each Key in KeyList of the primary Event
            IF AC1TYPE<>Key.AC1TYPE and DEF<>Key.AC1TYPE
               Mark this key
         END
         FOR each Key in KeyList of the primary Event
            IF AC2TYPE<>Key.AC2TYPE and DEF<>Key.AC2TYPE
               Mark this key
         END
         FOR each Key in KeyList of the primary Event
            IF SCT1TYPE<>Key.SCT1TYPE and DEF<>Key.SCT1TYPE
               Mark this key
         END
         FOR each Key in KeyList of the primary Event
       IF SCT2TYPE<> Key.SCT2TYPE and DEF<> Key.SCT2TYPE
               Mark this key
         END
END
FOR each Key in KeyList of the primary Event
      IF the Key is not Marked,
         Return the associated event list
END
IF no associated event list is found, THEN returns the de-
      fault associated event list
IF no default associated event list, THEN it is an error
```

TELL Method Process Trigger

This method is used to process Events from the a priori event file (*trig.evt)*. When it is brought into the simulation, the first primary Event activates an aircraft (e.g., UA1707) at a scheduled simulation time. Other Events include sector changes and the aircraft deactivation.

As an example, we repeat the sample *trig.evt* file from Table 3-3 in Table 3-5.

Table 3-5 has three primary Event vectors:

- UA1707 is activated under control of Sector 1 at simulation time 1000.
- UA1707 changes control from Sector 1 to Sector 2, also at simulation time 1000. Although scheduled for the same time, this Event will execute after the activation Event since it is after the activation Event in *trig.evt*. (Had it been before the activation Event, FAM 2.0 would have generated an error.)

- UA1707 is deactivated at simulation time 4000.

Once an a priori event vector is read from the *trig.evt* file, a trigger event object block *(trigevtObj)* is allocated and filled in from the data row in file *trig.evt* and then the trigger block is scheduled for execution in the future. The same *trigevtObj* is used for random events.

Table 3-5. Sample trig.evt File

NAME	1 TIME	2 AL	3 FN	4 AC_T	5 ALT_ AL	6 ALT_FN	7 SCT1	8 SCT2	9 ARPT	10 TRC	11 AOC	12 ACCHNL
ACTIVATE_AC	1000	UA	1707	747	NULL	0	1	0	NULL	NULL	NULL	0.0
SECT_CHG	1000	UA	1707	NULL	NULL	0	1	2	DEN	DEN	NULL	0.0
DEACTIVATE_AC	4000	UA	1707	NULL	NULL	0	2	0	NULL	NULL	NULL	0.0

Notes:

1. TIME—Simulation time of primary Event initiation.
2. AL—Airline of primary (#1) aircraft.
3. FN—Flight number of primary (#1) aircraft.
4. AC_T—Type of primary (#1) aircraft.
5. ALT_AL—Airline of secondary (#2) aircraft.
6. ALT_FN—Flight number of secondary (#2) aircraft.
7. SCT1—Identification number of primary (losing) sector.
8. SCT2—Identification number of secondary (gaining) sector.
9. ARPT—Name of airport.
10. TRC—Name of TRACON.
11. AOC—Name of AOC.
12. ACCHNL—Communications channel for aircraft-to-aircraft communications.

TRIGGER EVENT OBJECT BLOCK (*TRIGEVTOBJ*)

The trigger event object block (*trigEvtObj*) contains the information shown in Table 3-6.

Table 3-6. Trigger Event Object Block Contents

Field name	Data type	Description	Comments
ID	INTEGER	Number of Trigger Event	Mandatory
EVENT	STRING	Name of Trigger Event	Mandatory
TIME	REAL	Execution Time of Trigger Event	Mandatory
PRIORITY	REAL	Priority of the Event (e.g., 5.0)	Mandatory
AC1AN	STRING	Identifier of Aircraft 1 (e.g., UA)	Mandatory
AC1FL	INTEGER	Flight Number of Aircraft (e.g., 1707)	Mandatory
AC1TYPE	STRING	Type of Aircraft (e.g., 747)	Mandatory
AC2AN	STRING	Identifier of Aircraft 2 (e.g., UA)	Optional
AC2FL	INTEGER	Flight Number of Aircraft 2 (e.g., 1708)	Optional
SECT1NUM	INTEGER	Sector 1 number (e.g., 34)	Optional

Table 3-6. Trigger Event Object Block Contents (Continued)

Field name	Data type	Description	Comments
SECT2NUM	INTEGER	Sector 2 number (e.g., 99)	Optional
AIRPORT	STRING	Name of Airport (e.g., DENVER)	Optional
TRACON	STRING	Name of TRACON (e.g., DEN_TRA)	Optional
AOC	STRING	Name of AOC (UA)	Optional
CHANNEL	REAL	Channel value (e.g., 5.0)	Optional
startNextAssocevt	RESOURCEOBJ	Trigger to start next associated event	Mandatory
processTrigger	TELL method	Asynchronous method to process the trigger. This method is used for a pri-ori and random Events	

ASSOCIATED EVENT OBJECT (*ASSOCEVTOBJ*)

The associated event object (*assocevtObj*) holds information about the associated event that is taking place. For example, the primary Event Sector Change (*SECT_CHG*), has many associated events. One such associated event is Initial Call, which goes from the aircraft to the gaining sector. This Initial Call event is then scheduled for the aircraft object:

```
TELL evt[ORG] TO commenceEvt IN evt[DLY]
```

CROSS-REFERENCE BETWEEN *ASSOCEVTOBJ* AND *TRIGEVTOBJ*

Since there is only one associated event file for each type of primary Event, and the associated events in the file are executed each time that primary Event type occurs in the simulation, FAM 2.0 uses keywords for the event originator (*ORG*) and destination (*DST*) in the associated event file. FAM 2.0 translates these keywords into pointers to the addresses of actual objects in the simulation at run time. The model looks at various fields in the current *trigEvtObj* to resolve references of origin and destination addresses in the *assocevtObj*. Table 3-7 shows the location in the *trigEvtObj* object block for each generic identifier in associated events.

Table 3-7. Pointer Location in trigevtObj

Generic identifier	Pointer location
AC	trigevtObj[ac1an, ac1fn]
ALT_AC	trigevtObj[ac2an, ac2fn]
L_SECT	trigevtObj[sect1num]
G_SECT	trigevtObj[sect2num]
SECT	trigevtObj[sect1num]
AOC	trigevtObj[AOC]
TOWER	trigevtObj[airport]

Table 3-7. Pointer Location in trigevtObj (Continued)

Generic identifier	Pointer location
GROUND	trigevtObj[airport]
CLEARANCE	trigevtObj[airport]
OTHER	trigevtObj[airport]
APPROACH	trigevtObj[TRACON]
DEPARTURE	trigevtObj[TRACON]
FINAL	trigevtObj[TRACON]

PSEUDOCODE FOR *PROCESSTRIGGER* METHOD

Figure 3-17 contains the pseudocode for the process trigger method (*processTrigger*). To illustrate the process of assigning actual simulation addresses based on keywords, Figure 3-18 contains an example of a sector change associated event block **before** setting of ORG and DST pointers during the simulation. This is the event as it is derived from the event dictionary. Figure 3-19 shows how the originator and destination are changed to actual simulation objects.

Figure 3-17. ProcessTrigger Method Pseudocode

