GEORGIA INSTITUTE OF TECHNOLOGY — OFFICE OF CONTRACT ADMINISTRATION

PROJECT ADMINISTRATION DATA SHEET

[X] ORIGINAL [ ] REVISION NO. _____

Project No. G-36-633 (R6039-0A0) GTRC/GIT DATE 10 / 9 / 85

Project Director: A. P. Jensen (Bill Penny) School/~~Lab~~ ICS

Sponsor: U. S. Army Information Systems Selection & Acquisition Activity
Ft. Belvoir, VA

Type Agreement: Contract DAHC06-85-C-0012

Award Period: From 9/13/85 To 3/12/86 (Performance) 3/12/86 (Reports)

Sponsor Amount:	This Change	Total to Date
Estimated:	$ 173,284	$ 173,284
Funded:	$ 173,284	$ 173,284

Cost Sharing Amount: $ None Cost Sharing No: N/A

Title: Combat Service Support Systems Advanced Experimental Demonstrations (AED)

ADMINISTRATIVE DATA — OCA Contact William F. Brown X4820

1) Sponsor Technical Contact:

Glenn Racine
AIRMICS
115 O'Keefe Building
Georgia Institute of Technology
Atlanta, GA 30332-0800
894-3107

2) Sponsor Admin/Contractual Matters:

Celestine Holly
U. S. Army Information Systems
Selection and Acquisition Activity
Attn: ASW-MSC; Stop C-40
Ft. Belvoir, VA 22060-5456
(703) 664-6311

Defense Priority Rating: A70 Military Security Classification: N/A
(or) Company/Industrial Proprietary: N/A

RESTRICTIONS

See Attached Gov't Supplemental Information Sheet for Additional Requirements.

Travel: Foreign travel must have prior approval – Contact OCA in each case. Domestic travel requires sponsor approval where total will exceed greater of $500 or 125% of approved proposal budget category.

Equipment: Title vests with Gov't

COPIES TO:

SPONSOR'S I. D. NO. 02.102.017.86.001

Project Director
Research Administrative Network
Research Property Management
Accounting
Procurement/GTRI Supply Services
Research Security Services
Reports Coordinator (OCA)
Research Communications (2)
GTRC
Library
Project File
Other A. Jones

GEORGIA INSTITUTE OF TECHNOLOGY OFFICE OF CONTRACT ADMINISTRATION

SPONSORED PROJECT TERMINATION/CLOSEOUT SHEET

Date 7-13-87

Project No. G-36-633 School/[redacted] ICS

Includes Subproject No.(s) N/A

Project Director(s) A.P. Jensen GTRC / [redacted]

Sponsor U.S. Army Information Systems Selection & Acquisition Activity

Title Combat Service Support Systems Advanced Experimental Demonstrations (AED)

Effective Completion Date: 6/12/86 (Performance) 6/12/86 (Reports)

Grant/Contract Closeout Actions Remaining:

- [ ] None
- [X] Final Invoice or Final Fiscal Report
- [X] Closing Documents
- [X] Final Report of Inventions
- [X] Govt. Property Inventory & Related Certificate
- [ ] Classified Material Certificate
- [ ] Other _____

Continues Project No. _____ Continued by Project No. _____

COPIES TO:

Project Director
Research Administrative Network
Research Property Management
Accounting
Procurement/GTRI Supply Services
Research Security Services
[redacted]

Library
GTRC
[redacted]
Project File
Other Duane H.
Angela DuBose
Russ Embry

FORM OCA 69.285

Performance and Cost Report

Contract Number DAHC06-85-C-0012

Project Number G-36-633

Georgia Institute of Technology

September 13, 1985 to September 30, 1985

Overview

The project group continued to maintain and expand the TACCNET prototype system. Improved login handling and broadcast message handling are under development. Error logging and message tracking systems have been improved. Other planned improvements to TACCNET include: screen editing capability for JINTACCS messages, console monitoring facilities for network observation, message display utilities for demonstrations, robust login procedures, and administrative utilities.

The Honeywell GCOS C compiler has been ordered and is expected to arrive in October. Preparations are underway to transfer all TACCNET source code from the IBM PC/AT to the Honeywell DPS/6. Substantial changes will be required in the I/O routines due to differences between UNIX and GCOS.

The database backup and recovery system is being designed. It is expected to involve a combination of shell scripts and C functions. Modifications to the MSGPROC system will be required.

Meetings and Presentations

Regular meetings were held to exchange information and present results. No special meetings or presentations were held.

Plans

- Obtain and install the GCOS C compiler.
- Transfer TACCNET source code to the Honeywell DPS/6 and port to GCOS
- Design a database backup and recovery system for TACCNET.
- Develop a screen oriented editing interface for the JINTACCS message composition tool.

Personnel

Alton P. Jensen.....project administration
William Putnam.....project management and design
Steven Goldberg.....system design and development
Shinn Hong S.....research assistant

There are two professional researchers involved with the project. Bill Putnam works full time as project manager and chief software designer. Alton P. Jensen works one-quarter time as project administrator.

There are two students working part-time on this project. They carry work loads ranging from 12 to 20 hours per week depending on availability.

The hours spent on the project are classified and allocated into five categories. A breakdown of labor for the period is given below.

Category	Hours
Administration	70
Planning & Design	100
Prototyping & Testing	40
Programming & Debugging	50
Documentation & Reporting	20
Total	470

Expenses

Expenses for the period were solely in the category of salaries. Salaries of professional research personnel are classified as Professional Expenses. Salaries of student assistants are classified as Staff Expenses. A breakdown is given below.

Professional Expenses	$ 4,500.00
Staff Expenses	$ 400.00
Total Expenses	$4,900.00

Work Completed

Item	% Complete
C2 Database Backup and Recovery	
System design	25
Initial code development	0
Remote backup system	0
Database recovery	0
Archive search & retrieval	0
Alternate database site operations	0
Testing and integration	0
Work for Task 1	5
DPS 6 PASCAL to C Conversion	
Source code transfer	10
IOControl system	0
Caller	0
GenMsg	0
Integration with TACCNET	0
Work for Task 2	5
Technical Specification	0

G-36-633

Performance and Cost Report

Contract Number DAHC06-85-C-0012

Project Number G-36-633

Georgia Institute of Technology

October 1, 1985 to October 31, 1985

Overview

A task plan for the project is being prepared for presentation to AIRMICS. Work areas will include UNIX/GCOS conversion of TACCNET, JINTACCS message preparation facilities, C2 database backup and recovery, and TACCNET user interface development.

The database backup and recovery system is being designed. It is expected to involve a combination of shell scripts and C functions. Modifications to the MSGPROC system will be required.

A TACCS computer has been received by AIRMICS. It is being installed and will be brought up under UNIX if possible. Plans are to port TACCNET to the new system as soon as possible.

An advance copy of the Honeywell GCOS C Compiler arrived but was not accompanied by any documentation. Reiko Taylor is working on getting manuals and on expediting our order.

Meetings and Presentations

Regular meetings were held to exchange information and present results. In addition, planning sessions were called as needed.

Plans

- Install the GCOS C compiler. Obtain manuals and evaluate the compiler. Port TACCNET to the Honeywell under GCOS.

- Complete the design of the database backup and recovery system for TACCNET.

- Develop a screen oriented editing interface for the JINTACCS message composition tool.

- Install the TACCS and bring up UNIX. Port TACCNET to the TACCS.

Personnel

Alton P. Jensen.....project administration
William Putnam.....project management and design
Steven Goldberg.....system design and development
Shinn Hong S.....research assistant

There are two professional researchers involved with the project. Bill Putnam works full time as project manager and chief software designer. Alton P. Jensen works one-quarter time as project administrator.

There are two students working part-time on this project. They carry work loads ranging from 12 to 20 hours per week depending on availability.

The hours spent on the project are classified and allocated into five categories. A breakdown of labor for the period is given below.

Category	Hours
Administration	70
Planning & Design	150
Prototyping & Testing	160
Programming & Debugging	60
Documentation & Reporting	40
Total	480

Expenses

Expenses for the period were solely in the category of salaries. Salaries of professional research personnel are classified as Professional Expenses. Salaries of student assistants are classified as Staff Expenses. A breakdown is given below.

Professional Expenses	$ 4,500.00
Staff Expenses	$ 2,800.00
Total Expenses	$7,300.00

Work Completed

Item	% Complete
C2 Database Backup and Recovery	
System design	50
Initial code development	0
Remote backup system	0
Database recovery	0
Archive search & retrieval	0
Alternate database site operations	0
Testing and integration	0
Work for Task 1	7
DPS 6 PASCAL to C Conversion	
Source code transfer	50
IOControl system	10
Caller	0
GenMsg	0
Integration with TACCNET	0
Work for Task 2	10
Technical Specification	0

G-36-633

Performance and Cost Report

Contract Number DAHC06-85-C-0012

Project Number G-36-633

Georgia Institute of Technology

November 1, 1985 to November 30, 1985

Overview

A task plan for the project was prepared and presented to AIRMICS. Three work areas were identified: TACCNET improvements, Honeywell C conversion, and database backup and recovery. The conversion of Honeywell DPS/6 code from PASCAL to C will begin as soon as the C compiler is received and installed. The other work was begun immediately.

The planned improvements to TACCNET include: screen editing capability for JINTACCS messages, console monitoring facilities for network observation, message display utilities for demonstrations, robust login procedures, and administrative utilities. These improvements are primarily of a cosmetic nature and will improve the clarity and impact of TACCNET demonstrations.

While the DAS3 described in the CSSCS environment is not required to receive JINTACCS messages from the TACCS at this time, it is expected that that capability can be provided during the C code conversion. Some groundwork will be required so that existing C code from the UNIX based TACCS environment can be transferred to the Honeywell DPS/6 and retrofitted to run under GCOS. Numerous changes will be required due to the disparate nature of the GCOS and UNIX operating systems, but the interfaces and operations of the programs will be the same under both systems.

The database backup and recovery system is being designed. It is expected to involve a combination of shell scripts and C functions. Modifications to the MSGPROC system will be required.

A TACCS computer has been received by AIRMICS. It is supplied with the BTOS and DISTRIX 1.0 operating systems. An

examination of DISTRIX 1.0 uncovered or confirmed numerous deficiencies which have been reported. DISTRIX 2.0 is reported to fix some of the bugs. Jim Kearns is working on getting a pre release copy.

An advance copy of the Honeywell GCOS C Compiler arrived but was not accompanied by any documentation. Reiko Taylor is working on getting manuals and on expediting our order.

Meetings and Presentations

Regular meetings were held to exchange information and present results. In addition, planning sessions were called as needed.

A presentation based on the final report of the previous contract (g-36-610) was given. In addition, a briefing on TACCNET has been scheduled for December 18.

Plans

- Install the GCOS C compiler. Obtain manuals and evaluate the compiler.

- Complete the design of the database backup and recovery system for TACCNET.

- Develop a screen oriented editing interface for the JINTACCS message composition tool.

- Prepare a briefing in TACCNET for presentation in December.

Personnel

Alton P. Jensen.....project administration
William Putnam.....project management and design
Steven Goldberg.....system design and development
Shinn Hong S.....research assistant

There are two professional researchers involved with the project. Bill Putnam works full time as project manager and chief software designer. Alton P. Jensen works one-quarter time as project administrator.

There are two students working part-time on this project. They carry work loads ranging from 12 to 20 hours per week depending on availability.

The hours spent on the project are classified and allocated into five categories. A breakdown of labor for the period is given below.

Category	Hours
Administration	70
Planning & Design	100
Prototyping & Testing	160
Programming & Debugging	100
Documentation & Reporting	40
Total	470

Expenses

Expenses for the period were solely in the category of salaries. Salaries of professional research personnel are classified as Professional Expenses. Salaries of student assistants are classified as Staff Expenses. A breakdown is given below.

Professional Expenses	$ 4,500.00
Staff Expenses	$ 2,600.00
Total Expenses	$7,100.00

Work Completed

Item	% Complete
C2 Database Backup and Recovery	
System design	75
Initial code development	0
Remote backup system	0
Database recovery	0
Archive search & retrieval	0
Alternate database site operations	0
Testing and integration	0
Work for Task 1	10
DPS 6 PASCAL to C Conversion	
Source code transfer	80
IOControl system	10
Caller	0
GenMsg	0
Integration with TACCNET	0
Work for Task 2	15
Technical Specification	0

G-36-633

Performance and Cost Report

Contract Number DAHC06-85-C-0012

Project Number G-36-633

Georgia Institute of Technology

December 1, 1985 to December 31, 1985

Meetings and Presentations

Regular meetings were held to exchange information and present results. In addition, planning sessions were called as needed.

The briefing scheduled for Dec. 18 was canceled. It will be rescheduled at a later date. Bill Putnam will work on preparing a high level presentation on TACCNET to supplement the existing technical briefing.

Plans

- Complete coding for database backup and recovery system.
- Resolve problem with Honeywell C compiler program linking.
- Test and debug GCOS implementation of TACCNET.
- Implement message editor for JINTACCS composition tool.

Overview

Documentation for the Honeywell C compiler was received in December. Work began on transferring the UNIX-based C code for TACCNET from the PC/AT to the DPS/6. The transfer was completed without incident using simple terminal emulation and file transfer programs. Initial attempts to compile the C code were moderately successful with most routines compiled on the first try. Some compiler differences were encountered and corrections made, resulting in the successful compilation of approximately 95% of the C code.

A problem emerged when we attempted to link the compiled code into an executable program. The linker for the C compiler contains a bug which prevents programs larger than a few Kbytes from linking. We have a call in to Honeywell and are contacting them about a fix. In the mean time, we will begin making necessary changes that we already know about in the device dependent parts of the system.

Work proceeded on the database backup and recovery system for TACCNET. The system will allow the C2 database files (used by UNIFY) to be transferred via TACCNET to a designated backup site for safekeeping. The files will be stored in a special "backup" directory on the remote machine and can be retrieved automatically by sending a special network administrative message.

The user will be insulated from the actual administrative messages required to back up and recover the files by a set of shell scripts which will be invoked as command files.

Personnel

Alton P. Jensen.....project administration
William Putnam.....project management and design
Steven Goldberg.....system design and development
Shinn Hong S.....research assistant

There are two professional researchers involved with the project. Bill Putnam works full time as project manager and chief software designer. Alton P. Jensen works one-quarter time as project administrator.

There are two students working part-time on this project. They carry work loads ranging from 12 to 20 hours per week depending on availability.