```
Find the matching primary Event in the Event Dictionary List
IF trigger name = ACTIVATE_AC
   Activate a new aircraft and generated random events if
   necessary
ELSE IF trigger name = DEACTIVATE_AC
   IF no more activities involving this aircraft
      Print statistics and deactivate it
   ELSE
      Print statistics and schedule the aircraft to deacti-
      vate later
ELSE
   Get the list of associated events for this Trigger Event
   from the eventDictionary (See Figure 11)
   Set all the ORG and DST pointers and MODE of each asso
   cevtObj in the list to the proper objects (See Figure 12)
FOR each assocevtObj in the associated event list
   Set evt[ORG] and evt[DST] according to Table 3-4
   TELL evt.orgptr TO commenceEvt (SELF, newassoc, org, 0.0,
   A PRIORI) IN evt.dly;
END
```

*Figure 3-18. Associated Event Block **Before** Setting ORG and DST*

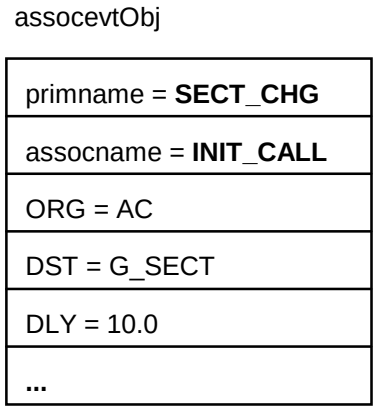
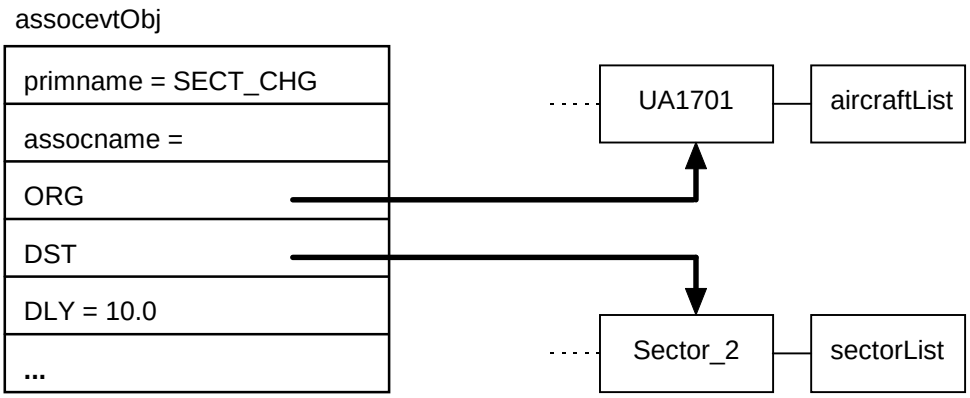


*Figure 3-19. Associated Event Block **After** Setting ORG and DST*



OPERATION OF *PROCESSTRIGGER*

An example would probably best illustrate the operation of *processTrigger*. Assume that the sector change primary Event (*SECT_CHG*) simply involves the four associated events shown in Table 3-8.

Table 3-8. Sample Sector Change Associated Events

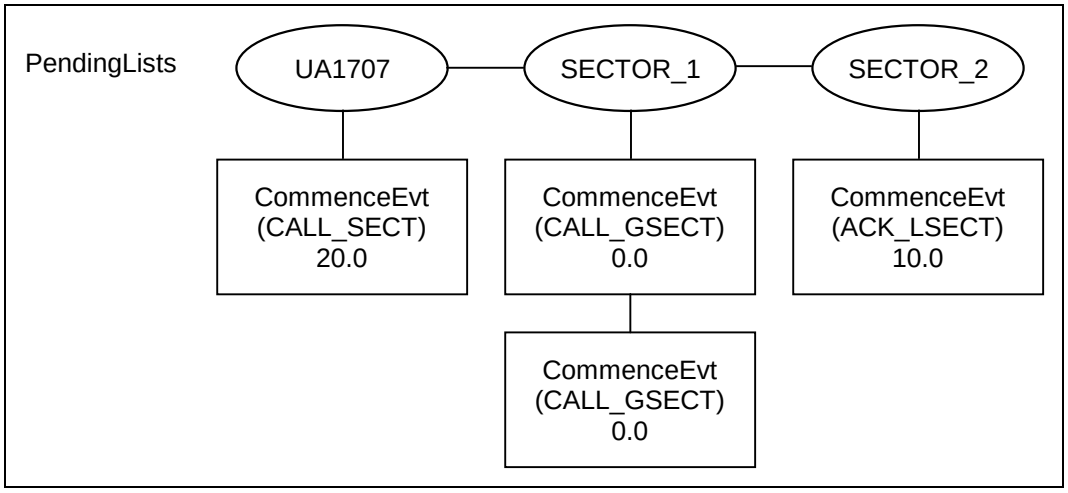
Primary Event	Associated event	ORG	DST	DLY
SECT_CHG	CALL_GSECT	L_SECT	G_SECT	0.0
SECT_CHG	ACK_LSECT	G_SECT	L_SECT	10.0
SECT_CHG	CALL_AC	L_SECT	AC	15.0
SECT_CHG	CALL_SECT	AC	G_SECT	20

When the simulation reaches the scheduled time of a *SECT_CHG* Event, *processTrigger* reads the associated event list. *ProcessTrigger* then places all four events on the pending list for the simulation object specified as ORG for later execution. *ProcessTrigger* uses method *commenceEvt* for all associated events (a priori or random). In this example, *processTrigger* uses the following line to schedule each associated event in the above list:

```
TELL evt[ORG] TO commenceEvt IN evt[DLY]
```

When all the associated events in our example have been scheduled, the pending list will look like Figure 3-20.

Figure 3-20. Event Pending Lists for Sector Change Event



There is a pending (event) list for each simulation object. In our example, with three objects involving four associated events in the sector change, Sector 1 will have two events and the other objects one apiece. The events are executed in time order, so the first event to be executed is *Call_GSectI,* followed in order by *Call_AC, Call_Sector, and Ack_Lsect.*

COMMENCE EVENT METHOD *(COMMENCEEVT)*

The commence event method (*commenceEvt*) is the main TELL method for *simObj*. In fact, this is the method that is at the heart of all event processing for FAM 2.0. All events, whether a priori or random, execute associated events via *commenceEvt*. *CommenceEvt* is called from TELL method *processTrigger* in *ItriggerMod.mod* only for the origin of an associated event. *ProcessTrigger* is the method that is executed when the simulation time equals the scheduled time for an a priori primary Event. Figure 3-21 shows the pseudocode for method *commenceEvt*.

Figure 3-21. Method commenceEvt Pseudocode

```
        IN trig       : trigEvtObj;
        IN evt        : assocevtObj;
        IN mode    : STRING;
        IN delta   : REAL;
        IN eventType  : STRING;

//trig - block containing primary Event data
//evt - block containing Associated Event data
//mode - ORG or DST
//delta - wait time
//eventType - A PRIORI or RANDOM
BEGIN
   IF simulation time > acdeactivationtime and the event
      type is RANDOM,
   THEN there will be no more random event allowed in
      commenceEvt ().
   ENDIF
   IF simulation time > simulationEnd
   THEN produce Warning message and quit
   ELSE
//SETUP FOR EXECUTION OF ASSOCIATED EVENT
      Set mate according to mode;
      IF mode = ORG
         NEW (actblk);
         Store beginwaittime in actblk for both ORG and DST
         Make sure that the previous associated event has
            been completed
         Reserve both ORG and DST (put in a request to
            acquire the reserve resource)
         Lock the occupied resource of SELF
         Figure out which object (ORG/DST) will have its
            channel occupied
         IF it is AC-AC communication, lock channel
            specified in TRIG.EVT
      IF ORG <> DST, make the DST busy
         deltadst = time we had to wait to occupy the DST
      TELL DST TO commenceEvt
         delta:= time we had to wait to occupy SELF
            (i.e., ORG)
         Release the reserved resource for ORIGIN and
             DESTINATION
      IF waitingTime > 0.0, update wait statistics for ORG
         and DST
```

Figure 3-21. Method commenceEvt Pseudocode (Continued)

```
//START EXECUTION OF ASSOCIATED EVENT
   FOREACH controller IN controllerQue
      lookup load for this primary event, associated event,
         mode and mate-type
      IF load > 0.0 update all continuous tasking statistics
          ASK each controller to transmit (load)
          FOREACH radio IN commQue
      lookup load for this primary event, associated event,
         mode and mate-type
      IF load > 0.0 update all continuous tasking statistics
         ASK each radio to transmit (load)
         FOREACH equipment IN equipmentQue
      lookup load for this primary event, associated event,
         mode and mate-type
      IF load > 0.0 update all continuous tasking statistics
         ASK each radio to transmit (load)
//FINISH EXECUTION OF ASSOCIATED EVENT
   Release the startNextAssocevt resource in the TRIGGER
      EVENT so that the next associated event can proceed
   Now WAIT for Maximum duration maximum of all loads above
   Unlock the appropriate channel for this activity
   Release the occupied resource in this object (ORG/DST)
      so that any events involving this object can proceed
   IF simObjType = AIRCRAFT and we are past its
      deactivation time,
      IF mode = ORG
      THEN dispose of the actblk
   THEN deactivate it now
END METHOD;


BEGIN
   IF simulation time > acdeactivationtime and the event
      type is RANDOM,
   THEN there will be no more random event allowed in
      commenceEvt ().

   IF simulation time > simulationEnd
   THEN produce Warning message and quit
   ELSE
//SETUP FOR EXECUTION OF ASSOCIATED EVENT
      Set mate according to mode;
      IF mode = ORG
         NEW (actblk);
         Store beginwaittime in actblk for both ORG and DST
         Make sure that the previous associated event has
            been completed
         Reserve both ORG and DST (put in a request to
            acquire the reserve resource)
```

Figure 3-21. Method commenceEvt Pseudocode (Continued)

```
         Lock the occupied resource of SELF
         Figure out which object (ORG/DST) will have its
            channel occupied
         IF it is AC-AC communication,
         THEN lock channel specified in TRIG.EVT
         IF ORG <> DST,
         THEN make the DST busy
            deltadst = time we had to wait to occupy the DST
   TELL DST TO commenceEvt
      delta:= time we had to wait to occupy SELF (i.e., ORG)
      Release the reserved resource for ORIGIN and
         DESTINATION
         IF waitingTime > 0.0, update wait statistics for
            ORG and DST
START EXECUTION OF ASSOCIATED EVENT
   FOREACH controller IN controllerQue
      lookup load for this primary event, associated event,
         mode and mate-type
      IF load > 0.0
         THEN update all continuous tasking statistics
         ASK each controller to transmit (load)
   FOREACH radio IN commQue
      lookup load for this primary event, associated event,
         mode and mate-type
      IF load > 0.0
         THEN update all continuous tasking statistics
      ASK each radio to transmit (load)
   FOREACH equipment IN equipmentQue
      lookup load for this primary event, associated event,
         mode and mate-type
      IF load > 0.0
         THEN update all continuous tasking statistics
            ASK each radio to transmit (load)
FINISH EXECUTION OF ASSOCIATED EVENT
   Release the startNextAssocevt resource in the TRIGGER
      EVENT so that the next associated event can proceed
   Now WAIT for Maximum duration = maximum of all loads
      above
   Unlock the appropriate channel for this activity
   Release the occupied resource in this object (ORG/DST) so
      that any events involving this object can proceed
   IF mode = ORG
   THEN dispose of the actblk
   IF simObjType = AIRCRAFT and we are past its deactivation
      time,
   THEN deactivate it now

END METHOD
```

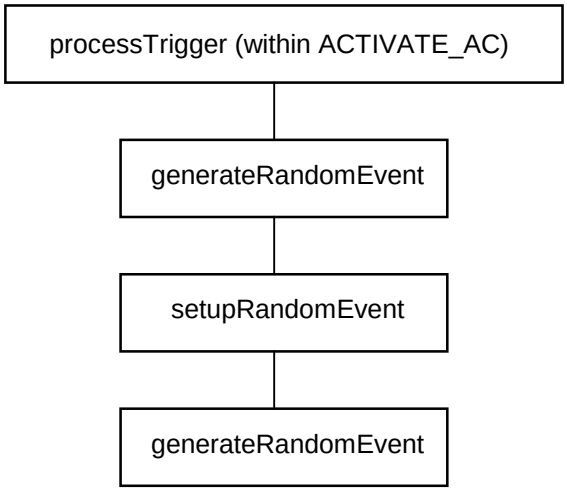
Random Event Processing

Random events are processed slightly differently than a priori events. For random events, FAM 2.0 uses a uniform, real distribution to choose an event from the random event file (*rand.evt*). For example, if *rand.evt* contains FIRE, ALT_CHG, LOW_FUEL as random event types, when FAM 2.0 generates a random event, each type has a 33.3 percent probability of being "chosen" as the random event type.

TELL METHODS

Figure 3-22 shows the logic flow of random event TELL methods.

Figure 3-22. Logic Flow of Random Event TELL Methods



PROCESSING RANDOM EVENTS

Figure 3-23 shows the pseudocode for TELL method *processTrigger*, which is executed when an aircraft is activated.

Figure 3-23. TELL method processTrigger

```
//When activating an aircraft:
IF random_mode = TRUE
THEN schedule generateRandomEvent
//This method is contained in ItriggerMod.mod.
```

Figure 3-24 shows the pseudocode for TELL method *generateRandomEvent.*

Figure 3-24. TELL method generateRandomEvent

```
//This TELL Method is contained in IaircraftMod.mod.
Find the activation and deactivation time of SELF (aircraft)
IF [(SimTime < deactivation time) AND (SimTime < simula-
    tionEnd)]
   WAIT until the AC has begun executing the first
      associated event of its first a priori primary Event
   REPEAT until [(SimTime < deactivation time) AND
      (SimTime < simulationEnd)]
   WAIT DURATION stream1.UniformReal (minInterRandomTime,
      maxInterRandomTime);
   IF (SimTime < deactivation time) AND (SimTime <
      simulationEnd)
         Select a random number from stream2.UniformInt
            (1, 100) and depending on the number of random
            Events (in rand.evt), pick a random Event for
            simulation E.g., if there are 10 random Events,
            and we picked 70, then we would choose random
            Event #7 from the list
         Increment numRandomEventsGenerated
         Setup a trigger Event with all relevant for this
            random primary Event
         Lookup associated event list in the event
            dictionary for this random primary Event
         ASK SELF TO setupRandomEvent (assocevtlist, trigger
            Event);
      ENDIF
   END REPEAT
ENDIF
END METHOD;
```

TELL Method *setupRandomEvent*

The TELL method *setupRandomEvent* is very much like *processTrigger* in its logic. This method is contained in *IaircraftMod.mod*. It sets up the ORG and DST pointers of each associated event block and then schedules the associated events via *commenceEvt*. The pseudocode for *setupRandomEvent* is shown in Figure 3-25.

Figure 3-25. TELL Method setupRandomEvent Pseudocode

```
//The input parameters are:
   IN aelist  : QueueObj;
   IN trig    : trigEvtObj

BEGIN
   Create a list of associated events called newassoclist
   FOREACH assocevtblk IN aelist;
      Increment eventctr
      Create a assocevtObj called newassoc and clone the
         associated event block from aelist
      Set the ORG pointer of newassoc (based on keywords,
         e.g., SECT, AC, TOWER, etc.)
      Set the DST pointer of newassoc (based on keywords,
         e.g., SECT, AC, TOWER, etc.)
      Add the newassoc to newassoclist
   END FOREACH

   FOREACH newassoc IN newassoclist
      TELL newassoc.orgptr TO commenceEvt (trig,
         newassoc,org, 0.0, RANDOM) IN newassoc.dly
   END FOREACH