The hours spent on the project are classified and allocated into five categories. A breakdown of labor for the period is given below.

Category	Hours
Administration	50
Planning & Design	80
Prototyping & Testing	100
Programming & Debugging	100
Documentation & Reporting	28
Total	358

Expenses

Expenses for the period were solely in the category of salaries. Salaries of professional research personnel are classified as Professional Expenses. Salaries of student assistants are classified as Staff Expenses. A breakdown is given below.

Professional Expenses	$ 4,500.00
Staff Expenses	$ 1,480.00
Total Expenses	$5,980.00

Work Completed

Item	% Complete
C2 Database Backup and Recovery	
System design	80
Initial code development	25
Remote backup system	20
Database recovery	10
Archive search & retrieval	0
Alternate database site operations	50
Testing and integration	0
Work for Task 1	25
DPS 6 PASCAL to C Conversion	
Source code transfer	100
IOControl system	50
Caller	50
GenMsg	0
Integration with TACCNET	0
Work for Task 2	20
Technical Specification	0

Performance and Cost Report

Contract Number DAHC06-85-C-0012

Project Number G-36-633

Georgia Institute of Technology

January 1, 1986 to January 31, 1986

Overview

The C functions and shell scripts for the database backup function have been installed and tested. Work is nearly complete on the recovery function. Code to allow the distribution and handling of TACCNET broadcast messages is in development.

Broadcast messages are those which are sent from one site to all other sites in the network. This type of message will be used to send out requests for database recovery from archived messages. It may also be used to distribute network wide routing information.

Shinn reports progress in the development of the screen editor interface for the JINTACCS message entry system. The system uses the curses screen handling package. This package is supported on almost all UNIX systems with only minor differences among the different UNIX versions.

The interface as planned will prompt the user in a screen oriented (as opposed to line oriented) manner and user the responses to build a JINTACCS format message. The user will be able to examine the message at any point in the composition process.

The CGOS system conversion is proceeding slowly. An apparent bug in the GCOS device handler causes the transmission of the DEL character as a filler in spite of configuration commands that should disable this "feature". The low level I/O routines of the IOCONTROL module are being rewritten for GCOS while the DEL problem and the C Linker problems are being investigated.

A hard disk failure on the "xenair" PC/AT Xenix system has

rendered that system inoperative. It will be repaired ASAP.

The TACCS DISTRIX system is now running TACCNET under DISTRIX 2.0. The port was accomplished without incident, and the system is functioning as a regular TACCNET node. We are unable to run the UNIFY dbms system on the TACCS (different cpu) at this time. It should be noted that there appears to be no way to access the internal modems on the TACCS from DISTRIX. There in no mention of them in the DSITRIX documentation and no description of the UNIX uucp program. TACCNET is running using two external Hayes modems.

Work on the Technical specification is expected to begin as soon as the contents of the spec are defined.

Meetings and Presentations

Regular meetings were held to exchange information and present results. In addition, planning sessions were called as needed.

A meeting was held to discuss the desireed content of the TACCNET Technical Specification required for Task 3. The spec should contain a high level design description, user manual, and an installation guide.

A briefing and demonstration were scheduled for February 19.

Plans

- Complete low level I/O rewrite for TACCNET on DPS 6.
- Continue development of screen oriented JINTACCS composition tool.
- Complete, install, and test database recovery system.
- Arrange for the repair of the PC/AT.
- Correct the GCOS C Linker problem and the DEL problem.
- Develop an outline for TACCNET Technical Specification.

Personnel

Alton P. Jensen.....project administration
William Putnam.....project management and design
Steven Goldberg.....system design and development
Shinn Hong S.....research assistant

There are two professional researchers involved with the project. Bill Putnam works full time as project manager and chief software designer. Alton P. Jensen works one-quarter time as project administrator.

There are two students working part-time on this project. They carry work loads ranging from 12 to 20 hours per week depending on availability.

The hours spent on the project are classified and allocated into five categories. A breakdown of labor for the period is given below.

Category	Hours
Administration	50
Planning & Design	80
Prototyping & Testing	108
Programming & Debugging	100
Documentation & Reporting	28
Total	366

Expenses

Expenses for the period were solely in the category of salaries. Salaries of professional research personnel are classified as Professional Expenses. Salaries of student assistants are classified as Staff Expenses. A breakdown is given below.

Professional Expenses	$ 4,500.00
Staff Expenses	$ 1,560.00
Total Expenses	$6,060.00

Work Completed

Item	% Complete
C2 Database Backup and Recovery	
System design	95
Initial code development	100
Remote backup system	60
Database recovery	30
Archive search & retrieval	10
Alternate database site operations	60
Testing and integration	10
Work for Task 1	45
DPS 6 PASCAL to C Conversion	
Source code transfer	100
IOControl system	80
Caller	75
GenMsg	0
Integration with TACCNET	20
Work for Task 2	30
Technical Specification	5

Performance and Cost Report

Contract Number DAHC06-85-C-0012

Project Number G-36-633

Georgia Institute of Technology

February 1, 1986 to February 28, 1986

Overview

The PC/AT with the hard disk problem was repaired and reinstalled. It is once again operational in TACCNET.

The latest version of TACCNET has been installed on the two PC/AT systems (xenics and xenair), one ONYX system (sysa), and the TACCS (taccs) DISTRIX system. This version contains a robust login procedure, database backup system, broadcast messages, and improved error logging.

The JINTACCS message compostion tool is nearing completion. It now allows the user two operate in command or editing mode and two toggle between these modes at will. Input syntax checking and a help facility are under development.

Meetings and Presentations

Regular meetings were held to exchange information and present results. In addition, planning sessions were called as needed.

A demonstration was presented on Feb. 26. Test scenario 3 was executed successfully. A request was made for better console monitoring utilities.

We are having some trouble getting customer support for the Honeywell C compiler. Our main contact is Jim Hughes here in Atlanta, but he is only moderately helpful. The questions are slowly being resolved.

Plans

- Continue PASCAL to C conversion on GCOS system.
- Complete database backup and recovery system - installation and testing phase.
- Complete JINTACCS compostion tool and integrate into demonstration.
- Develop a better monitoring interface for demonstrations.

Personnel

Alton P. Jensen.....project administration
William Putnam.....project management and design
Steven Goldberg.....system design and development
Shinn Hong S.....research assistant

There are two professional researchers involved with the project. Bill Putnam works full time as project manager and chief software designer. Alton P. Jensen works one-quarter time as project administrator.

There are two students working part-time on this project. They carry work loads ranging from 12 to 20 hours per week depending on availability.

The hours spent on the project are classified and allocated into five categories. A breakdown of labor for the period is given below.

Category	Hours
Administration	50
Planning & Design	80
Prototyping & Testing	100
Programming & Debugging	120
Documentation & Reporting	20
Total	370

Expenses

Expenses for the period were solely in the category of salaries. Salaries of professional research personnel are classified as Professional Expenses. Salaries of student assistants are classified as Staff Expenses. A breakdown is given below.

Professional Expenses	$ 4,500.00
Staff Expenses	$ 1,600.00
Total Expenses	$6,100.00

Work Completed

Item	% Complete
C2 Database Backup and Recovery	
System design	95
Initial code development	100
Remote backup system	95
Database recovery	60
Archive search & retrieval	50
Alternate database site operations	70
Testing and integration	30
Work for Task 1	55
DPS 6 PASCAL to C Conversion	
Source code transfer	100
IOControl system	80
Caller	80
GenMsg	5
Integration with TACCNET	30
Work for Task 2	45
Technical Specification	5

G-36-633

Performance and Cost Report

Contract Number DAHC06-85-C-0012

Project Number G-36-633

Georgia Institute of Technology

March 1, 1986 to March 31, 1986

Overview

Work continued on the Honeywell PASCAL to C conversion. Modification of low level I/O routines is complete and testing has begun. The next phase will involve the conversion of directory and file handling routines. Problems with the port configuration and the modems have hampered progress, but have been resolved.

The C program linker for GCOS still refuses to link moderate to large programs. Help from Honeywell has produced a "workaround" solution which requires the user to interrupt the compilation and manually edit the linker control file to set certain link parameters before restarting the link process. This allows us to link the programs, but is a severe inconvenience and must be regarded as a temporary fix.

The database backup and recovery functions are complete and are being tested. The incremental database recovery from archives is being coded and should be complete in April. A scenario will be devised to demonstrate the system.

Coding for the JINTACCS message composition tool is complete. Shinn is debugging the program and making some cosmetic changes.

Meetings and Presentations

Regular meetings were held to exchange information and present results. In addition, planning sessions were called as needed.

Plans

- Test and debug database backup and recovery system.

- Test and debug JINTACCS message composition tool.

- Implement incremental database recovery from archives.

- Rewrite file and directory handling routines for GCOS TACCNET.

- Test GCOS C low level I/O routines in TACCNET.

Personnel

Alton P. Jensen.....project administration
William Putnam.....project management and design
Steven Goldberg.....system design and development
Shinn Hong S.....research assistant

There are two professional researchers involved with the project. Bill Putnam works full time as project manager and chief software designer. Alton P. Jensen works one-quarter time as project administrator.

There are two students working part-time on this project. They carry work loads ranging from 12 to 20 hours per week depending on availability.

The hours spent on the project are classified and allocated into five categories. A breakdown of labor for the period is given below.

Category	Hours
Administration	50
Planning & Design	50
Prototyping & Testing	110
Programming & Debugging	188
Documentation & Reporting	20
Total	418

Expenses

Expenses for the period were solely in the category of salaries. Salaries of professional research personnel are classified as Professional Expenses. Salaries of student assistants are classified as Staff Expenses. A breakdown is given below.

Professional Expenses	$ 4,500.00
Staff Expenses	$ 2,080.00
Total Expenses	$6,580.00

Work Completed

Item	% Complete
C2 Database Backup and Recovery	
System design	100
Initial code development	100
Remote backup system	100
Database recovery	90
Archive search & retrieval	75
Alternate database site operations	80
Testing and integration	60
Work for Task 1	80
DPS 6 PASCAL to C Conversion	
Source code transfer	100
IOControl system	95
Caller	80
GenMsg	5
Integration with TACCNET	40
Work for Task 2	55
Technical Specification	5

Performance and Cost Report

Contract Number DAHC06-85-C-0012

Project Number G-36-633

Georgia Institute of Technology

April 1, 1986 to April 30, 1986

Overview

Development proceeds on the message processor routines to recover JINTACCS messages from site archives. These messages will be re-queued for transmission to a specified site for used in reconstituting that site's C2 database. Functions for the regular backup and recovery of the C2 database are installed and have been validated by testing.

A draft outline for the TACCNET technical specification has been prepared. The spec is supposed to conform to MIL SPEC 490. It was discovered that MIL SPEC 490 has been superceded by 490A. MIL SPEC 490A references documents not available at airmics. These documents will be located and provided asap.

We are attempting to obtain a copy of the JINTACCS handbook, which would allow us to put more messages into the system. Glenn will try to get the book for us.

We have available a Kurzweil Voice System machine which can do voice recognition with a 1000 word maximum vocabulary. We are considering ways to incorporate this machine into the research using voice recognition software to build JINTACCS messages.

Steve reports that the Honeywell does not allow timeouts during read operations. Some rewriting of the code is required to allow for this. Other problems related to the configuration of ports have been recorded. The system will require some modification of UNIX based code to handle special timeout, DEL, and BEL situations arising from GCOS limitations on asynchronous reads and writes. These changes will be bracketed by "#ifdef" statements in the C code so that they will only be compiled as needed.

Meetings and Presentations

Regular meetings were held to exchange information and present results. In addition, planning sessions were called as needed.

A meeting was held to discuss the content of the Technical Specification. There is some uncertainty as to the intended use of the specification. More information will be obtained.

We have become aware of a software product developed by Consultant's Choice, Inc. which is purported to be able to read and parse JINTACCS messages. Arrangements are being made to investigate the product and evaluate its suitability for use in the CSSCS environment. A presentation by CCI representatives was held at the AIRMICS office. A decision was made to examine the system further and to solicit sources for JINTACCS message processing systems for open evaluation.

Plans

- Complete the C2 database archive recovery functions.
- Make arrangements to evaluate available JINTACCS message processing systems.
- Develop a software test plan for JINTACCS message processing systems.
- Develop and install a screen oriented interactive monitor for

taccnet.

- Install and test the JINTACCS message composition tool.

Personnel

Alton P. Jensen.....project administration
William Putnam.....project management and design
Steven Goldberg.....system design and development
Shinn Hong S.....research assistant

There are two professional researchers involved with the project. Bill Putnam works full time as project manager and chief software designer. Alton P. Jensen works one-quarter time as project administrator.

There are two students working part-time on this project. They carry work loads ranging from 12 to 20 hours per week depending on availability.

The hours spent on the project are classified and allocated into five categories. A breakdown of labor for the period is given below.

Category	Hours
Administration	50
Planning & Design	50
Prototyping & Testing	120
Programming & Debugging	200
Documentation & Reporting	40
Total	460

Expenses

Expenses for the period were solely in the category of salaries. Salaries of professional research personnel are classified as Professional Expenses. Salaries of student assistants are classified as Staff Expenses. A breakdown is given below.

Professional Expenses	$ 4,500.00
Staff Expenses	$ 2,500.00
Total Expenses	$7,000.00

Work Completed

Item	% Complete
C2 Database Backup and Recovery	
System design	100
Initial code development	100
Remote backup system	100
Database recovery	100
Archive search & retrieval	75
Alternate database site operations	80
Testing and integration	70
Work for Task 1	85
DPS 6 PASCAL to C Conversion	
Source code transfer	100
IOControl system	95
Caller	80
GenMsg	5
Integration with TACCNET	55
Work for Task 2	65
Technical Specification	10

Considerations in the Design and Development of a Combat Service Support Computer System

by

Principal Investigator:
Alton P. Jensen, Professor

Project Manager:
William O. Putnam, Research Scientist II

Project Staff:
Steven L. Goldberg, Research Assistant
Hong S. Shinn, Graduate Research Assistant

School Of Information and Computer Science
Georgia Institute Of Technology
Atlanta, Georgia 30332

Presented to:

U.S. Army Institute for Research In Management
Information, Communications, and Computer Science
(AIRMICS)

December 19, 1986

Contract No. DAHC0-85-C00012
Research Project No. G36-633

Table of Contents

1. Introduction

This document has been prepared as an examination of the issues surrounding the development and implementation of the Combat Service Support Control System (CSSCS) and the Command and Control (C^2) Database. In order to fully explore these issues, prototype CSSCS and C^2 Database systems were designed and implemented. Descriptions of these prototypes are included here, along with instructions for installation and operation of the prototype systems.