   Dispose of newassoclist
END METHOD
```

OUTPUT

The output statistics are defined for personnel (*controllerObj*), radio (*sysObj*), and equipment (*sysObj*) of each simulation object (i.e., aircraft, sector, AOC, TRACON controller, airport controller, and channel). There are nine statistics:

- MAX_WAITING—the maximum number of associated events waiting.
- AVE_QUE_LEN—the average queue length of associated events.
- MAX_WAIT_TIME—the maximum wait time for associated events.
- AVE_WAIT_TIME—the average wait time for associated events.
- NUM_SERVED—the number of associated events served.
- TOTAL_TASK_TIME—the total task time.
- PERC_TASKED—the total task time/simulation length.
- MAX_CONT_TIME—the longest period of continuous tasking.
- AVE_CONT_TIME—the average period of continuous tasking.

In addition, FAM produces statistics on the number of aircraft in each sector:

- SECTOR_ID—the sector ID.
- NUM_AIRCRAFT—the number of aircraft in that sector.
- MAX_AIRCRAFT—the maximum number of aircraft in that sector.
- AVE_AIRCRAFT—the average number of aircraft in that sector.

Resetting Output Parameters

Resetting output parameters is based on the user input of <*stat_start*> specified in the scenario file. In *IprocedureMod.mod*, a TELL method *resetOutputStat-Counters* is scheduled at stat_start time to wipe out any statistics collected so far in various simulation objects. The statistics are actually stored in personnel (*controllerObj*), radio (*sysObj*), and equipment (*sysObj*) of each simulation object. The entire aircraft list, sector list, AOC list, TRACON controller list, and airport controller list are traversed in order to reset the statistical counters of all the personnel/radio/equipment of each simulation object in these lists.

Gathering Output Statistics

The gathering of output statistics is done via LMONITORED objects of MODSIM. The ASK method *printQueStats*, which belongs to personnel (*controllerObj*) and radio/equipment (*sysObj*) of each simulation objects is prints out all the statistics gathered for those objects.

WAITING TIME AND NUMBER OF WAITS STATISTICS

Waiting time and number of waits statistics are collected at the simulation object level via LMONITORED real and integer data types. Table 3-9 lists the relevant fields in *simObj*.

Table 3-9. Wait Statistics Fields in simObj

Statistics counters	Type
waitingTime	LMONITORED REAL
waitingTimeStats	rStatObj
numberOfWaits	LMONITORED INTEGER
numberOfWaitsStats	iStatObj

These statistics are updated in method *commenceEvt* in *IsimMod.mod*. The data is then copied to corresponding variables for the active personnel, radio, and equipment in that simulation object for that associated event.

CONTINUOUS TASKING STATISTICS

The continuous tasking statistics are updated in method *updateContTaskStats* for personnel (*controllerObj*) and radio/equipment (*sysObj*) in IsimMod.mod. This method is called from commenceEvt.

Statistics of interest are the longest period of continuous tasking and average period of continuous tasking. The method *updateContTaskStats* simply updates the counter with the loads for the current associated event and keeps track of whether or not the object (personnel/radio/equipment) has been continuously tasked.

NUMBER OF ACTIVITIES SERVED (*NUMSERVED*) STATISTIC

The number of activities served statistic is gathered in method transmit for personnel (*controllerObj*) and radio/equipment (*sysObj*) in *IsimMod.mod*. This method is called from *commenceEvt*. Every time FAM 2.0 calls `transmit()`, it increments *numServed* by 1.

CONTROLLER AND SYSTEMS OBJECTS STATISTICS

Table 3-10 shows the statistics counter variables in *controllerObj* and *sysObj*.

Table 3-10. Controller and Systems Objects Statistics Counters

Statistical counters	Description
maxWaitingTime	Maximum waiting time for associated events
aveWaitingTime	Average waiting time for associated events
maxNumWaiting	Maximum number of associated events waiting
aveNumWaiting	Average number of associated events waiting
numServed	Number of events served
totalTaskTime	Total tasking time
contTaskTime	Longest period of continuous tasking
contTaskTimeStats	Statistics object for the continuous task time
intervalTotalTaskTime	Period of continuous tasking
percTasked	Total task time/simulation length

Final Report

In the module *IfinalReportMod.mod*, the TELL method *Print* produces the final report for sectors, AOCs, TRACON controllers, airport controllers, channels, and miscellaneous aircraft/sector statistics.[1] If there are active aircraft at the end of the simulation, then the statistics for these aircraft are printed at that time. This

[1] The exception is the aircraft object type. The aircraft prints its own statistics before it deactivates. All other simulation objects print their statistics at the time of the final report.

method is scheduled for execution in the main module. Figure 3-26 contains the pseudocode to produce the final report.

Figure 3-26. Final Report Production Pseudocode

```
BEGIN
IF NumActPending > 0, produce Warning and StopSimulation

//AIRCRAFT STATISTICS (For aircraft that have not been
   deactivated)
   FOREACH aircraft IN aircraftList
      ASK aircraft TO printACStats;
   END FOREACH;
   Print The number of aircraft in the model is
   Print Column headers

//ARTCC STATISTICS
   FOREACH sector IN sectorList
      ASK sector TO printSectorStats;
   END FOREACH;

//AOC STATISTICS
   FOREACH AOC IN AOCList
      ASK AOC TO printAOCStats;
   END FOREACH;

//AIRPORT STATISTICS
   FOREACH airport IN aptContList
      ASK airport TO printAirportStats;
   END FOREACH;

//TRACON STATISTICS
   FOREACH TRACON IN TRACONContList
      ASK TRACON TO printTRACONContStats;
   END FOREACH;

//CHANNEL STATISTICS
   Write out column headers
   FOREACH channel IN channelList
      ASK channel TO printStats;
   END FOREACH;