The project began with the results of previous work undertaken to study the issues of communications and message passing in the environment of the CSSCS. These results were expanded upon and incorporated into the development of the prototype CSSCS system. In addition to message passing and failure detection, the prototype system addresses the issues of automatic message routing, dynamic network reconfiguration, remote node identification, network security, message generation and processing, C^2 database interactions, and database backup/recovery.

The CSSCS environment is described as a loosely coupled *occasionally connected* network of independently operating computers. By *occasionally connected*, we mean that nodes in the network may come and go at will under normal operating conditions. In addition, the CSSCS environment is conducive to catastrophic node failures of network nodes. The necessity of mobility and risk of destruction are unavoidable features of the battlefield environment in which the CSSCS will operate. Node failures must be detected and handled by the system with a minimum of human intervention.

Development of the prototype system was done under the UNIX† operating system. UNIX was chosen because of its wide availability in the class of small machines being considered for use by the Army as the TACCS. UNIX provides a full featured multi-tasking, multi-user environment conducive to the development of software. It is well supported in a variety of hardware environments including personal computers, minicomputers, and mainframes. The C programming language, standard on all UNIX systems, is a powerful, high level language incorporating structured programming features, flexible data structures of almost any desired level of complexity, and systems programming features for simple access to UNIX or to other processes.

Because the CSSCS environment requires real-time response to changing conditions and due to the unpredictable nature of communications in that environment, a robust, flexible, timesharing system is desirable. The UNIX operating system provides these qualities and offers a greater degree of portability among

† UNIX is a trademark of AT&T Bell Laboratories.

different types and sizes of machines than any other system currently available.

The reader of this report is not expected to be a UNIX expert, but is expected to be familiar with basic UNIX system concepts including files, directories, access permissions, pipes, I/O redirection, shells, shell commands, text editors, and login procedures. Some aspects of the prototype TACCNET system require the editing of configuration files, including the system password file. It is recommended that the reader study the UNIX system administration manuals carefully before modifying any system files.

2. System Overview

The CSSCS is intended to perform a prescribed set of functions in a designated environment. The details of these functions and the nature of this environment impose certain constraints on the system. In this section we will define the functions to be performed by the CSSCS and describe the CSSCS environment.

The CSSCS is intended to support the CSS commander on the battlefield at the corps, division, and brigade levels. The term "CSS commander" is refers to the officer responsible for managing the CSS function at any particular site. CSS is defined as the functional areas of supply, maintenance, field services, transportation, personnel, and health services found in the divisions, corps, and theater Army.

2.1. CSSCS Functions

The CSSCS is an information system for the Combat Service Support (CSS) node of the Army Command and Control System (ACCS). As such, the CSSCS will interface will the other four nodes of the ACCS as well as with other functional systems within CSS (e.g. STAMMIS). The interface to external nodes will be via JINTACCS messages, both sent and received. The interface to STAMMIS will be limited to the extraction of data which will be posted to the Command and Control (C^2) database which will be an integral part of the CSSCS.

The basic functions of the CSSCS are:

- to provide a transport and communications network for information exchange among CSS units, primarily in the form of JINTACCS messages;
- to provide a database of information for use by CSS commanders and personnel in the performance of CSS functions;
- provide decision support functions to the CSS commanders on the battlefield.

2.2. CSSCS Environment

The CSSCS will operate in the battle field environment of the modern Army. This requires mobility and portability of all systems as well as transparency with respect to communications media. CSS units will appear as nodes in a loosely connected network capable of frequently changing topology. Nodes may join and leave the network at will as they change locations in the battlefield environment. Nodes are also subject to catastrophic failure due to enemy activity.

These elements of the environment require a network which is able to detect arrivals, departures, and failures and adjust operations accordingly.

The CSSCS must be able to detect errors in the routing or delivery of a message and reroute the message as necessary to ensure timely delivery to an appropriate CSS unit. In the event that a CSS node is rendered inoperative, it will be necessary to recover its C^2 database from a backup (at another node) and reconstitute the database by collecting all messages sent to the node after the backup and before the failure. It may be necessary for one CSS node to perform the function of a down node, taking its place in the network and carrying out the function of the down node until that node can be replaced.

The modern battlefield will offer a variety of communications media including existing telephone networks, microwave links, optical links, packet radio, and other more traditional media. Since it may not be possible to determine in advance the media available for CSSCS data transfer, it is desirable to have a system which is independent of communications medium. Limited bandwidth for digital communications encourages reduction of data redundancy in message formats and message redundancy in reporting systems.

The CSSCS environment places the following constraints on the CSSCS:

- restricted bandwidth for communications
- media transparent communications
- nodes join and depart the network at will
- nodes subject to catastrophic failure
- must provide distributed backup and recovery of C^2 databases
- must automatically route messages for timely delivery
- must detect failures and reroute messages accordingly
- messages in JINTACCS format

The system must observe these constraints and carry out its functions with a minimum of operator intervention.

3. Design Issues

In this section the major CSSCS design issues and decisions identified during the development of the TACCNET system will be discussed. These issues are divided into three categories. High level issues are those which concern the entire network and its functions. Middle level issues involve the various TACCNET components, their interfaces, and their modes of operation. The low level design issues involve operating system interfaces, hardware limitations, protocols, and device management.

In the development of the prototype TACCNET system for CSSCS, it was observed that many of these issues were tradeoffs between performance and functionality. That is, increased functionality could be achieved only at the expense of reduced performance in some related area. The decisions made in the development of the prototype tend to lean in favor of performance and simplicity of design or implementation. The TACCNET system was intended to demonstrate capabilities and is not suitable (in its current state) for a fielded, operational environment.

3.1. High Level Issues

The design issues to be discussed in this section are those which impact the system at the top level. These decisions will have a major impact on the nature of the software to be developed and on its functionality.

Many of these issues are not clearly defined at the time of preparation of this report. Further refinement of the functional description of the CSSCS and its internal and external interfaces and operations will be required before all of these issues can be fully resolved.

3.1.1. Network Topology

The Army is organized in a hierarchical fashion with responsibilities distributed among various internal organizations. These organizations are related through the chain of command in a formally defined hierarchical manner. The organizations within CSS which use the CSSCS will have a defined hierarchy, and it is reasonable to assume that their reporting and communications will observe this hierarchy. For our purposes, communications will be in the form of JINTACCS messages transmitted among CSSCS nodes.

3.1.1.1. Connectivity

Two types of networks are possible: a fully connected network in which any node can contact any other node as needed, or a polled network in which some nodes are only contacted by other nodes at specified intervals. The TACCNET prototype is a fully connected system. This choice was made to insure that important messages of high priority would not have to wait for a higher-echelon node to call in order to be transmitted and processed. In the prototype, nodes call one another whenever they have messages to deliver. They first check to be sure that a conversation with the desired system is not already in progress. Priority messages get fastest possible service, preempting routine messages or conversations if necessary. In a polled system, messages would have to wait (half the polling interval, on the average) for the next call from the master system regardless of priority.

The cost of a fully connected system is redundancy of capability, increased system complexity, greater overhead in communications, and increased usage of communications bandwidth. A polled system would tend to cut down the number of calls and increase the size of the transmissions as larger batches of messages would accumulate between contacts. The cost is mainly in the area of increased delay in propagating information through the system. In the battle field environment, timeliness of information can often be critical. There is also the matter of deciding who polls whom, how often to call, and how to handle

high priority messages. It is possible to envision a compromise, where some nodes are polled and others communicate at will. This type of system could take advantage of the inherent hierarchical relationships between superior and subordinate elements within CSS.

Store and forward message passing could be used to allow nodes which do not know how to contact each other directly to communicate through intermediate nodes. A modified "post office" scheme can be used to handle messages for unknown destinations - they are passed on to a designated node which may know how to deliver them (or may pass them on to another designated node, and so on). This would allow nodes to restrict their database of network contacts to those needed for routine operations.

3.1.1.2. Priority Message Handling

During the development of TACCNET we were unable to find information detailing the exact messages used within CSS and the frequencies of transmission of those messages. We did encounter some documentation which indicated that there would be response time criteria associated with some JINTACCS messages. The expected response times ranged from less than one minute to over a week. This implies a need for different classes of service for messages with different response time requirements. It is also reasonable to expect that there will be messages of different priorities (flash, immediate, secret, etc.) requiring different levels of service.

The question then arises: what types of special services are required for these types of messages? We may expect, at least, that there will be a requirement for immediate or fastest possible delivery and processing. There may also be a need for immediate reply while the sending node remains on the line waiting. Processing of database queries could be stratified to provide immediate service for queries at top priority levels. Ports and connections which are busy with routine transmissions may be preempted for priority transmissions. The incorporation of these capabilities complicates the system and can introduce the possibility of deadlocks or race conditions as processes of different priority compete for resources.

There is also the matter of node failures. What is to be done when there is a priority message for a site that cannot be contacted? Should the message be immediately rerouted to a designated backup node for prompt action? Does the type of message affect the action to be taken? If so, a knowledge base for the messages will be required for the system to determine the correct action for each type of message and each priority level. Obviously, this can become quite a serious consideration in the design of the system. The bottom line is that the system should give fastest possible service to important messages; preferably without degrading the performance of the network with respect to routine

traffic.

The TACCNET prototype provides preemptive service for priority messages, inserting them into active conversation streams where possible, preempting routine transmission if necessary, and scheduling calls and resources to give fastest possible service to priority messages. Only two levels of messages are recognized: priority messages and routine messages. Priority messages get fastest possible service while routine messages are served on a first-come-first-served basis. In a fielded system there will probably be a need for more different levels of priority, but we do not know at this time how many or what types of service they would require.

3.1.1.3. Alternate Sites

In the event that a node is down and cannot be reached, what is to be done with messages for that node? Some messages are routine reports and can wait until the site returns to action and calls for them, but others (especially priority messages) may require fast or even immediate action. With this in mind, the TACCNET prototype uses a table of designated *alternate sites* for each node in the network to re-route messages targeted for inactive sites. The method used is to keep a list of sites which can take over the processing of important messages during the node's absence. The list is traversed in order; if the first alternate site is also unavailable, the next one on the list is tried. Courtesy copies of all message sent to alternate sites should be kept and delivered to the original destination site once it has returned to operation.

This method implies a hierarchy of alternate sites in case of failure. This is consistent with the organization of CSS and the military chain of command. Without more information on the internal operation of CSS and the usage of JINTACCS messages for CSS functions we cannot determine what the hierarchy (and therefore the ordering of alternate sites) should be. There is also the question of how many alternate sites to put on the list. Should the chain stop at some point or should it continue all the way up the CSS chain of command. What is to be done with a message when no alternate sites are available?

3.1.2. Routing

We have already discussed the issues surrounding the topology of a CSSCS network. Regardless of the type of network proposed, it must be able to route message efficiently and automatically through the network. The user should be relieved of the details of selecting a path and should only need to know the desired destination of his message. In the case of routine or automated reports and queries, the system should keep track of message origins and destinations, leaving the operator responsible only for message composition or auditing.

Due to the dynamic nature of the CSSCS network environment, there are advantages to implementing a store and forward message passing system. When a message cannot be delivered directly to the desired site, it can be routed through and intermediate site which may have an active link to the desired site. Messages labeled for unknown destinations could be passed on to a designated "post office" site responsible for maintaining a complete database of network nodes and addresses. New nodes or nodes returning to action could call in to the post office to register and pick up waiting mail. The post office site could then distribute the new node's address to the rest of the network.

A side effect of the store and forward capability is the ability to route a message through a chain of intermediate sites to a final destination. Message paths would be composed of a sequence of node names. This could be used to broadcast messages of general importance to related groups of nodes or to use a specific set of links so as to avoid down or unreliable links. Ordinarily the user would supply only the final destination node name and the system would choose the shortest available path to that system, probably a direct connection via dialup. The user would, however, have the ability to override the system choice and specify a particular path. Aliases could be maintained by the system for complex or lengthy paths allowing the user to send a message to a designated group of nodes without remembering all of the nodes and their order or connectivity.

The TACCNET prototype provides store and forward message passing, automated path selection with optional user override, and path aliasing. It does not provide the post office method of dealing with undeliverable messages but does provide mail holding for departed or inactive sites.

3.1.2.1. Message Forwarding

We have already discussed alternate sites and message passing. In a network where nodes are expected to be mobile and to enter and leave the network at random we must provide a means for forwarding messages to appropriate nodes in the event that the designated recipient cannot be reached. This means that the system must have a set of criteria for use in evaluating the state of a node in the network. These criteria will be used to decide whether a node is down, temporarily unavailable, active, or destroyed. They system must be able to automatically decide whether to hold messages and keep trying to contact the remote site or to forward the messages to another site for delivery or processing.

The system must monitor the state of each node and take appropriate actions to maintain connectivity and continued operability. This may require the automatic rerouting of messages to insure prompt processing. It may require generation and maintenance of courtesy copies of messages for bypassed nodes

so that they may be brought up to date when they return to action. It may require special handling of priority messages for down nodes when it is not acceptable to wait through the retry process.

The TACCNET prototype provides the capability to determine the state of a site and to automatically route messages around a down site while keeping courtesy copies of all messages for bypassed sites. The courtesy copies are delivered when the site is successfully contacted and the site is restored to active status. The system keeps track of the current state of each node and keeps a record of the last successful contact as well as the number of failed attempts to contact a down site. Sites are declared to be down when the number of failed contact attempts exceeds a user determined threshold.

3.1.3. Failure Management

The discussion of message rerouting brings up the topic of link failure. There are different classes of failures which the system must be able to recognize and handle. The system may be limited in its ability to recognize some types of failures by the limitations of the communications equipment.† In any event, the system must be smart enough to distinguish between local failures (eg. can't dial out) and remote failures (eg. no answer or no login at remote modem).

3.1.3.1. Classes of Failures

The first class of failure is the local failure. This includes conditions such as no available ports, no response from local modem, modem unable to dial out, and inactive phone line. These conditions indicate local hardware or system problems and should not count against the remote site's connection history. They should not be considered when trying to determine whether a remote site is up or down. The proper response to these conditions will usually be to notify the operator and wait for correction of the situation. The incident should be logged automatically so that patterns of performance may be analyzed.

The second class of failure is the remote failure. This includes no answer from remote modem, busy signal, no carrier, no login prompt from remote system, and login or startup failure. These conditions span a range of problems from malfunctioning hardware to invalid login id or password. When the remote system answers the phone but does not allow login and synchronization we know that the site is operational and not destroyed. The correct action may be to keep calling or to change the login id or password for that system. When there

† For example: the D. C. Hayes Smartmodem employed in the TACCNET prototype does not distinguish between the "busy signal" and the "no answer" condition. The Cermetek Infomate does.

is no answer to the call, the site may be down or destroyed and the messages may need to be rerouted. If there was a busy signal, it may be sufficient to wait a while and call back. If the call is answered but there is no carrier, there may be a problem with the modems (possible hardware incompatibility). In each case, a note must be kept in a log file describing the result of the attempted call and the possible cause of failure.

The third class of failure is the transmission interruption. This includes link failure during transmission, cancellation of transmission, and preemption for priority messages. These types of failures do not usually indicate that the site has gone down. It will usually be sufficient to retry the connection after a short delay and continue transmission at the point of interruption. If the remote site has in fact gone down, the failure will be detected and handled as a class 2 failure as described above.

The TACCNET prototype detects and handle each of these classes of failures. Because of certain hardware limitations, it does not have the desired degree of resolution for class 2 failure diagnosis. It does recognize login and startup failures as well as transmission interruptions. Log files are maintained for each site showing the history of contacts and messages transferred. A system table is kept for the sites to monitor site status and contact times. Another table is used to monitor local port activity.

3.1.4. C^2 Database Backup and Recovery

Since the CSS environment can be volatile and nodes may be destroyed, it is desirable to build into the network the capability to back up and restore important information. Examples of such information include network configuration tables and the unit commander's C^2 database. This requires the designation of backup sites for each node. These need not necessarily be the same as the alternate processing sites for the node, but that would be a logical choice.

To increase the possibility of being able to recover a lost site's information, the database should be backed up on more than one remote site, each at a different location. The system designer must decide how many backup sites to allow or provide and how to insure that they are kept current. Another issue is frequency of backups: how often do we take a snapshot of the data for backup?

When the failure occurs and recovery is desired, how will it be initiated? A message may be sent to one of the backup sites to request an upload of the last database backup. It will be important to know and validate the time of that backup. It will probably be desirable to request retransmission of any messages sent to the destroyed node after the date of the last backup. The method for requesting these retransmissions must not flood the network with redundant

messages, but must make sure that all relevant information is obtained. It may be possible to avoid retransmission of old messages by restoring a snapshot backup and updating it from a higher level node.

3.1.5. Node Emulation

The TACCNET prototype system provides the capability to run more than one copy of the system on a single physical machine. This allows a node to perform the functions of a down or departed node in addition to its own work. Other nodes in the network do not need to know that the down node is being emulated.

All that is required to set up an emulated node is to create a root directory for TACCNET to use to handle the emulated node's work, disseminate the new phone number for the emulated node, adjust the site tables of the local and emulated node to show that they are in fact resident on a single machine, and invoke TACCNET in the new root directory.

This capability can be used to maintain a logical configuration even when network nodes are destroyed.

3.2. Middle Level Issues

In this section we will discuss issues relevant to the design, implementation, and operation of the major CSSCS communications system components. These issues include component interfaces, data structures, resource management, message processing, and database operations.

3.2.1. Component Interfaces

The CSSCS will be composed of several components including Decision Support, DataBase Management, and Communications. These components will in turn be composed of subsystems and processes which will cooperate in the performance of various tasks. The interfaces between CSSCS components and among the subsystems of the components must be well defined before the systems are implemented so that efforts are not wasted or duplicated.

In the discussion below we assume that the development environment for the CSSCS is that of the UNIX operating system or one of its derivatives. The issues discussed, however, are not unique to UNIX based systems.

3.2.1.1. Interprocess Communications

The number of separate but related functions to be performed by the CSSCS suggests that a group of asynchronous communicating processes be employed. This allows work to be distributed by function and to be performed more efficiently. A simple system could include a process to monitor incoming message traffic and schedule communications contacts, a process to initiate and execute an individual communications link, and a process to receive and process incoming messages. These processes would all be active at the same time and would need to communicate and share data.

The UNIX operating system does not provide sophisticated interprocess communication facilities, but it does provides processes with the ability to spawn other processes and to communicate with them in a limited fashion. A good example of this is found in the communications system of the TACCNET prototype. The top level process, *qms*, monitors system message queues and schedules calls to remote sites as needed. When it schedules a call, it spawns a new process known as a *caller* and passes information about the nature of the call to that process. The *qms* is then free to continue its operations without waiting for the call to be made or completed. When the *caller* has established contact with the remote site it spawns another new process called *iocontrol*. This process handles the transfer of messages between the two sites. The *caller* waits for *iocontrol* to finish and checks for errors or problems. It then terminates the connection and exits, returning all its resources to the system.

This example shows how processes can generate other processes to allow distribution of work. The *qms* is free to start up several *caller*s as needed, subject to the number of communications ports available on the system. The *qms* communicates with the *caller* only at startup time, telling it what system to call. The *caller* and *iocontrol* processes have a closer relationship, since the *caller* is responsible for setting up and maintaining a connection for *iocontrol* to use and for handling any errors that *iocontrol* may encounter. The *iocontrol* process could be implemented as a subroutine of *caller*, but its existence as a separate process allows us to use it at both ends of the communications link. When the *caller* connects to the remote site an *iocontrol* process is invoked (without a *caller*, since the connection is already established) to manage that end of the conversation. Once the originating *iocontrol* has sent all its messages, the two *iocontrol* processes swap roles and the remote site sends any messages in its queue. The two *iocontrol* processes are identical, and will continue to switch roles and exchange messages until both sites have emptied their queues.

The UNIX operating systems allows direct interprocess communications only between parent and child processes. This makes it difficult for cooperating asynchronous processes to interact in a well defined manner. Most UNIX implementations do not provide semaphore operations or other means for processes to synchronize operations or control resources. This means that the system developer will need to provide such facilities in the system. Many UNIX based software systems handle interprocess communication through disk files. Presence or absence of a special file can be used to indicate interrupt conditions, flag errors, or exchange critical information among processes. This method is slow and cumbersome, but is portable to all UNIX systems and is therefore in wide use. It is also possible to have a master process spawn all system processes as its children and then act as a switching center for communications using signals and waits.

In any event, there will be a need for limited interprocess communications among the CSSCS components. The system developer should attempt to use standard interfaces and communications methods which will be easily portable.

3.2.1.2. Directories as Queues

It is convenient to implement individual messages in the system as text files. This allows us to avoid the problems of queueing and organizing messages, leaving all that to the operating system. This is particularly convenient in a UNIX system where directories of files can be opened and read as files themselves. With a minimum of programming, we are able to treat the directories of files as first-in-first-out queues of messages. Files can be added, deleted, and moved using the UNIX function library commands *link()* and *unlink()*. Race conditions in the creation and copying of files are possible where continuously active processes scan the queues for input. The system must not be allowed to

begin processing a message file before it is completely written. This problem is avoided by using temporary "hidden" files which do not show up in normal directory scans.† These files are created, filled with text, and then made visible to the system for processing. They have been *enqueued.* In a system which provides better file protection it may be possible to rely on the operating system for the enqueueing and manipulation of files.

3.2.1.3. File Locking

Since the system must keep tables of information on sites, ports, and other resources, and since these tables are being read and updated by multiple processes executing concurrently, there must be some protection against conflicting reads and updates. The UNIX operating system in its normal implementation does not provide mechanisms such as file or record locking. It is possible to implement file locking using the file creation commands. A process wishing to access a critical file or resource must first attempt to create a special *lockfile.* If the lockfile already exists (meaning that some other process is using the resource) the create operation fails. The process can then exit or it could keep trying to create the lockfile, sleeping briefly in between attempts to give the other process an opportunity to finish with and release the resource. A threshold must be defined to limit the number of failed attempts and prevent infinite wait loops. When the create operation succeeds, we are confident that no other process is accessing the file.

Of course, the success of this method depends on the cooperation of all processes using the resource. Rogue processes which access the resource without regard to the lockfile can cause unpredictable results. There is also the potential for deadlock since processes are "on their honor" to release the resource when they are through. These methods are standard for UNIX programmers, but may be avoided if the development operating system provides true file or record locking capabilities. The TACCNET prototype uses the above method for illustrative purposes and to guarantee portability across the UNIX family.

3.2.1.4. Preemption

We have already discussed preemption of resources for priority message handling. Since some systems do not allow communication between unrelated processes, we must find a way to signal or interrupt a running process. The simplest method is to have processes periodically check for the presence of a special file. A process wishing to interrupt another process simply creates the special file in a given directory. When the target process sees the interrupt file

† On UNIX systems, files having a period (".") as the first character of their names are hidden files.

it exits, freeing its resources for use by the waiting process. The disadvantage of this method is that the periodic checks add to the system processing overhead and may degrade system preformance. The TACCNET prototype system checks for interrupt files between transmission packets (if so configured) and between message files (by default).

3.2.2. Connection Management

One of the major mid-level design issues is that of connection management. This encompasses the establishment, monitoring, and termination of connections between systems.

3.2.2.1. Modem Setup

Once the system has determined that a call should be made to a remote node, a process must take a communications port, set the configuration of the port, contact the modem on that port, determine the telephone number of the remote node, instruct the modem to dial the number, and listen for the remote node to answer. This process must determine whether the remote node has answered, is busy, or has not answered the call. This will require interaction with the modem.

Once the remote node has answered, the process must log in to the remote node and start up a cooperating process to talk to. Errors or problems during the login and synchronization must be detected and handled, recovering when possible. The TACCNET system uses a simple finite state automaton to handle the login and handshaking with the remote system.

Once the connection has been established, the caller process can turn the connection over to another process which will manage the conversation. This process could be a subroutine of the caller process, but may be better implemented as a separate process for reasons described above. When the conversation is complete, it will be the responsibility of the caller process to make sure that the connection is broken and that the phone is hung up properly. The modem may need resetting to prepare it for use by the next caller.

3.2.2.2. Dialing and Error Detection

In order to establish a connection between nodes, the originating node must connect to a modem and instruct the modem to dial the phone number of the remote node. It is possible that nodes may be configured with more than one dialin line in order to reduce connection failures due to busy lines. The system site table (described fully in section 4.3.4.1) must then contain a list of phone numbers for each node in the network. When a *caller* process is invoked it

must be supplied with this list of phone numbers and the name of the communications port to use.

The *caller* can then try dialing each number on the list in succession until one is answered and a connection established. Once a connection is established, operation proceeds normally. If the *caller* cannot get an answer on any of the numbers, the connection attempt has failed and this result must be logged. The site table will be modified to indicate a failed attempt and the retry count will be checked against the defined threshold to determine if the system should be declared down.

If the remote site is not declared down, the system could wait for some specified delay interval and then try again. This interval is called the *retry delay* and should be configurable by the operator at run time. If the node is declared down, the system should forward all messages for the remote node to the designated backup site and make courtesy copies for the down node. The system should then try to call the down site periodically to see if it has returned to service. The interval between attempts should be long compared to the retry delay and should also be configurable by the operator. This delay is called the *down delay.*

We would like to be able to determine the reason that the remote node did not answer the call. To do this requires a modem which can distinguish between "ring, no answer" and "busy, no answer". We must also detect the case where the remote modem answers the phone but does not supply a carrier signal and the case where it answers and gives the carrier tone but no login prompt appears.

Each of these conditions represents a different type of failure and the system may respond differently in each case. For instance, the busy signal does not necessarily mean that the remote system is down. It may suffice to call again in a few minutes when the current conversation is complete. We may not want to increment the retry count in the site table in the case of busy signals, as we would when a call rings and gets no answer.

The type of modem used will determine the ease or difficulty of detecting these conditions. The ideal modem should provide a different return code for each of these conditions, as well as codes for "connection established", "modem ready", and "no connections".

There is some question of how to determine whether a remote node is up or down. Obviously, if we can contact the node it is up. However, just because we cannot contact the node we cannot immediately assume that it is down. It may be that the line is busy, as described above. It may be that the system is

up but its modem is turned off or broken. The fact is that we cannot determine that a remote node is down in any direct manner.

We can, however, establish criteria to guide the operation of the system and cover the general case of down nodes. One way to do this is to pick a threshold value for the number of failed attempts to call a node. The system keeps count of the attempts, resetting the number to zero when a successful connection is made. If the number of failed attempts exceeds the threshold value the node is declared down and its messages are forwarded to the designated backup node.

We must consider the possibility that there may be problems with the local modem. These could include a broken or disconnected modem, a disconnected phone line, and so on. Failures due to these conditions are not related to the state of the remote node and should not be counted as failed attempts.

The system designer may not have sufficient information to define the failure thresholds or the delay intervals between attempts to call the remote node. These values should be regarded as tunable system parameters to be defined by the system operator when the system is started. They should be read in from a system configuration file at run time and should be supplied with reasonable default values.

3.2.2.3. Disconnect

When a call is complete we must be sure that the system can hang up the phone and reset the modem for use by the next *caller*. Some telephone networks do not automatically disconnect a line when one party terminates the call. This can lead to permanently busy lines and unnecessary failed calls. When a conversation is complete, each system, both remote and local, should issue a hangup command to its modem and then check the modem status to make sure the line has been dropped. The modem should then be reset for the next call and the port connection released.

3.2.3. Database Operations

In this section we will discuss some of the issues surrounding automated database operations and JINTACCS automation. One expected use of JINTACCS messages will be the transfer of information from machine to machine within the CSSCS. Messages will use data from the unit Command and Control (C^2) databases. It will be worthwhile to consider the issues described below in the development of automated database servers for C^2 and JINTACCS processing.

3.2.3.1. Message Types

There are many different message types in the JINTACCS set. The messages are primarily intended to report information for storage or aggregation. Some of the messages, however, may be used to request rather than report information. This is done by labeling the message as a request message in a special field of the initial main text of the message.

Request messages to be processed automatically will require the presence of a database server on the system. That server must be able to distinguish between request messages and report messages. Request message will require the server to query the database for information, format the information into a JINTACCS message, and send the response message to the unit which generated the request.

Report messages are those which contain data to be placed into the C^2 database. A report message may contain a full report of the status of some item or unit, or it may contain update information for certain elements of a unit. At this time, it is not clear how to use the update feature of JINTACCS messages and all incoming report messages are regarded as full status reports, overwriting existing information in the database.