//AIRCRAFT/SECTOR STATISTICS
   Print Column headers of SECTOR_ID, NUM_AIRCRAFT,
MAX_AIRCRAFT, AVE_AIRCRAFT
   FOREACH sector IN sectorList
      write out the statistics
   END FOREACH;
   Report error or warning if there is any.
END METHOD;
```

Error Processing

ERROR MESSAGES

There are two different types of errors: file processing errors and run time errors (errors occurring during the simulation run). Once an error is detected, FAM 2.0 will attempt to catch additional errors in the same file before quitting the model. However, if the error is serious and the model cannot possibly continue, it will quit. Upon quitting, FAM 2.0 will generate all the error messages and direct them to the *fam.err* file. If the error occurs during file processing, then the corresponding file name, column and row will be shown. This mechanism provides the flexibility for the user to correct more than just one error.

Examples of file processing errors are

- missing a load value in the load file and
- a simulation object type does not exist.

Examples of run time errors are

- missing simulation object or a requested object no longer exists and
- a sector change occurs but the user does not specify the losing sector.

Appendix A contains a list of all error messages and their meanings.

WARNING MESSAGES

FAM 2.0 also identifies potential problems that might cause erroneous output statistics. These problems are considered to be nonfatal, and FAM 2.0 will continue to execute. For example, if the user intends to deactivate an aircraft at time 1000, and there are activities pending, then FAM will issue a warning message in *fam.err* and schedule the aircraft for automatic deactivation. This will help the aircraft complete its activities and gather all the necessary statistics before deactivation.

Appendix A
FAM 2.0 Error & Warning Messages

"Error: number of pilots must be one or more."

"Error: number of communication devices must be zero or more."

"Error: number of equipment must be zero or more."

"Error: Random event *'random event name'* does not exist."

"Error: There is no AOC *'aoc name'* for primary event *'primary event name'* and associated event *'associated event name'* at time *'t'* with delay *'delay'* [MODE=ORG]"

"Error: There is no tower controller for airport *'airport name'* (primary event *'primary event name'* and associated event *'associated event name'* at time *'t'* with delay *'delay'*) [MODE=ORG]"

"Error: There is no ground controller for airport *'airport name'* (primary event *'primary event name'* and associated event *'associated event name'* at time *'t'* with delay *'delay'*) [MODE=ORG]"

"Error: There is no clearance controller for airport *'airport name'* (primary event *'primary event name'* and associated event *'associated event name'* at time *'t'* with delay *'delay'*) [MODE=ORG]"

"Error: There is no other controller for airport *'airport name'* (primary event *'primary event name'* and associated event *'associated event name'* at time *'t'* with delay *'delay'*) [MODE=ORG]"

"Error: There is no TRACON controller for approach with TRACON *'TRACON name'* and airport *'airport name'* (primary event *'primary event name'* and associated event *'associated event name'* at time *'t'* with delay *'delay'*) [MODE=ORG]"

"Error: There is no TRACON controller for departure with TRACON *'TRACON name'* and airport *'airport name'* (primary event *'primary event name'* and associated event *'associated event name'* at time *'t'* with delay *'delay'*) [MODE=ORG]"

"Error: There is no TRACON controller for final with TRACON *'TRACON name'* and airport *'airport name'* (primary event *'primary event name'* and asso-

ciated event *'associated event name'* at time *'t'* with delay *'delay'*) [MODE=ORG]"

"Error: There is no AOC *'aoc name'* for primary event *'primary event name'* and associated event *'associated event name'* at time *'t'* with delay *'delay'* [MODE=DST]"

"Error: There is no tower controller for airport *'airport name'* (primary event *'primary event name'* and associated event *'associated event name'* at time *'t'* with delay *'delay'*) [MODE=DST]"

"Error: There is no ground controller for airport *'airport name'* (primary event *'primary event name'* and associated event *'associated event name'* at time *'t'* with delay *'delay'*) [MODE=DST]"

"Error: There is no clearance controller for airport *'airport name'* (primary event *'primary event name'* and associated event *'associated event name'* at time *'t'* with delay *'delay'*) [MODE=DST]"

"Error: There is no other controller for airport *'airport name'* (primary event *'primary event name'* and associated event *'associated event name'* at time *'t'* with delay *'delay'*) [MODE=DST]"

"Error: There is no TRACON controller for approach with TRACON *'TRACON name'* and airport *'airport name'* (primary event *'primary event name'* and associated event *'associated event name'* at time *'t'* with delay *'delay'*) [MODE=DST]"

"Error: There is no TRACON controller for departure with TRACON *'TRACON name'* and airport *'airport name'* (primary event *'primary event name'* and associated event *'associated event name'* at time *'t'* with delay *'delay'*) [MODE=DST]"

"Error: There is no TRACON controller for final with TRACON *'TRACON name'* and airport *'airport name'* (primary event *'primary event name'* and associated event *'associated event name'* at time *'t'* with delay *'delay'*) [MODE=DST]"

"Error: number of controllers must be one or more."

"Error: number of dispatchers must be one or more."

"Error: File *'file name'* does not exist."

"Error: Error occurs while opening file *'file name'*."

"Error: Expected *'token'* (Error occurs in file *'file name'* at line # and column #)"

"Error: Expected '=' (Error occurs in file *'file name'* at line # and column #)"

"Error: *'token'* is not a valid real number; a real value is expected."

"Error: Expected a real value."

"Error: Expected a string."

"Error: Mode *'type name'* is invalid."

"Error: Type *'type name'* does not exist."

"Error: Loads expected in file *'file name'*."

"Error: *'token'* is not a valid integer number; an integer value is expected."

"Error: Expected integer value."

"Warning: There are pending activities at time *'simulation time'*. "

"Error: index value *'value of index'* is out of range."

"Error: Time of a priori event must be in increasing order."

"Error: Time of activity for aircraft with airline *'airline'* and flight number *'flightnumber'* is less than its activation time."

"Error: Time of activity for aircraft with airline *'airline'* and flight number *'flightnumber*" is greater than or equal to its deactivation time."

"Error: Primary event *'primary event name'* is not in the event dictionary."

"Error: Sector type *'type name'* is not in the sector dictionary."

"Error: AOC type *'type name'* is not in the AOC dictionary."

"Error: Airport controller type *'type name'* is not in the airport controller dictionary."

"Error: TRACON controller type *'type name'* is not in the TRACON controller dictionary."

"Error: There is no TRACON controller for TRACON named *'TRACON name'* and controller named *'controller name'*. "

"Error: Airport position must be defined in the TRACON dictionary file."

"Error: No random event was specified."

"Error: stat_start must be: stat_start <= simulation_end"

"Error: Random mode must be TRUE or FALSE."

"Error: Reuse seed mode must be TRUE or FALSE."

"Error: *'tag name'* is an invalid tag."

"Error: tag *'tage name'* is already defined."

"Error: *'token'* must be preceded by a proper tag."

"Error: AOC *'aoc name'* does not exist."

"Error: Airport *'airport name'* does not exist."

"Error: TRACON *'TRACON name'* does not exist."

"Warning: Trigger event *'trigger name'* and associated event *'associated event name'* at time *'simulation time'* cannot be processed because the simulationEnd time has occurred."

"Error: There is no default load for primary event *'primary event name'*, associated event *'associated event name'*, mode *'mode name'* and type *'type name'*."

"Error: Previous activity ended after current time."

"Error: ACTIVATE_AC at time *'time'* fails; aircraft type *'type name'* is not in the aircraft dictionary."

"Warning: You are trying to deactivate aircraft *'aircraft name'* at time *'simulation time'*, and there are pending activities. Adjust its deactivation time."

"Error: DEACTIVATE_AC at time *'time'* fails; aircraft *'aircraft name'* does not exist."

"Error: Primary event *'primary event name'* at time *'time'* fails; aircraft *'aircraft name'* does not exist."

"Error: Primary event at time *'time'* fails; sector # *'sector number'* does not exist."

"Error: There is no associated event list for primary event *'primary event name'*, aircraft 1 type *'type name'*, aircraft 2 type *'type name'*, sector 1 type *'type name'*, sector 2 type *'type name'* at time *'time'*. "

"Error: There is no AOC *'aoc name'* for primary event *'primary event name'* and associated event *'associated event name*" at time *'time'* with delay *'delay'* [MODE=ORG]."

"Error: There is no tower controller for airport *'airport name'* (primary event *'primary event name'* and associated event *'associated event name'* at time *'time'* with delay *'delay'*) [MODE=ORG]"

"Error: There is no ground controller for airport *'airport name'* (primary event *'primary event name'* and associated event *'associated event name'* at time *'time'* with delay *'delay'*) [MODE=ORG]"

"Error: There is no clearance controller for airport *'airport name'* (primary event *'primary event name'* and associated event *'associated event name'* at time *'time'* with delay *'delay'*) [MODE=ORG]"

"Error: There is no other controller for airport *'airport name'* (primary event *'primary event name'* and associated event *'associated event name'* at time *'time'* with delay *'delay'*) [MODE=ORG]"

"Error: There is no TRACON controller for approach with TRACON *'TRACON name'* and airport *'airport name'* (primary event *'primary event name'* and associated event *'associated event name'* at time *'time'* with delay *'delay'*) [MODE=ORG]."

"Error: There is no TRACON controller for departure with TRACON *'TRACON name'* and airport *'airport name'* (primary event *'primary event name'* and associated event *'associated event name'* at time *'time'* with delay *'delay'*) [MODE=ORG]."

"Error: There is no TRACON controller for final with TRACON *'TRACON name'* and airport *'airport name'* (primary event "primary event name" and associated event *'associated event name'* at time *'time'* with delay *'delay'*) [MODE=ORG]."

"Error: There is no AOC *'aoc name'* for primary event *'primary event name'* and associated event *'associated event name*" at time *'time'* with delay *'delay'* [MODE=DST]."

"Error: There is no tower controller for airport *'airport name'* (primary event *'primary event name'* and associated event *'associated event name'* at time *'time'* with delay *'delay'*) [MODE=DST]"

"Error: There is no ground controller for airport *'airport name'* (primary event *'primary event name'* and associated event *'associated event name'* at time *'time'* with delay *'delay'*) [MODE=DST]"

"Error: There is no clearance controller for airport *'airport name'* (primary event *'primary event name'* and associated event *'associated event name'* at time *'time'* with delay *'delay'*) [MODE=DST]"

"Error: There is no other controller for airport *'airport name'* (primary event *'primary event name'* and associated event *'associated event name'* at time *'time'* with delay *'delay'*) [MODE=DST]"

"Error: There is no TRACON controller for approach with TRACON *'TRACON name'* and airport *'airport name'* (primary event "primary event name" and associated event *'associated event name'* at time *'time'* with delay *'delay'*) [MODE=DST]"

"Error: There is no TRACON controller for departure with TRACON *'TRACON name'* and airport *'airport name'* (primary event "primary event name" and associated event *'associated event name'* at time *'time'* with delay *'delay'*) [MODE=DST]"

"Error: There is no TRACON controller for final with TRACON *'TRACON name'* and airport *'airport name'* (primary event "primary event name" and associated event *'associated event name'* at time *'time'* with delay *'delay'*) [MODE=DST]"

"Error: Expected tag *'tag name'*."

"Error: Sector *'sector ID'* does not exist."

"Error: Channel *'channel value'* could not be found."

"Error: Aircraft *'aircraft name'* does not exist; primary event *'primary event name'* and associated event *'associated event name*" fails at time *'t'*; you must specify an aircraft when an associated event involves AC."

"Error: Aircraft *'aircraft name'* does not exist; primary event *'primary event name'* and associated event *'associated event name*" fails at time *'t'*; you must specify an aircraft when an associated event involves ALT_AC."

"Error: There is no sector ID *'sector ID'* for primary event *'primary event name'* and associated event *'associated event name';* you must specify the sector ID when an associated event involves SECT."

"Error: There is no sector ID *'sector ID'* for primary event *'primary event name'* and associated event *'associated event name';* you must specify the sector ID when an associated event involves L_SECT."

"Error: There is no sector ID *'sector ID'* for primary event *'primary event name'* and associated event *'associated event name';* you must specify the sector ID when an associated event involves G_SECT."

"Error: scenario file *'file name'* does not exist."

Appendix B
FAM 2.0 Test Plan

This appendix details the testing that LMI carried out on FAM 2.0. The first section details the testing done during the model's development. The second section contains a chart showing the validation and verification testing on the finished model.

DEVELOPMENT TESTS

Random Event Processing

The *random.evt* file is a master event file that contains a list of the random events to be used in simulation. If the random mode is TRUE (in the scenario file), random events will be generated for a particular aircraft between its activation time and deactivation time. However, if the simulation end time occurs before the aircraft is deactivated, random events will be generated up to the simulation end time. Table B-1 contains the tests for random event processing.

Table B-1. Random Event Processing Tests

Scenario	Required result
Sequential a priori events: $T(event_1) < T(event_2) < \ldots < T(event_n)$	No activities will be queued.
Concurrent a priori events: $T(event_1)\ (T(event_2)\ (\ldots\ (T(event_n)$	Activities will be queued.
Simulation end occurs before activities are completed.	Warning: There are pending activities at time T. Simulation continues until all scheduled events are completed.
$T(event_i) > T(event_j)$ and $i < j$	Error: Time of a priori event must be in increasing order of time.
Missing data in columns TIME, ACCHNL	Error: Expected a real value.
Missing data in columns ACAL, AC_T, ALT_ACAL, ARPT, TRC, AOC	Error: Expected a string.
Missing data in columns ACFN, ALT_ACFN, SCT1, SCT2	Error: Expected integer value.
A priori event involving two aircraft but, ACCHNL is not specified	Error: Channel is required for activities between two different aircraft.

Event Dictionary

The event dictionary contains one or more associated event lists for each primary Event (a priori or random). The associated event list depends on the types of aircraft 1, aircraft 2, sector 1, and sector 2. Not all events involve two aircraft and two sectors. FAM matches the corresponding objects from the primary Event vector (in *trig.evt*) to their types in the event dictionary.

For example, if a SECT_CHG in *trig.evt* involves an aircraft and two sectors, then FAM will match SECT_CHG and column AC1TYP, SECT1TYP and SECT2TYP in order to find the correct associated event file. DEF is allowed for these column values. NULL is only allowed for AC2TYP since there is no second aircraft in the SECT_CHG vector in *trig.evt*. Table B-2 contains the test for event dictionary error detection.

Table B-2. Event Dictionary Error Detection Test

Scenario	Required result
SECT_CHG involving one aircraft and two sectors in *trig.evt*, but SECT_CHG row in event dictionary (event.dic) contains: //EVTNAME…AC1TYP…AC2TYP SCT1TYP…SCT2TYP…ASSOCFILE PRIORITY SECT_CHG…747…NULL NULL…NULL…sect_chg.evt ... 5.0 This vector should be (italics for emphasis) SECT_CHG…747…NULL *SECTOR_A…SECTOR_A*…sect_chg.evt… 5.0	Error message: “Error: There is no associated event list for primary event” “*SECT_CHG*”, aircraft 1 type “*747*”, aircraft 2 type “*NULL*”, sector 1 type “*SECTOR_A*”, sector 2 type “*SECTOR_A*” at time t.”

Simulation Objects

AIRCRAFT

Each aircraft will be brought into simulation based on its activation and deactivation time. Once an aircraft is activated, it remains in simulation until it is deactivated. However, the deactivation time of an aircraft must be strictly greater than its activation time, as well as the times for any events scheduled for that aircraft. Time of any event involving aircraft j must be greater than or equal to its activation time and strictly less than its deactivation time. Table B-3 contains the tests for aircraft event sequencing.

Table B-3. Aircraft Event Sequencing Tests

Scenario	Expected result
Sequential events: No events are overlapping in time 2 aircraft 2 TAKE_OFF Events (different time) aircraft1: First TAKE_OFF aircraft2: Second TAKE_OFF First TAKE_OFF completes before second TAKE_OFF	Aircraft will be deactivated at specified time after second TAKE_OFF.
Sequential events: Second event involving second aircraft continues after its deactivation time 2 aircraft 2 TAKE_OFF Events (different time) aircraft1: First TAKE_OFF aircraft2: Second TAKE_OFF Second TAKE_OFF does not complete before deactivation of second aircraft	Warning: Aircraft #2 can not be deactivated at time *t*. Adjust its deactivation time. When the last activity involving this aircraft is completed, this aircraft will be deactivated automatically by FAM 2.0.
Concurrent events: events are overlapping 1 aircraft 1 TAKE_OFF Event and 1 SECT_CHG Event (same time)	Events will queue up for the aircraft. TAKE_OFF will happen before SECT_CHG. time.

AIR ROUTE TRAFFIC CONTROL CENTER

An ARTCC is a collection of sectors. For sector change events, sectors serve as either a losing sector or gaining sector. Sectors are active throughout simulation. Each sector will serve only one activity at a time. If the same sector is being requested more than once at the same time, FAM 2.0 will queue these activities for this sector until the completion of the previous activity. If an activity involves a sector, the sector number must be provided in the a priori event file. If the sector number is not specified or incorrect, simulation will stop during runtime. This error cannot be caught before start of simulation. Table B-4 contains the ARTCC tests.

Table B-4. ARTCC Tests

Scenario	Required result