3.2.3.2. Database Access From Programs

In order for the system to access the information in the database it must be able to formulate and execute queries in the database query language. It is possible to write queries and store them in files for use by the server. This is the method employed by the prototype TACCNET system. If the database management system used for the C^2 database provides an embedded query language facility for use with the development language of the server system, a number of operations can be carried out more efficiently than with stored queries.

A message dictionary can be built in the DBMS for use by the server in formatting information into JINTACCS messages. The embedded query language makes it possible to retrieve information about the message structure from the message dictionary in a simple and logical manner. The embedded query language also allows database query and update operations to be designed and coded for each message and then compiled into a message handling program using compiler generating tools such as **yacc** and **lex.**† This reduces and confines the message specific programming to a small part of the server system.

† The yacc and lex programs are supplied with the UNIX operating system and are used to develop parsers and compilers. The processing of JINTACCS messages is very similar to the operation of a compiler in which input text in a given syntax is decoded and certain operations are performed based on the input text.

3.2.3.3. JINTACCS Automation

JINTACCS message are defined in a hierarchical manner. Messages are composed of sets which are then composed of fields. Fields are composed of data elements which may in turn be broken into sub-elements. This suggests a hierarchical representation of the message as a collection of set structures ordered in a defined sequence. The ordering of sets for each message type is given in the JINTACCS definition manual.

The messages are intended to be readable both by human operators and by machines (ie. database servers). Unfortunately, there are numerous inconsistencies and ambiguities remaining in the message definitions which make it difficult for machines or humans to parse the messages correctly. One example of this is the lack of data element delimiters in chain fields. The string "100LBOG90" could mean 100 pounds of OG90 or 100 liters of BOG90.† We expect many of these problems to be ironed out in future revisions of the JINTACCS standards. In any event, it is desirable to minimize and if possible avoid message specific code in the database server system.

3.2.4. Message Generation

Messages may be generated by users or by processes running on the CSSCS machine. These messages will travel through the network following some path to their destinations. It will be necessary to maintain an audit trail for each message so that it can be tracked through the system. This requires that each message be uniquely identified across the entire network. We have already required that each node be uniquely identified. If we use the identifier of the node at which a message originates as part of the message identifier, we need only sequence the messages on each node in order to uniquely label each message. An example of this would be a message generated at a site named **bravo** which would have an identifier of the form

Xbravo12345

where **X** is an indicator of message type (email, JINTACCS, administrative, etc.) and **12345** is the sequence number of the message, assigned at the time of creation. The sequence number must be large enough so that it does not roll over too often (which would allow duplicate names). The sequence number could be generated from the system clock or from a global variable which would be incremented after each new message is created.

The message identifier must become part of the message and must travel with the message through the network. It may be convenient to use the message

† This example is taken from the Fuel-Oil-Lubricant field of the JINTACCS POL Location message S026.

identifier as the name of the file in which the message is stored. This is the case in the prototype TACCNET system. The message id is then preserved as the message file is transferred to other nodes in the network.

In any event, the unique message id allows us to retrace the trail of lost messages and to recover them from archives at the origin or intermediate sites. Each message should be copied to an archive when it arrives at a node.

3.2.5. Message Processing

It is not sufficient to provide file transfer among CSSCS nodes and call that a network. Functions such as error logging, routing, failure management, and network configuration require various levels of message processing capability. In this section we will describe some of the functional areas which involve the processing of messages passed by the network.

3.2.5.1. Network Administration

One goal for the CSSCS network is that it be dynamically reconfigurable. By this we mean that the topology of the network can be modified "on the fly" without taking the network down. Ideally, this should be handled automatically by the network software without operator intervention.

Several types of operations could be performed on remote nodes through the use of a special class of administrative messages. Addition and deletion of nodes should be handled automatically upon receipt of a message defining the action to be taken. Special messages should be provided to query a remote node's configuration tables and return the results of the query. It will be desirable to be able to change or update information in the configuration tables of remote sites by sending messages.

All this implies that the system will need to be able to examine incoming messages and determine whether or not they are administrative messages. Messages will need to be identified by type for ease of processing. These capabilities can be combined with other message processing functions described below in a system message processor to be run continuously at each node.

3.2.5.2. Electronic Mail & File Transfer

File transfer capability is basic to the operation of the CSSCS network. Though the primary usage of the system is expected to be the transmission of JINTACCS messages, it is easy to piggyback an electronic mail system on top of the basic network. This could be used for document and file transfer, system backups, administrative messages, and so on. The only cost is a little extra

programming in the message processor to recognize user mail messages or file transfers and pass them on to the operating system for handling. This is particularly easy to do on UNIX systems, where the files can be piped into the **mail** program which then takes over responsibility for delivery.

3.2.5.3. Path Validation

Each message that comes into a node, whether from a remote node, a local user, or a process, must be examined to determine its destination. The network path to that destination must then be determined and validated. Unknown destinations or invalid paths must be detected and handled. Messages with valid destinations must be placed in the proper system queues corresponding with their destinations. Some messages will be bound for users or processes at the destination node (eg. a database server) and must be identified as such. This requires that a local message processor examine each message, determine its proper disposition, and take appropriate actions.

3.2.5.4. Message Forwarding

In the event that a message cannot be delivered to the intended destination, some process must take charge of the message and decide what to do with it. It may be that the proper action is to hold the message until the desired site returns to action. Alternatively, the message may need to be rerouted to some alternate site for processing. We have already discussed the issue of message routing and alternate sites. Here, we point out that these functions are best carried out by the message processing part of the system.

3.2.5.5. Message Holding

In the event that a node has departed and is out of contact, we may wish the system to hold any messages for that node until its return to action. The message processing system will need to keep the status of each node in the network available so that it can tell whether to hold or forward messages for such a node. The system may need to create special directories in which to hold these messages. It will need to detect the return of such nodes to action, giving them their saved messages when they call in.

3.3. Low Level Issues

In this section, topics of relevance to the details of the TACCNET system design will be discussed. Questions ranging from queue and port management to hardware and operating system requirements will be addressed.

Many of the issues treated here are in light of the higher level design decisions discussed previously. In analyzing these decisions, an attempt has been made to consider the broadest problems and their solutions. Most of the discussion here focuses on the details of the TACCNET implementation itself, since this best demonstrates the issues uncovered by the research. Where applicable, alternative approaches are proposed.

3.3.1. Queue Management

The use of file system directories as queues is very convenient in that it allows the developer to leave the task of managing and ordering the message files to the operating system. There are, however, certain issues which must still be considered.

3.3.1.1. Directory Access

Secondary storage must be partitionable into directories or collections of files, and a naming convention must exist for referencing these directories and their contents. Directories are essential to the technique of separating messages into queues for specific systems or processes.

In every UNIX environment, as with many other operating systems, each level of a tree-structured directory can be treated as a standard data file. This permits application programs to query the contents of a directory rapidly and easily, as long as the structure of a directory entry is defined. Though this methodology is not absolutely necessary for accessing the names of files within a directory, it is a practicable and consistent technique for doing so.

Since directories can represent queues in the CSSCS environment, and their files can represent JINTACCS and administrative messages, this structure seems well-suited to the purpose of organizing information for the CSSCS.

Another advantage of this organization is that files can be created and removed in an "atomic" operation. That is, the function of deallocating or renaming a file is no more than removing or changing the entry for that file from the directory. Most file systems which are organized in this way support the concept of "linking", in which data is a single object which can be addressed by several

different names. Creating a new link to a file is the same as instantaneously creating a file with the new name anywhere in the filesystem. When a new link is created, the data within the "new" file is guaranteed to be complete (as complete as the original file) the instant the new directory entry appears.

Since each component of the TACCNET system is asynchronous with the others and constantly scans queues for new entries to be processed, the possibility of a race condition occurs when a new file appears in a queue before it is completely ready for processing. With the file system features mentioned above, CSS messages can be enqueued in their entirety and in a single operation, through the creation of a temporary file (not "visible" in any queue) and then the ultimate linking of that file to the target queue.

3.3.1.2. File Locking

UNIX does not provide a method for restricting concurrent access to data files. When two asynchronous processes attempt to access the same data file, there must be some guarantee that a race condition will not occur. If one of the processes were to modify the file while another was reading it, the data could be corrupted, resulting in serious errors. To assure that such concurrency of access is not permitted, a file locking mechanism must be instituted.

In the TACCNET prototype, lock files are used to indicate that a corresponding file is "busy", or being accessed by another process. Before a process can begin reading or writing a common data file, it must create the designated lock file for that data file. If the lock file already exists, in which case it cannot be created, the process cannot access the file. In this way, all other processes are "locked out" of a file until its corresponding lock file is removed by the process currently accessing the file.

The action of creating the lock file is considered to be atomic. That is, when any two processes attempt to create the lock file at the same time, using the UNIX **open(2)** call, for example, only one will be permitted to create the file, and the other will receive an error condition. If this facility did not exist, more complicated schemes would be necessary for assuring mutual exclusion for the lock file itself.

If a process attempts to lock a file and is unable, provisions must be made for assuring either eventual access to the file or eventual failure. Once the file is released by a process, its associated lock file is removed. A waiting process can seize the file by creating the lock file after that point. If more than one process is waiting, only one will gain access to the file, and the others will continue to try to create the lock file until they either succeed or give up.

It is assumed that all accesses to a data file are for relatively short intervals of time, so that a threshold can be determined for retrying access to the file. A process cannot wait forever to lock a file. Eventually, based on the amount of unsuccessful retries, the process must give up its attempt to access the file, declaring an error. This prevents deadlock, and signals the system administrator that an abnormal condition has occurred, such as a process terminating without releasing its resources.

Some operating systems (including certain UNIX implementations) include support for system semaphores. These are normally implemented in the system kernel and are accessed via system calls. They can be used to guarantee exclusive access to protected resources such as modem ports, the site table, and the port table. While the use of these features provides a more reliable and efficient protection mechanism than that described above, it must be considered carefully as it is not likely to be portable across different operating systems or even different UNIX versions.

3.3.1.3. Preemption

The different components of the CSSCS act asynchronously, although there is some level of communication among the components in certain specific cases. Specifically, the *qms* process must be able to send a message to a *caller* process requesting that a line be released immediately. This sort of preemption requires a mechanism for communications between two independent processes.

Different operating systems support different levels of interprocess communication. The UNIX operating system, for example, only supports communication between processes through the software interrupt facility. That is, a process which has created a subprocess can then send interrupts to the "child" and receive a return code from the child upon its completion. Unfortunately, this essentially unidirectional communication requires that a process maintain a list of the process identifications for all of its children so that it can know where to send an interrupt. The task of maintaining and validating process information is both difficult and system-specific.

In the TACCNET prototype, each asynchronous process is associated with a queue or set of queues, in a hierarchical fashion. For example, the *qms* works with the highest level TACCNET directory, scanning the subqueues for new entries, while a given *iocontrol* process is directly associated with the system queue for which it is invoked. This structure permits another form of communication, in which signals are created in the form of specific data files in the queue of the process to be signalled.

This approach is used in the TACCNET system because it is not operating-system dependent and it is easy to implement. When the *qms* desires to preempt communications with a remote system to gain access to a *dialout* modem, it sends such a signal to the affected process by placing a special interrupt file in the currently active system's queue. This file, appropriately named "**.Interrupt**", will be deleted when the affected process recognizes its existence in the system queue. In this way, the interrupt signal is acknowledged so that the *qms* can then take over control of the released resources.

Since a process must **poll** its queue for special interrupt signal files, these signals are not truly asynchronous. A process might not recognize the existence of an interrupt file for a long time, if at all. While ignoring the interrupt is not prescribed, the process requesting an interrupt must expect that an interrupt may not be acknowledged, in which case it should not wait forever for a resource. If a process cannot gain control after a certain time threshold, it should abort, indicating the error to the operator for manual intervention.

The approach of creating signal files and forcing the affected process to poll for such files was chosen in the implementation of TACCNET for reasons of simplicity and portability. Other methods could be more effective if their implementation were not complex and operating-system specific.

3.3.2. Port Management

The CSSCS will have several communications ports available for network use. It will be necessary to allocate these resources efficiently to guarantee maximum service. Issues to be considered include port multiplexing, port status, numbers of ports per node, and priority service.

3.3.2.1. Configuring Ports

Because each system in the network must be capable of both calling remote systems and receiving calls from remote systems, the issue of assigning ports for these functions must be addressed. Two approaches can be taken to configuring communications ports for this purpose. One technique is to multiplex each port between both of these functions. Each port on the system would then be capable of both dialing a remote node and receiving calls from a remote node. Another technique is to reserve a select number of *dialin* ports and a select number of *dialout* ports, which do not overlap. The *dialin* ports would only receive calls from remote systems, while the *dialout* ports would only be used to connect to remote systems. In either scheme, once a connection is established between two systems, data can flow in either direction.

In considering the tradeoffs involved in implementing each of these techniques, it was decided that the most important parameters in analyzing their workability are the **minimum number of ports** necessary for full network capability, the **portability** of the software required to implement each scheme, and the **complexity of implementation** of such a system.

The port multiplexing technique offers the ability to fully utilize all communications ports on a given machine, so that the number of ports necessary is minimized. However, in considering the details of such an implementation, it was found that the level of complexity of an implementation of such a system would be very high. For example, if a port is currently idle and in a *dialin* state, and a call is scheduled for a remote system, how can we safeguard against a remote system dialing in just as the state of the line is changed to *dialout* ? One of the two connections, if not both, will fail, causing a significant delay in the eventual success of each connection. This type of **race condition** adds greatly to the difficulty of implementing this scheme. In most cases, adequate solutions would require very low-level dependencies on the hardware and operating system device drivers, greatly degrading the portability of the system. The mechanism for varying ports between *dialin* and *dialout* functions is not consistent among different versions of UNIX, negatively impacting portability as well.

The alternative technique, which is to allocate specific ports as either exclusively *dialin* or exclusively *dialout*, essentially doubles the minimum number of ports necessary to make a given machine fully connected. On the other hand, such a scheme could be easily implemented by associating a single process with each *dialin* port (such as a UNIX *getty*) that would invoke the *iocontrol* system when a call is received. When a call to a remote system is scheduled, the *qms* would be guaranteed of the availability of the *dialout* port upon examination of the port table. (This process is described below.) Such a system would be much more portable, since it would not rely on the details of operating system port manipulations.