1 aircraft, 1 sector 1 TAKE_OFF Event: aircraft1, sector1	Statistics collected for aircraft and sector.
1 aircraft 2 sectors 1 SECT_CHG Event: aircraft1, sector1, sector2	Statistics collected for aircraft1, sector1, and sector2.

Table B-4. ARTCC Tests (Cont.)

Scenario	Required result
1 aircraft, 2 sectors 2 SECT_CHG Events (different time) aircraft1: sector1 => sector2 aircraft1: sector2 => sector1	Statistics collected for aircraft1, sector1, and sector2.
1 aircraft, 3 sectors 2 SECT_CHG Events (different time) aircraft1: sector1 => sector2 aircraft1: sector2 => sector3	Statistics collected for aircraft1, sector1, sector2, and sector3.
2 aircraft, 2 sectors 2 SECT_CHG Events (same time) aircraft1: sector1 => sector2 aircraft2: sector2 => sector1	Statistics collected for aircraft1, aircraft2, sector1, and sector2. Events queue up for sectors 1 and 2. No deadlocks for resource acquisition.
2 aircraft, 3 sectors 2 SECT_CHG Events (same time) aircraft1: sector1 => sector2 aircraft2: sector2 => sector3	Statistics collected for aircraft1, aircraft2, sector1, sector2, and sector3; events queue up for sector 2.

AIRLINE OPERATIONS CENTER

There is no limit to how many AOCs a user can define. If an AOC is defined, it will remain active throughout simulation. If an activity involves an AOC, the AOC name must be provided in the a priori event file. If the AOC name is not specified or incorrect, simulation will stop. Table B-5 contains the test plan for AOC events.

Table B-5. AOC Event Test Plan

Scenario	Required result
1 aircraft 1 AOC 1 TAKE_OFF Event aircraft1: AOC1	Statistics collected for aircraft1 and AOC1.
1 aircraft 1 AOC 2 TAKE_OFF Events (different time) aircraft1: AOC1 aircraft1: AOC1	Statistics collected for aircraft1 and AOC1.
1 aircraft 2 AOC 2 TAKE_OFF Events (different time) aircraft1: AOC1 aircraft1: AOC2	Statistics collected for aircraft1, AOC1, and AOC2.

Table B-5. AOC Event Test Plan (Cont.)

Scenario	Required result
2 aircraft 2 AOC 2 TAKE_OFF Events (different time) aircraft1: AOC1 aircraft2: AOC2	Statistics collected for aircraft1, aircraft2, AOC1, and AOC2.
2 aircraft 1 AOC 2 TAKE_OFF Events (same time) aircraft1: AOC1 aircraft2: AOC1	Statistics collected for aircraft1, aircraft2, and AOC1; events queue up for AOC 1.
2 aircraft 2 AOC 2 TAKE_OFF Events (same time) aircraft1: AOC1 aircraft2: AOC2	Statistics collected for aircraft1, aircraft2, AOC1, and AOC2.
1 aircraft 1 sector 1 AOC 1 TAKE_OFF Event aircraft1, sector1, AOC1	Statistics collected for aircraft1, sector1, and AOC1.

AIRPORT CONTROLLER

There are four different types of controllers for each airport:

- Tower
- Ground
- Clearance
- Other.

An airport controller (e.g., tower) is a collection of personnel, radios, and equipment. The airport controller must be defined before it can be used. Once the controller is defined, it will remain in the simulation throughout the run. If an activity involves a controller (e.g., tower, ground, clearance, other), the airport name must be provided in the a priori event file. If the airport name is not specified or incorrect, simulation will stop.

Table B-6 contains the test plan for airport controllers. The scenarios in Table B-6 use only tower as the airport controller. The same scenarios have been applied for any of the other three controller types.

Table B-6. Airport Controller Tests

Scenario	Required result
1 aircraft 1 tower 1 TAKE_OFF Event aircraft1: tower1	Statistics collected for aicraft1 and tower1.
2 aircraft 1 tower 2 TAKE_OFF Events (different time) aircraft1: tower1 aircraft2: tower1	Statistics collected for aircraft1, aircraft2, and tower1.
2 aircraft 1 tower 2 TAKE_OFF Events (same time) aircraft1: tower1 aircraft2: tower1	Statistics collected for aircraft1, aircraft2, and tower1; events queue up for tower 1.
1 aircraft 1 tower 1 sector 1 TAKE_OFF Event aircraft1, tower1, sector1	Statistics collected for aircraft1, tower1, and sector1.
1 aircraft 1 tower 1 sector 1 AOC 1 TAKE_OFF Event aircraft1, tower1, sector1, AOC1	Statistics collected for aircraft1, tower1, sector1, and AOC1.

TRACON CONTROLLER

Each TRACON controller may serve more than one position and more than one airport. There are three positions: approach, departure and final. Each TRACON controller must be defined and assigned an airport name and position before it can be used. A TRACON controller may serve

- all three positions for the same airport,
- different airports and different positions, or
- different airports and the same positions.

If an activity involves a TRACON controller, the position (e.g., approach, departure, final), the airport name and TRACON name must be specified in the a priori event file. If the TRACON controller of the given TRACON name does not serve the given airport name and position, simulation will stop.

Table B-7 contains the first phase of the test plan for TRACON controller objects. The scenarios in Table B-7 assume that the TRACON controller, Cont_One, is defined, and the airport and TRACON name are provided in the a priori event file.

Table B-7. TRACON Controller Tests, First Phase

Scenario	Required Result
Assume that the controller serving Denver, Approach will be used. Now, Cont_One is defined as the following: CONT_NAME AIRPORT POSITION Cont_One Denver Approach Cont_One Denver Departure Cont_One Denver Final	Statistics collected for TRACON controller Cont_One.
Assume that the controller serving Denver, Approach will be used; but Cont_One is defined as the following: CONT_NAME AIRPORT POSITION Cont_One Denver Departure Cont_One Denver Final	Error: There is no TRACON controller for position Approach.

Table B-8 contains the second phase of the TRACON controller test plan. The scenarios in Table B-8 assume that the TRACON controller is defined; airport and TRACON name are provided in the a priori event file. The TRACON controller named Cont_One will serve all three positions for the given airport.

Table B-8. TRACON Controller Tests, Second Phase

Scenario	Required result
1 aircraft 1 TRACON controller 1 TAKE_OFF Event aircraft1, Cont_One	Statistics collected for aircraft1 and Cont_One.
2 aircraft 1 TRACON controller 2 TAKE_OFF Events (different time) aircraft1: Cont_One aircraft2: Cont_One	Statistics collected for aircraft1, aircraft2, and Cont_One; events are queued up for Cont_One.
2 aircraft 1 TRACON controller 2 TAKE_OFF Events (same time) aircraft1: Cont_One aircraft2: Cont_One	Statistics collected for aircraft1, aircraft2, and Cont_One; events queue up for Cont_One.
1 aircraft 1 TRACON controller 1 sector 1 TAKE_OFF Event aircraft1, Cont_One, sector1	Statistics collected for aircraft1, Cont_One, and sector1.
1 aircraft 1 TRACON controller 1 sector 1 AOC 1 TAKE_OFF Event aircraft1, Cont_One, sector1, AOC1	Statistics collected for aircraft1, Cont_One, sector1, and AOC1.
1 aircraft 1 TRACON controller 1 sector 1 AOC 1 tower 1 TAKE_OFF Event aircraft1, Cont_One, sector1, AOC1, tower1	Statistics collected for aircraft1, Cont_One, sector1, AOC1, and tower1.

COMMUNICATIONS CHANNEL

All communications channels are defined in dictionary files. If any a priori event involves two different aircraft, then the channel must be defined in the a priori event file. The value of channel must be a real number greater than zero. Table B-9 shows the tests for communications channels.

Table B-9. Communications Channels Tests

Scenario	Required result
A channel is requested by an event Event1: channel 1	Statistics collected for channel 1.
A channel is requested by multiple events Event1: channel 1 . . . $Event_n$: channel 1	Statistics collected for channel 1; events are queued up for channel 1.
An event involves communicating via radio 2 in a sector (the sector has 2 radios); but that sector has only 1 channel assigned through the dictionary file.	Error: channel not found.

File Processing

TYPE FILES

Each type file contains the type of the simulation object and the load file name associates with it. The scenarios below involved the aircraft type. But, same test scenarios have been applied to other simulation object types, such as, sector, AOC, airport controllers, and TRACON controllers. Table B-10 lists the tests for the type files.

Table B-10. Type File Tests

Scenario	Expected result
File is in correct format	No error.
The load file name is not specified	Error: Load file is expected.
The load file name is specified and cannot be located or opened	Error: File can not be opened.

DICTIONARY FILES

Sector Dictionary File

Sector dictionary file contains sector IDs, sector types, and 10 channels associate with each sector. The maximum number of channel is 10. Table B-11 contains the sector dictionary file tests.

Table B-11. Sector Dictionary File Tests

Scenario	Required result
Sector type is not defined	Error: Sector type does not exist.
No channel is specified	Error: At least one channel must be specified.
Sector ID is not an integer	Error: Sector ID must be integer.
Missing channel columns	Error: Channel value is expected at column #.
Extra channel columns	Error: Extra channel value.

AOC Dictionary File

The AOC dictionary file contains AOC name, AOC types, and 10 communications devices associated with each AOC. Table B-12 contains the AOC dictionary file tests.

Table B-12. AOC Dictionary File Tests

Scenario	Required result
AOC type is not defined	Error: AOC type does not exist.
No channel is specified	Error: At least one channel must be specified.
Missing channel columns	Error: Channel value is expected at column #.
Extra channel columns	Error: Extra channel value.

Airport Dictionary File

The airport dictionary file contains airport name, controller name, controller type, and 10 communications devices associated with each controller. Table B-13 contains the airport dictionary file tests.

Table B-13. Airport Dictionary File Tests

Scenario	Required result
Airport controller type is not defined	Error: Airport controller type does not exist.
No channel is specified	Error: At least one channel must be specified.
Missing channel columns	Error: Channel value is expected at column #.
Extra channel columns	Error: Extra channel value.

TRACON Dictionary file

The TRACON dictionary file contains the TRACON name, controller name, controller type, and 10 communications devices associated with each TRACON controller. Table B-14 contains the TRACON dictionary file test plan.

Table B-14. TRACON Dictionary File Tests

Scenario	Required result
TRACON controller type is not defined	Error: TRACON controller type does not exist.
TRACON controller name in the [AIRPORT_POSITION] is not defined	Error: TRACON controller name does not exist.
No channel is specified	Error: At least one channel must be specified.
Missing channel columns	Error: Channel value is expected at column #.
Extra channel columns	Error: Extra channel value.
Position is not approach, departure, or final	Error: Position name must be approach, departure, or final.

LOAD FILE

Each load file contains the number of personnel (controllers/dispatchers/pilots), communications devices, and equipment. It also contains task and communications channel loads and equipment system utilization associated with each activity. Table B-15 contains the tests for the load files.

Table B-15. Load File Tests

Scenario	Required result
Missing NUM_PILOTS (for aircraft load file) or NUM_PILOTS < 1 or NUM_PILOTS > 3	Error: NUM_PILOTS is expected and must be between 1 and 3.
Missing NUM_DISPATCHER(for AOC load file)	Error: NUM_DISPATCHER is expected.
Missing NUM_CONTROLLER (for sector, airport controller, TRACON controller load files)	Error: NUM_CONTROLLER is expected.
NUM_CONTROLLER and NUM_DISPATCHER must be between 1 and 10 (In case of airport, there must be one tower controller)	Error: NUM_CONTROLLER must be between 1 and 10.
Missing NUM_COMMDEVICES or NUM_COMMDEVICES > 10 or NUM_COMMDEVICES < 1	Error: NUM_COMMDEVICES must be between 1 and 10.
Missing NUM_EQUIPMENT or NUM_EQUIPMENT > 10 or NUM_EQUIPMENT < 0	Error: NUM_EQUIPMENT is expected and must be between 0 and 10.
Missing =	Error: = is expected.
Non-integer value for the number of controllers, or communication devices, or equipment	Error: Integer value is expected.
Missing tag [LOAD]	Error: Tag [LOAD] is expected.
Missing tag [LOAD_END]	Error: Tag [LOAD_END] is expected.
Extra load column	Error: Extra load column is found at line # column #.
Missing load column	Error: Load value is expected at line # column #.

SCENARIO FILE

The scenario file relates the actual input file names to the pseudonyms used by FAM 2.0. Table B-16 contains the test plan for the scenario file. In Table B-16 the order of each tag is not relevant. However, under each tag, parameters must be specified exactly as shown in the user document.

Table B-16. Scenario File Tests

Scenario	Required result
Missing $DATA	Error: $DATA is expected.
Missing =	Error: = is expected.
Missing $WORKING	Error: $WORKING is expected.
Missing stat_start	Error: stat_start is expected.
Missing simulation_end	Error: simulation_end is expected.
Simulation_start < simulation_end	No error.
stat_start >= simulation_end	Error: Value of stat_start must be < simulation_end time.
Missing a priori_file	Error: a priori_file is expected.
Missing random_mode	Error: random_mode is expected.
Random_mode <> “TRUE” and random_mode <> “FALSE”	Error: random_mode must be TRUE or FALSE.
Missing reuse_seed	Error: reuse_seed is expected.
Reuse_seed <> “TRUE” and reuse_seed <> “FALSE	Error: reuse_seed must be TRUE or FALSE.
Missing random_file	Error: random_file is expected.
Missing min_inter_random_time	Error: min_inter_random_time is expected.
Missing max_inter_random_time	Error: max_inter_random_time is expected.
Min_inter_random_time (max_inter_random_time	No error.
Min_inter_random_time > max_inter_random_time	Error: min_inter_random_time must be (max_inter_random_time).
Missing event_dictionary	Error: event_dictionary is expected.
Missing aircraft_type	Error: aircraft_type is expected.
Missing sector_type	Error: sector_type is expected.
Missing sector_dictionary	Error: sector_dictionary is expected.
Missing aoc_type	Error: aoc_type is expected.
Missing aoc_dictionary	Error: aoc_dictionary is expected.
Missing airport_controller_type	Error: airport_controller_type is expected.
Missing airport_controller_dictionary	Error: airport_controller_dictionary is expected.
Missing tracon_controller_type	Error: tracon_controller_type is expected.
Missing tracon_controller_dictionary	Error: tracon_controller_dictionary is expected.

Other Runtime Errors

FAM 2.0 catches a few other runtime errors caught during file preprocessing before simulation begins that have not been described yet. Table B-17 contains the tests for these errors.

Table B-17. Other Runtime Error Tests

Scenario	Expected result
The ORG or DST keyword in the associated event file (e.g., L_SECT, AOC, departure) does not have corresponding identifier in the event vector of the a priori-event file. (Refer to the user documentation in the section for associated event)	In the case of a sector: Error: There is no sector ID for primary event "primary event name" and associated event "associated event name". You must specify the Sector ID when an associated event involves SECT.
Missing LOADS when an associated event is being executed, but corresponding loads are not found in the involved simulation objects	Error: There is no load for primary event (e.g., "*SECT_CHG*") and associated event (e.g., "*CALL_AC*"), mode = mode (e.g., "*ORG*") and type = type (e.g.," *SECTOR_A*").
Missing default LOADS when an associated event is being executed; and corresponding loads are found, but the mode and mate-type cannot be matched exactly and there are defaults specified.	Error: There is no default load for primary event (e.g., "SECT_CHG") and associated event (e.g., "CALL_AC"), mode = mode (e.g., "ORG") and type = type (e.g.," SECTOR_A")
The time for the various events in the file is not in ascending order; events are permitted to begin at the same time	Error: DLY time must be in ascending order.

VALIDATION AND VERIFICATION TESTING

Figure B-1 details the tests that LMI ran on the finished FAM 2.0 model. Each scenario was created with increasing complexity and scale. No errors were encountered during this testing.

Figure B-1. Validation and Verification Test Scenarios

FAM Test Scenario Objectives

Scenario #	Aircraft	ARTCC	Tower	TRACON	AOC	Notes
1	2	2	-	-	-	Sector Change
2	1	1	1	1	1	Departure from Airport
3	1	1	1	1	-	Arrival at Airport
4	4	1	2	2	1	Combination of scenarios 1-4: 1. take-off :: exit 2. take-off :: land 3. enter :: land 4. enter :: exit
5	4	7	2	2	1	Extension of scenario 4, including Sector Change events
6	50		2	2	1	Extension of scenario 5, with mulitple aircraft

Appendix C
FAM 2.0 ModSim III Code Listings

The code listings for the FAM 2.0 program modules run to over 100 pages. Because of their limited use and volume, they are published separately. The paper document and the electronic files are available from LMI. An individual or organization that does not have access to an HP-UNIX platform but does have a ModSim III site license can read the electronic files, which are simple text files, and create their own model running on their platform.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	April 1998	Contractor Report

4. TITLE AND SUBTITLE
Aircraft/Air Traffic Management Functional Analysis Model, Version 2.0, Technical Description

5. FUNDING NUMBERS
C NAS2-14361
Task 97-03

WU 538-04-14-02

6. AUTHOR(S)
Melvin Etheridge, Joana Plugge, Nusrat Retina

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Logisitics Management Institute
2000 Corporation Ridge
McLean, Virginia 22102-7805

8. PERFORMING ORGANIZATION REPORT NUMBER
NS703S2

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Langley Research Center
Hampton, VA 23681-0001

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
NASA/CR-1998-207657

11. SUPPLEMENTARY NOTES
Langley Technical Monitor: Robert E. Yackovetsky
Final Report

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Unclassified - Unliited

Subject Category 01 Distribution: Nonstandard

Availability: CASI (301) 621-0390

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 words)***

The Aircraft/Air Traffic Management Functional Analysis Model, Version 2.0 (FAM 2.0), is a discrete event simulation model designed to support analysis of alternative concepts in air traffic management and control. FAM 2.0 was developed by the Logistics Management Institute (LMI) under task order NS703 of the National Aeronautics and Space Administration (NASA) contract number NAS2-14361. This document provides a technical description of FAM 2.0 and its computer files to enable the modeler and programmer to make enhancements or modifications to the model. Those interested in a guide for using the model in analysis should consult the companion document, Aircraft/Air Traffic Management Functional Analysis Model, Version 2.0 Users Manual.

14. SUBJECT TERMS
airport
air traffic management
air transportation

15. NUMBER OF PAGES
70

16. PRICE CODE
A04

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unlimited