In choosing a technique for the prototype TACCNET system, the development team ranked such factors as portability and ease of implementation above the number of ports that would be necessary. The current TACCNET system utilizes the distinct *dialin* vs. *dialout* technique, in which a UNIX *getty* is associated with each *dialin* port. Although a program could be written as part of the TACCNET software to monitor incoming calls, using a *getty* allows a *dialin* port to be used for any system login and does not limit the port for strictly TACCNET use. To permit access to the TACCNET system, a special account can be created to run the *iocontrol* program in *slave* mode when a login is made to that account.

3.3.2.2. Port Usage

Several problems arise when considering the control of access to input/output ports by multiple processes. Information must be maintained concerning the ports that are available, their functions, and their status. The TACCNET prototype addresses most of these problems through the use of a **Port Table** and lock files.

For a given TACCNET configuration, there will be a static number of ports available for *dialout*. One of the main responsibilities of the *qms* is to orchestrate the usage of these ports for calling remote systems when messages need to be transmitted. The Port Table file contains a list of the existing *dialout* ports and the status of each.

This file, described in detail in the "TACCNET Installation and Operation" section of this document, contains entries for each *dialout* port available to the TACCNET system. Each entry contains the port name, the system for which a *caller* has been invoked on that port, and the state of the connection currently being made on that port. If the port is free, the system name is set to "free", and the connection state is set to "available".

When the *qms* schedules a call to a remote site, it examines the Port Table to see if a *caller* has already been invoked for that site. If so, there is no need to call, and the *qms* will continue scanning its queues. The *qms* also checks for lockfiles on the system queues which would indicate that a conversation with the desired site is already in progress, having been originated by the remote site.

If the remote site is not currently in communication with this system, the *qms* will examine the Port Table to see if any ports are available for a *caller* to call that system. If there is at least one available port, the *qms* immediately modifies the Port Table to reflect the name of the site about to be called, setting the connection state to a key corresponding to the highest priority of the messages to be transmitted to the site.† The port is released when the *caller* terminates the connection for any reason.

If there are only routine messages to be sent and no port is available for the *qms* to assign, no action is taken until a port becomes available. If priority messages are to be transmitted, however, the *qms* will seize one of the ports (if there are any ports being used for routine messages alone), forcing the process

† Concurrent updates to the Port Table are not permitted, so, as with most tables in the system, lock files are used to guard against any concurrent access to the file.

associated with that port to release it.

Because a remote system may call the local system at any time, lock files are associated with system queues. These lock files assure mutually exclusive access to the files within a system queue. There can never be two active conversations between the same two sites at the same time because of this, since one of the two connections will be dropped when the *qms* sees the locked system queue.

Other techniques, including more general control of system-port assignments and functions, are possible and could be considered. The specific methods for port management described above contain what seems to comprise the minimal set of functions necessary to provide for multiple port management in this environment.

3.3.3. Communications Protocol

In addition to end-to-end error detection, the communications protocol must provide certain functions to the network. These functions include node identification, file identification, broadcast message handling, and support for preemption or cancellation of a transmission.

3.3.3.1. Identification

Each node in the network has a unique name which must be known by any other TACCNET node with which it communicates. Because of the division of messages into system queues, there must be away for a system to identify itself when it calls another node so that the proper queues can be associated with a transaction. When a connection is established by a remote node, the *iocontrol* system is invoked on the local node. The remote system identifies itself by transmitting a data packet containing its name. The local node will verify the packet, sending an acknowledge code in response. Before files can be transferred between the systems, the local node must next verify the *existence* of the remote node (by examining its tables) and transmit a second acknowledge code to validate the connection. If the remote node is not listed in its tables, the local system will reject the call, disabling any further dialog with the remote.

A second level of identification occurs immediately before each file (or message) is transmitted from one node to the other. Since message names are chosen by the system to be unique throughout the network, these are preserved when a message is transferred to other nodes. Thus, the communications protocol dictates that each message be preceded by its unique network identification. This identification is transmitted by the sender in the form of a

data packet preceding the message text to be transmitted. The receiving node validates the message name packet, sending an acknowledgement, and assigns this file name to the data that follows. Immediately after acknowledging the file name packet, the receiving node will transmit a file acceptance acknowledgement, indicating whether or not it will accept the file. If it rejects the file (transmitting a negative acknowledgement), the sender will delete the file from its queue and proceed with any more files it might have to send. Otherwise, the sender will transmit the packets that compose the message.

Yet a third level of identification takes place within packets. To ensure that synchronization is maintained between two systems, packets are numbered relative to a given session. These numbers are incremented each time a new data packet is prepared for transmission. Packets that have to be retransmitted in case of error maintain their identity to ensure the order of the packets within the received file. Because of the nature of the protocol, only two unique packet numbers are required, although the current implementation allows for 256 distinct packet numbers. Packets are considered to be in error if they contain packet numbers not within with the expectations of the receiver.

3.3.3.2. Message Rejection

Because each message that travels throughout the network must have a unique name, it is possible to assure that redundant messages (or those which have already been received by a given node) are not wastefully retransmitted.

During communications between two nodes, each message is identified by its network-wide message name, as described above. Since each message received by a system is locally archived, a given node can determine if it has already received a message based on this identification, and thus reject the message if it is redundant.

For example, when a database recovery is initiated by a node which has just re-entered the network, a *broadcast message* is transmitted requesting that all other nodes in the network retransmit archived messages for that node. Broadcast messages are propagated in a redundant fashion to all nodes in the network. In other words, some nodes may receive several identical messages from different sources. With message rejection, only the message identification packet for a redundant message is transmitted, since the receiver will reject messages it has already received. Thus, redundant database recovery requests are not transmitted, greatly decreasing message traffic where possible.

3.3.3.3. Preemption and Cancellation

There are several cases in which the transfer of messages between two CSSCS nodes must be interrupted. Two such cases are the preemption of connections for priority traffic and the cancellation of connections by operator request. The communications protocol must support a mechanism for these cases in which a connection may be dropped "cleanly" and abruptly.

In the prototype TACCNET communications protocol, a "cancel" control code is defined to indicate the need for immediate termination of a connection by either node. When a node recognizes a priority preemption or operator cancellation request, as discussed in the "Queue Management" section above, it may transmit this "cancel" code as a control packet in place of the next control *or* data packet it would normally transmit. This code will inform the remote system that the connection is being prematurely terminated so that no failures will be assumed when the line is released.

Since the protocol is defined such that each node must respond to each packet transmitted by the other node, cancellation can be recognized on the remote system at the instant the need arises. Each of the systems can release its resources and resume operations as prescribed by its local environment. The abrupt termination of a connection during transmission or reception will not be interpreted as an error in this case, since an external entity will have evidently requested this action. If a node suddenly loses a connection with a remote without receiving this "cancel" control packet, however, an error will be logged and the site will be considered to have gone down for an unexplained reason.

In the case of priority messages entering the queue for a system currently connected, there will be no need to fully interrupt the conversation and then redial the remote system. Instead, priority messages can be "inserted" into the queue, so that they are transmitted in place of the very next routine message to be transmitted. Priority messages in the same queue with routine messages are always given immediate service.

The current prototype does not, however, support a mechanism for forcing an immediate reversal of control. When a priority message is inserted into the outgoing queue of a given system, it may not be transmitted until all of the incoming messages are received. This is because each site gains control of the conversation when the other site has transmitted all of its messages.

A possible expansion to the protocol would permit one site to regain control over the other in the event of priority messages, so that it could transmit the messages and then return control. The system requesting control would have to wait for an acknowledgement from the controlling system, as the latter might be transmitting priority messages at the time of the request, in which case it should not be interrupted.

3.3.4. Hardware Issues

In order to implement a communications system with the desired degree of robustness and failure recovery desired, certain capabilities must be present in the system hardware. These requirements fall into two classes: communications equipment requirements and computer system requirements.

3.3.4.1. Communications Requirements

In order that the software can clearly and rapidly detect the failure of a connection to a remote system, the communications equipment must provide a method for detecting failures. A modem must be able to recognize and indicate that a connection could not be made, by returning a "no answer" or "busy" status code to the software driving it. Such a modem must also assure that, when a connection is broken for any reason, the communications line will be properly reset and the software will be informed of this condition. The interface for the modem must thus provide for the generation of result codes.

There must also be a programmatic method for configuring the modem parameters such as parity and local echo, as well as for instructing the modem to dial a number and return result codes. Equally important, there must be a simple method for switching the modem to a command mode, so that the software can force the modem to disconnect from a remote system at any time.

Within the communications hardware of the computer system, there must exist a set of standard RS-232 input/output ports with connectors for attaching modems and other RS-232 lines. These ports must be accessible and selectable by application software. As well as providing the raw input/output interface for the system, these ports must be configurable for such parameters as baud rate, parity, and word length. To obey conventions of asynchronous computer communications, these ports must be configured as Data Communications Equipment (DCE's) which connect to peripherals that will take on the role of Data Terminal Equipment (DTE's).

Based on the decision to define input/output ports as either solely *dialin* or solely *dialout*, there must be at least two ports on a fully connected TACCNET system, one configured for *dialin*, and the other for *dialout*. There may be any number of additional ports defined for either *dialin* or *dialout*.

It is not important which type of modem is associated with a *dialin* port, since it need only serve the function of answering an incoming call and connecting the calling system. The current TACCNET prototype requires, however, that all *dialout* ports be connected to a D.C. Hayes Smartmodem, or a modem with an identical command set.

3.3.4.2. System Requirements

At a higher level, the computer operating system must provide a versatile interface to the actual input/output ports that will be connected to modems or other devices. Because the *iocontrol* program must be able to adjust the parity, word length, baud rate, and buffering mode of an input/output port, the operating system must provide an interface for configuring an RS-232 port programmatically and straightforwardly. More specifically, the software must be able to disable parity, request single-character input/output (disable buffering), and force the flushing of input buffers when applicable.

The TACCNET system relies heavily upon the ability to perform **timeouts** to control its communications. For this reason, the computer system must provide for aborting a **read** operation after a specified time threshold is exceeded. In the very least, the system should allow **read** operations to be interrupted by external means, such as "alarm" or "countdown" functions. It would be possible to implement timeouts using alternative techniques, such as **busy-waiting,** but these are typically CPU-bound and do not facilitate use of absolute time thresholds.

The computer system must also provide a facility for associating a given *dialin* port with a process or system login procedure. In the case of the latter, support must exist for invoking a specified application program to run with a given user account once logged in.

Since *dialin* ports will be associated with processes in this way, the computer system must provide a naming convention or other access technique for the identification of the "current" port. For example, the *iocontrol* system will be invoked on a *dialin* port after a connection has been established, and will need to make modifications to the port attributes of the *dialin* port. It will thus need access to the name of the port so that it can configure the port and begin communications.

Above all, the computer system must support an architecture for multiprogramming. There should be operating system support of functions to permit application software to create and execute specified processes, wait for their completion, and receive status information after they are done.

The UNIX System III operating system, under which the entire prototype CSSCS has been developed, fits all of these criteria and was chosen for this reason. The prototype will operate in most System III and System V derivatives, including XENIX and DISTRIX. Other UNIX implementations, such as Berkeley UNIX, perform most of these functions as well, although the CSSCS is not guaranteed to be compatible with these.

4. TACCNET Prototype Specification

In the following sections we will describe the prototype system developed at Georgia Tech as part of a series of Advanced Experimental Demonstrations for AIRMICS. The system has been dubbed "TACCNET" for Tactical Army Command and Control Network. TACCNET provides connectivity and communications among the nodes of the Combat Service Support System. It allows these nodes to exchange information in the form of JINTACCS messages, text files, electronic mail, and distributed database backups. The prototype system supports a small set of CSS applicable JINTACCS messages and offers a relational database for storage and retrieval of command and control information. Support for other JINTACCS messages can be added by entering the message format into the message data dictionary stored in the relational database. The database can be backed up over the network to another node and retrieved as needed. A single node can simulate multiple nodes in the network if needed, taking the place of a down or destroyed node.

The hardware and software requirements for the prototype system are described in detail below. Installation and configuration information is also provided. The system has been tested on IBM, Burroughs, and ONYX systems and should be easily portable to any computer running the UNIX operating system. All of the code is written in the C programming language. Some user commands are actually shell command scripts written in the Bourne shell command language.

4.1. TACCNET System Requirements

In this section the hardware and software requirements for the TACCNET software are defined. The system was designed and developed with portability as a major concern and should be easily ported to any machine using the UNIX System III operating system or one of its derivatives.

4.1.1. Hardware

The TACCNET software has been installed and tested on ONYX and IBM PC/AT computers. These machines were used to simulate the TACCS system. The system requires at least one 360K floppy disk and a minimum of 10 megabytes of hard disk storage. The minimum memory required to run the system is 512K bytes. The GCOS part of the system is installed on a Honeywell DPS Level 6 minicomputer.

4.1.2. Operating Systems

All software for the TACCS has been written in the C programming language and developed under the ONYX and XENIX operating systems. Each of these systems is a derivative of the UNIX operating system. The software will run without modification on any UNIX System III machine.

Software for the DAS3 has been written in C and developed under the GCOS Mod 400 operating system using the M4_CC compiler. While there are differences in implementation details between the GCOS and UNIX versions of TACCNET, the functionality at the communications and protocol level is the same. The message processing and network management levels have not been implemented in the GCOS version.

4.1.3. Communications Equipment

The modems employed in the development and demonstration of the prototype system are D. C. Hayes Smartmodem 1200 standalone modems. These modems are auto-dial, auto-answer, programmable asynchronous 1200 bps devices intended for use with voice grade telephone lines. The TACCNET software uses Smartmodem commands to program the modem and dial the phone, thus requiring the use of the Hayes Smartmodem on all *dialout* lines. Any auto-answer modem could be used for the *dialin* lines.

Each TACCS system must have at least one telephone line for *dialin* use and one line for *dialout* use. More lines may be allocated for each mode if available. Section 4.3 explains the configuration of phone lines for the system. The DAS3 must have one phone line for *dialout* use. The modem on this line must

be a Cermetek Infomate 1200 bps modem. The proper configuration of the Honeywell port is explained in section 4.3.

4.1.4. Software

The TACCNET prototype system provides automated database operations for a sample set of three JINTACCS messages. These messages may be generated from information stored in the prototype C^2 database or they may be used to update information in the database. These functions are provided by the database *server* program and are dependent on the presence of the UNIFY† relational database management system. The dbms must be configured to include the relations defined for the C^2 database prototype which is described in section 4.2.4.

† UNIFY is a trademark of the UNIFY Corporation.

4.2. TACCNET System Overview

The TACCNET software is composed of three major subsystems. The *Communications* system handles reception and transmission of messages. The *Message Processing* system examines messages and processes them according to type and content. The *Database Server* system handles all messages originating from or posted to the C^2 database. In this section we will examine the components of each of these systems and describe the actions performed by each component.

4.2.1. User Interface

Once the TACCNET system is installed and started, the users of the system do not interact directly with the main system processes. Users have a number of commands available to generate messages, examine messages, monitor transmissions, control system operation, and access the C^2 database.

The main TACCNET programs, *qms*, *msgproc*, and *server*, run in the background and invoke the other programs as needed. Their activities are unseen by the system users unless the system monitoring commands are used. An interactive console monitor program is provided so that the user may observe the coming and going of messages in the queues, monitor the various log files, and check the status of system tables.

4.2.2. Communications

The Communications system is made up of three cooperating programs: *qms*, *caller*, and *iocontrol*. These programs interact by placing messages in appropriate input and output queues for processing. Each queue is a directory in the UNIX file system and each message is a UNIX text file.

4.2.2.1. QMS: Queue Manager and Scheduler

The Queue Manager/Scheduler (*qms*) process handles the high level functions of the Communications system. Each TACCNET node that the local system can call is represented by a queue bearing the TACCNET name of the remote node. Messages to be sent to a node are placed in that node's queue. *qms* scans the queues periodically to look for messages to transmit. When it finds a queue with at least one message to be sent, it examines the system port table to see if a dialout port is available. If a port is available, a call to the desired node will be scheduled. It may find that a transmission is already in progress for the desired node. In that case, no call is necessary so it goes on to check the other queues.

In the event that a call is not in progress and a port is not available, *qms* chooses a course of action based on the priority of the outgoing message:

- If there are no priority messages in the queue for the remote site, *qms* will let the transmission wait until a port becomes available;
- If there are priority messages in the queue, *qms* will attempt to preempt a port engaged in routine transmission to some other node;
- If all ports are busy with priority transmissions, the new transmission will have to wait until a port becomes available.

The *qms* schedules a call to a remote node by marking a port as "in use" and invoking the *caller* process with the name of the port to use and the name of the node to call. The *qms* will continue to schedule calls until all ports are in use or there are no more queues containing messages for nodes that have not been scheduled for a call.

Before scanning the TACCNET node system queues for messages, *qms* always checks the **priority** queue for new priority messages. Any messages found there are moved to the appropriate system queues according to their destinations. The *qms* remembers which nodes had priority messages and attempts to immediately schedule calls to these nodes before scanning for routine messages in the system queues.

4.2.2.2. Caller: Dialing and Login Handling

The *caller* process is invoked by the *qms* with the name of a port to use and the name of a remote system to call. The *caller* validates the site name and looks up the telephone number for the site. It then opens the given port and issues commands to the modem to dial the phone. It monitors the modem's response codes to determine whether or not a connection is established. When the remote system answers the call, *caller* logs in to the system using its TACCNET node name as its userid and password.

The userid accounts defined for use by TACCNET are set up to invoke *iocontrol* as the shell. This means that the *iocontrol* process is automatically loaded on the remote system and attached to the port used to call into the system. Meanwhile, the originating system has established that the call was answered and has invoked its own copy of *iocontrol* in the "master" mode. The two *iocontrol* processes exchange synchronization information and transmission begins.

In the event that the call is not answered (i.e. phone busy or no answer), the *caller* receives a signal from the modem that no answer was detected. The *caller* then checks the remote node's entry in the site table to see if the

messages should be held or forwarded. The site table contains a record of the number of unsuccessful attempts to call the remote node and a record of the date and time of the last successful contact with the node.

If the number of failed calls is below a defined threshold, the site will be deemed "probably down" and messages will be held for a later attempt. No more calls will be scheduled for that node for a short period of time. If the number of failed calls exceeds the threshold, the site is declared "definitely down" and its messages are forwarded to a defined backup site for processing. No calls will be scheduled for a down site for a longer time period.† A courtesy copy of each forwarded message is kept for delivery to the down site when contact is reestablished.

Sites marked as "probably down" or "definitely down" are still called periodically. Once contact is reestablished, whether by calling the remote site or by receiving a call from that site, the site table is reset to indicate that the site is back up.

In the event that a call is answered and transmission begun, only to be interrupted in progress, *iocontrol* will return an error code to the *caller* and a similar course of action will be followed.

4.2.2.3. IOControl: Message Transmission

While a functional message transmission scheme was developed under the previous contract, this year's effort was aimed at revamping the system with the objectives of increasing both efficiency and data integrity. To reach these goals, the project designed a transmission protocol using the stop-and-wait technique of character-oriented data communications protocols.

The object of the technique is to provide data transparency and error detection, along with retransmission in case of errors. Defined in the protocol are two types of "packets": data packets and control packets, modeled after the BSC protocol. The information in a data packet is surrounded by a packet header and checksum trailer for error detection. The control information consists of single-byte acknowledgement and inquiry codes and does not contain checksum information. Embedded "escape" characters assure data transparency.

In the design of this communication system, independence and transparency of the system itself were key issues. The design does not rely heavily upon the exact environment in which the system is used. This makes it more general

† If the "down" site calls in on its own, it will be restored to the active list.

than the application for which it was developed. Although the implementation deviates slightly† from this goal, it provides a clean machine-to-machine interface that is suitable for the other facets of this project. The following is a general discussion of the system and the communications protocol driving it.

4.2.2.3.1. Overview

The *iocontrol* program manages communications between any two computers in the network, as outlined in previous sections. The *iocontrol* system is invoked for a queue corresponding to a particular node in the network and assumes a connection to that node. It operates as an independent entity in that it only interfaces with the *qms* (Queue Manager System) as it is invoked by the *qms*, or with the Message Processor as it places correctly-received messages in the input queue for the Message Processor. *iocontrol* operates in one of two modes: when *calling* another system or when *called by* another system. Such modes assign the calling system the role of "primary" and the called system the role of "secondary." The same program is loaded on both machines such that, initially, each program knows its respective role. Once connected, the two programs exchange synchronization and startup information. After the two machines have established and validated their connection, the primary system sends all files queued for the secondary system across the link. When all files have been sent, the two systems exchange roles, and the secondary system sends all its queued files to the primary system. This exchange continues until neither system has files to transmit, and the link is dropped.

4.2.2.3.2. Priority Messages

In order to demonstrate the capability of handling messages of different priority levels (requiring different classes of service), the TACCNET prototype system has two priority levels. Messages with a priority of "1" are regarded as "high priority" and are given fastest possible service, possibly at the expense of delayed or preempted transmission of other messages. Messages with a priority other than "1" are regarded as "routine" messages and are normally given first-in-first-out service. While there are only two priority levels in the prototype, it is easy to generalize to the case of several priority levels requiring different types of service.

In the event that a high-priority message appears in the queue to be sent to the secondary system, the primary will send that message as soon as it finishes sending the current message. If a priority message is destined for a different system and there are no more available lines, the affected system will

† The protocol contains some features specific to GCOS in order to deal with some limitations of that system.

immediately relinquish the line, interrupting transmission of any routine message as soon as it recognizes the condition. This is achieved by the appearance of a special file, ".Interrupt", in the current queue as soon as the priority message is recognized.

4.2.2.3.3. Message Header Packets

Each file transmitted is preceded by a special packet containing the name of the file on the sending system. This "header" packet achieves two goals: it allows for network-wide uniqueness of file names, and it gives the receiving system the option of rejecting a message file if it has already received a file bearing the same name. By rejecting messages based on their file names, no time is spent sending a file that will be redundant. All file names are unique throughout the network since each is composed partly of the originating system's network identification.

4.2.2.3.4. Protocol Implementation

To promote transmission integrity, files are transmitted as sequences of data packets, each containing a text block surrounded by control information indicating a packet number relative to the initiation of the current link, a checksum, and a flag indicating the end of block or message. The packet number and the checksum are represented in hexadecimal using ASCII characters so that control information contains no special codes. Control codes are used to mark the beginning and end of the text block and to indicate end-of-file or end-of-message. These are represented by their ASCII equivalents. Figure 4.1 is a diagram of the positional fields within a data packet.

Since some operating systems buffer input until a new line is received,‡ all packets are followed by a **CR** (Carriage Return) code. This extra character is not considered part of the protocol.

If a packet's checksum does not match the receiver's expectations, or if a packet is received improperly for any other reason, the receiving system transmits a control packet containing a **NAK** (Negative Acknowledgement) code. This causes the sender to retransmit the packet with a limit of five retries. After the retry limit has been reached, both systems abort assuming serious line problems. It is then the responsibility of the originator to reattempt the connection at a later time.

---

‡ The GCOS 400 operating system used by the DAS3 is such a system.

packet-number	STX	< - - t e x t - - >	DLE	ETB	cksum	DLE	EM

- The packet number is a two-byte value (in ASCII as described above).†
- The **STX** indicates the start of the text block.
- The text is a variable-length sequence of characters. The maximum block length is adjustable in the software. Each block will be filled to the maximum length except for the last block read from the file, which may be shorter. Any occurrences of the **DLE** character are doubled to permit transparency.
- To insure transparency, the **DLE** (Data-Link Escape) precedes all control information.
- The **ETB** position contains either an **ETB** or an **ETX**. **ETB** represents the end of a block, indicating that there are more blocks to come. **ETX** indicates that the block is the last block in the current file.
- The checksum is a four-byte value calculated from the characters in the text block.
- The **EM** (End of Message) indicates the end of the packet.

Figure 4.1 Data packet format.

Under normal conditions, all data packets are acknowledged with an **ACK** (Acknowledgement) code and the next packet, if any, is transmitted. Thus, the scheme is one of a stop-and-wait nature, where the window size is exactly 1 packet. This window size was chosen because the propagation delay for ground communications signals using telephone lines is typically negligible relative to the speed of transmission. Increasing the window size does not seem beneficial given the project's test-bed environment.

Control packets are used for synchronization and acknowledgement. They are three bytes long and have the form shown in Figure 4.2 below. The control-code field consists of a single-character control code as described above. The **DLE** and **EM** indicate the end of the packet. The following codes are currently used: **ENQ** (Enquire), **ACK**, **NAK**, and **CAN** (cancel). The **ENQ** control code is used to establish synchronization at the onset of communications. The **ACK** and **NAK** codes are used to acknowledge or reject packets. The **CAN** control code instructs the receiving system to cancel or abort a connection immediately.

control-code	DLE	EM

Figure 4.2 Control packet format.

† This limits the range of packet numbers to 0-255. Since the window size of the protocol is only 1 packet, the packet number is only used to distinguish between two consecutive packets. It is allowed to roll over as needed.

With the exception of control information, all data, including message file names, system names, and message text, are transmitted as data packets so that proper error detection can occur. This ensures that line distortions during transmission of any data will not cause file names, system names, or actual message text to be misinterpreted.

4.2.2.3.5. Binary Data Transmission

Provision for data transparency makes it possible to implement the transfer of eight-bit "binary" data between any two machines capable of processing eight data bits.† The *iocontrol* system treats all data as eight-bit quantities so that no distinction is made between a normal ASCII file transmission and a binary file transmission. This leaves the communication level completely transparent to higher-level functions, except in the case of machine architecture incompatibilities.

Any binary file placed in the queue for a system will be sent to that system *as is* with no conversions. No provisions were made to insure against failure due to machine incompatibilities. If the receiving system cannot accept eight data bits and the file contains any eight-bit values, the message will be considered in error, and both machines will abort.

The facility to transmit binary data was implemented in order to transfer data base files between two machines without any conversions. This allows two machines of the same type to directly interchange data bases to maintain back-ups in case of failures.

4.2.3. Message Processing

The Message Processor (*msgproc*) examines messages and takes actions based on the labeling and content of the messages. It periodically checks its input queue, **msgprocq**, for new message files to process. Messages are removed from that queue and placed in the appropriate system queue for delivery. Unreadable or invalid messages are placed in the queue **errorq**.

4.2.3.1. Initial Processing

msgproc first examines the message file name to determine the type of the message. The message file name is composed of three fields. The first field, the message type, is a one character field; it is followed by the node name, which may be up to six characters long.

---

† The Honeywell DPS-6 is excluded because the GCOS 400 operating system restricts all communications to seven-bit data words.

The third field is the sequence number, which is five hexadecimal digits. An example of a message name is:

Rxenair02ca4

which indicates that the message is a routine transmission from site **xenair.**

Valid message types are:

A	-	Administrative messages
C	-	Courtesy copies of messages
E	-	Invalid or undeliverable messages
F	-	Messages to be forwarded to an alternate node
H	-	Messages with invalid or unreadable headers
M	-	New messages just entering the system
N	-	Messages which received a NAK during transmission
P	-	Priority messages
R	-	Routine messages
S	-	Messages with invalid path specifications
U	-	User mail messages

When a new message is generated it is labeled as type 'M' and placed in the **msgprocq.** *msgproc* examines the message header to determine the priority and destination of the message. Priority messages are labeled type 'P' and placed in the **priority** queue. Routine messages are labeled type 'R' and placed in the system queue corresponding to the desired destination.

Destination paths are composed of a string of TACCNET node names separated by the exclamation point character (!). If the destination path contains more than one site name, the message is placed in the queue corresponding to the first name on the path list. If *msgproc* cannot understand the message header it will attempt to determine the nature of the problem and label the message accordingly (types S, H, and E) before placing the message in the **errorq.**

4.2.3.2. Message Forwarding

If the *caller* determines that a site is down and cannot be contacted, it delivers all messages queued for that site to the Message Processor for forwarding to a backup site. The file alternate site table contains a list of sites and their backups. A site may have more than one backup site defined; they are listed in order of preference.

The *caller* changes the type of each message to type 'F' and places it in the Message Processor input queue, **msgproc.** *msgproc* then determines the new destination by reading the message header and looking up the destination site in the alternate site file. A courtesy copy of the message is made and placed back in the original destination queue to be delivered when contact is reestablished. A new header containing the address of the chosen backup site is

added to the message and it is placed in the backup site's queue. The courtesy copy message is type 'C' and the forwarded message is now type 'R' or 'P' depending on its priority. Courtesy copy messages are never forwarded.

In the event that the backup site is also down, the process is repeated and the next backup site in the alternate site list will be tried. This will continue until the message is successfully delivered or the backup sites are exhausted. A courtesy copy will be created and enqueued for each down site.

4.2.3.3. User Mail

TACCNET may be used to send electronic mail to users at other TACCNET sites. The address of a user is simply the name of the site followed by the userid of the recipient. The two names must be separated by an exclamation point (!) as in any other path specification. For example, to send a message to user "frank" at TACCNET node "bravo" the address would be

bravo!frank

Received messages are placed in the Message Processor input queue. The Message Processor reads the destination path of the message and attempts to look it up in the path table. If the path is not defined, *msgproc* checks the system userid file to see if the message is for a user. If so, the message is reclassified type "U" and placed back in the input queue. When a user mail message is found in the queue, *msgproc* invokes the UNIX mail utility and sends the message to the named user.

There are two user id names which have special meaning to the Message Processor. The userid **net.adm** is an alias for the network administrator function of the Message Processor. Messages addressed to that user will be treated as network administrative messages to be processed by the system. The userid **server** is an alias for the C^2 Database Server process. Messages sent to that address will be placed in the server input queue for processing as JINTACCS messages for the C^2 database system.

4.2.3.4. Administrative Messages

The operating environment of the TACCNET system requires that it anticipate the frequent arrival and departure of nodes. Further, the network must be reconfigurable without human intervention at each node. To this end, a special class of message was created. The network administrative message (class "A") is used to add, delete, change, or examine the information in a node's site table. Administrative messages are also used to transfer files among TACCNET nodes and to request database backup and recovery operations.

The format of the administrative messages used to configure the network is simple. The first line of the message body is the administrative command to be performed. This may be **add**, **delete**, **examine**, or **change**. The remainder of the message must be a copy of the site table entry to be accessed. See section 4.3.4.1 for a description of the site table entry.

Administrative messages to transfer files or recover databases are slightly different. There are three commands: **transfer**, **save**, and **recover**.

The **transfer** command is used to request that a copy of a file be sent from one system to another. The format of the command is:

transfer <source path>!<source file name> <target path>!<target file name>

where **<source path>** and **<target path>** are the TACCNET paths of the source and target sites respectively, **<source file name>** is the UNIX path name of the file to be copied, and **<target file name>** is the UNIX path of the file to receive the copy.

If **<source path>** is omitted, the system uses the current site name. If **<target path>** is omitted, the system uses the originator's address path from the message header. If **<target file name>** is omitted, the system uses **<source file name>** as the default. The **<source file name>** must be supplied by the user.

The **save** command is used to tell the system to save the message contents in a named file. The format of the command is:

save <file name>

where **<file name>** is the UNIX path name to be given to the saved file. The **<file name>** must be supplied by the user. All contents of the message file following the header and the command line will be copied into the named file.

The **recover** command is used to invoke a database recovery operation. The command as the form:

recover <site> MM/DD/YY hh:mm:ss

where **<site>** is the TACCNET site for which the recovery is performed. It is followed by a date and time in the standard format shown above. Upon receiving a "recover" command the system searches the message archives for all messages sent to the named site after the given date. These messages are linked into the site queue to be retransmitted.

All network administrative messages must be addressed to **site!net.adm** where **site** is the name of the TACCNET node to be accessed. These messages are not really user mail messages. The **net.adm** userid is an alias for the Message

Processor.† When a network administrative message is found in the input queue, MSPPROC reads it and performs the desired function. A record of the originator and the action taken is kept in the Message Processor log file.

4.2.4. Database Operations

The main purpose of the TACCNET system is to transmit JINTACCS messages for Combat Service Support units. These messages are to be generated from and posted to the Command and Control (C^2) database. It is desirable to minimize human intervention in the processing of these messages. The Database Server (*server*) manages the interface between the TACCNET message system and the C^2 database. The Database Server in the TACCNET prototype uses the UNIFY relational database management system, which must be installed according to the manufacturer's instructions before the *server* can operate.

The *server* program examines messages in its input queue and updates the C^2 database or generates response messages as needed. Its input queue is the directory named **server** in the TACCNET master queue (usually **/usr/taccnet**). Messages addressed to **site!server** are placed in this queue by *msgproc* when they reach the final site in the destination path. The *server* examines this queue periodically for new messages to process. Messages must be in JINTACCS format with a valid TACCNET header. JINTACCS messages may be either update messages or request messages. An update message contains data to be placed in the C^2 database. A request message causes the *server* to get data from the C^2 database and generate the requested message addressed to the originator of the request.

4.2.4.1. The Message System

The primary purpose of the message system is to allow a user to create messages in JINTACCS format manually or automatically using two different databases, the Message Database and the Command and Control (C^2) Database.

The system may be invoked in either of two modes: manual or automatic. A message may be created manually by the user in response to system prompts displayed on the terminal. Alternatively, a message may be created and filled in automatically by the system using information from the C^2 database. The manual interface to the message system is called *jms*, for JINTACCS Message System. The automated interface (used primarily by programs or in pipes) is called *ams*, for Automated Message System. Both systems require the presence of the UNIFY relational database system and the TACCNET prototype message dictionary. The message dictionary is included in the C^2 database prototype

† Mail for the site administrator should be addressed to **root** or **taccnet**.

which comes with the TACCNET installation materials.

4.2.4.2. The Database

The database is composed of two logically independent parts: the Message Database and the C^2 Database. The Message Database contains the static parts of the JINTACCS messages: the message format information. The C^2 Database contains the variable parts of the messages transmitted or received. That is, information is automatically extracted from received messages and placed in the database; transmitted messages are constructed using information automatically extracted from the database.

The Message Database supplies a structural definition to a message so that a user can create a message in the predefined (JINTACCS) format from the terminal without knowing the specific details of the message format. The user only needs to know the desired message type†. Furthermore, the created message in JINTACCS format can be translated into a more readable form by providing a data field identifier for each data item. The Message Database, which is used as a message dictionary, was designed based on the definition of messages given in the ACCS COM Message Standards (See ACCS-A3-500-003, June 1984 for the standard format of Army Command and Control System (ACCS) character-oriented messages (COM)).

The C^2 Database contains information needed for command and control functions. It is also used to maintain all data objects in their most recent state while preserving the integrity of the database. Information from the C^2 database is transmitted to other units in JINTACCS format messages. Information from incoming JINTACCS messages is placed in the database.

Using these two databases, the message system is capable of automatic message creation and automated database management.

4.2.4.2.1. The Design of the Database

The nature of the JINTACCS message format is in essence hierarchical. The messages are composed of sets, which in turn contain fields. The fields are made up of data elements which may be composed of "chains" of sub-fields. The JINTACCS formats supply logical linkages between the defined sets, fields, and elements in the JINTACCS universe. This type of structure is hierarchical by nature and would usually dictate a hierarchical database system for most efficient management. However, the primary usage of the database system on the TACCS is for the manipulation of the Command and Control information

† We use the term *message type* to mean a message format specification such as POLLOC or SHORTSUP. A *message* is an instance of a *message type*.

used by each CSS unit. Information is transmitted from one unit to another via JINTACCS messages. Once entered, the message format data is relatively static, while the C^2 data changes frequently. System users perform queries on the C^2 database which often require relational operations. These considerations, coupled with the ease of use, power, and availability of relational database systems, convinced us to employ a relational system for both database tasks.

Having chosen the relational model for our database design, desirable properties of relation schemas were considered. For the purpose of eliminating problems of redundancy and anomalies (i.e. update, insertion, and deletion anomalies), the relational schemas have been normalized. The resulting database schema guarantees that the redundancy in relations is kept to a minimum and that anomaly problems do not occur when data objects are created, changed, or deleted from the database.

4.2.4.2.2. The Message Database

The database information for ACCS COM's is managed under the following relational database schema, which is the collection of five relation schemas:

msg (*mno, mtitle, malias*)
sets (*malias, sno, scat, setid*)
field (*fsetid, fno, fcat, fname, fcol, fdfi, fdui*)
dfis (*ddfi, ddui, dfdesc, dcolhdr, dlrj, dformat, dremarks*)
cdfis (*ccdfi, cno, cvdfi, cdui*)

Note that a relation schema is the set of attributes associated with a relation name, e.g. the first relation schema is the set of three attributes *mno*, *mtitle*, and *malias* associated with relation name **msg**.

An ACCS COM message is composed of a collection of sets, each of which again consists of fields, forming a hierarchical structure. Relation **msg** contains only the top-level information of every message for its identification. A message is assigned a message number (*mno*), a long message title (*mtitle*), and a short message name (*malias*), which all uniquely identify the message.

Relation **set** contains the specification fro all the different sets which may be used to build a message. The *malias* field tells which message te set belongs to and the *sno* field specifies the ordering of sets within a message. Each set is identified with its set identifier (*setid*) and classified into one of three occurrence categories(*scat*): mandatory (M), conditional (C), and optional (O) entries.

Level 3 relation **field** defines the collection of fields (*fname*) for each set (*fsetid*) and their attributes, i.e. the field number of each data field in the set (*fno*), the occurrence category (*fcat*), the data start column (*fcol*) in the case of columnar

sets, the DFI (*fdfi*) and the DUI (*fdui*).

A bottom level relation **dfis** shows the characteristics for each data field specified by a pair of DFI (*ddfi*) and DUI (*ddui*), i.e. field descriptor (*dfdesc*) and column header (*dcolhdr*) (if applicable) for each data field, left/right justification of data (*dlrj*), the number and type of characters in each data field (*dformat*), and the remarks (*dremarks*). Another low level relation **cdfis**, which will not be used for the current version, may be used later to identify, for each composite DFI (*CDFI*), its component elementary DFIs (*EDFI*).

4.2.4.2.3. The C^2 Database

The C^2 database schema is comprised of a number of relations to store various kinds of command and control information. The following are sample schemas for the messages S006, S026, and S034: relation **lsi** for S034 (SHORTSUP: Supply Shortages); relations **cas** and **scl** for S006 (CASSTATS: Casualty Information Report); and relations **pol** and **ptloc** for S026 (POLLOC: POL Locations).

lsi (*lunitid, lsicat, lsimod, lqty, lunit, lreq, lreqno*)
cas (*cunitid, csmos, ckia, cwia, cmia, cnbc*)
scl (*sscmos, sclass*)
pol (*ptname, pfol, ptyp, pqty, unit*)
ptloc (*pptname, ploc*)

Relation **lsi** shows, for every military unit (*lunitid*), the current stock for each logistical support item(LSI) (*lsicat*: LSI item and *lsimod*: LSI model).

Relation **cas** includes the casualty information in four categories: killed (*ckia*), wounded (*cwia*), missing (*cmia*), and non-battle casualties (*cnbc*). These are then broken down by specialty (*csmos*) for each military unit (*cunitid*).

Relation **scl**, for each specialty (*sscmos*), identifies the corresponding military personnel class (*sclass*).

Relation **pol** describes the current stock of Class III items (fuel, oil, and lubricants) for each location (*ptname*). The item class is given by *pfol*. The packaging unit is given by *unit*. The item type is given by *ptyp*. The quantity on hand is given by *pqty*.

Relation **ptloc** defines location (*ploc*) for each point name (*pptname*).

4.2.4.3. Implementation of the Database Schema

The database schema described above has been realized using the relational database management system UNIFY. The actual database design is shown in figures 4.3 and 4.4 below.

RECORD/FIELD	REF	TYPE	LEN	LONG NAME
lsi	100			lsi
*lsikey		COMB		lsikey
lunitid		STRING	24	UNIT_ID
lsicat		STRING	6	ITEM
lsimod		STRING	6	MODEL
lqty		STRING	4	QTY
lunit		STRING	3	UNIT
lreq		STRING	1	REQ
lreqno		STRING	16	REQ_NO
cas	100			casualties
*caskey		COMB		caskey
cunitid		STRING	24	UNIT_ID
cscmos		STRING	5	SC_MOS
ckia		STRING	4	ACTKIA
cwia		STRING	4	ACTWIA
cmia		STRING	4	ACTMIA
cnbc		STRING	4	ACTNBC
scl	100			class
*sscmos		STRING	5	SC_MOS
sclass		STRING	1	MILPERCL
pol	100			pol
*polkey		COMB		polkey
ptname	pptname	STRING	26	Name
ptyp		STRING	6	Type
pfol		STRING	1	Fuel_Oil_Lube
pqty		NUMERIC	5	Quantity
punit		STRING	3	Unit_of_Measure
ptloc	100			ptloc
*pptname		STRING	26	Name
ploc		STRING	20	Location

Figure 4.3 UNIFY schema for prototype Command and Control database.

The column headed RECORD/FIELD lists the record names left justified, with the field names indented underneath. The number following the record name is an estimate of the expected number of records of that type. An asterisk in front of a field name indicates that it is the primary key of the record. The column headed by REF is used to indicate the logical relationships that exist between the various files. The name of the primary key of another record goes in this column. The column headed by TYPE indicates the data type of the field. LEN is the display length of the field on screens and reports. LONG NAME is a more descriptive name used by SQL (UNIFY query language) and other system utilities. See the UNIFY Reference Manual for further explanation.

RECORD/FIELD	REF	TYPE	LEN	LONG NAME
msg	100			messages
*malias		STRING	10	ALIAS
mno		STRING	4	MESSAGE NO
mtitle		STRING	30	TITLE
sets	100			sets
*setkey		COMB		SET_KEY
salias	malias	STRING	10	ALIAS
sno		NUMERIC	2	SET_NUMBER
scat		STRING	3	CAT
setid		STRING	8	SET_IDENT
field	200			fields
*fldkey		COMB		FIELD_KEY
fsetid		STRING	8	SET_IDENT
fno		NUMERIC	2	FNO
fcat		STRING	3	CAT
fname		STRING	40	FIELD_NAME
fcol		STRING	3	SCOL_J
fdfl		STRING	5	DFI
fdui		STRING	3	DUI
dfls	300			dfls
*dflkey		COMB		DFI_KEY
ddfl		STRING	5	DFI
ddui		STRING	3	DUI
dfdesc		STRING	8	FLD_DESC
dcolhdr		STRING	24	COL_HEADER
dlrj		STRING	1	J
dformat		STRING	10	NO_TYPE
dremarks		STRING	20	REMARKS
cdfls	300			cdfls
*cdflkey		COMB		CDFI_KEY
ccdfl		STRING	5	CDFI
cno		NUMERIC	2	CNO
cedfl		STRING	5	EDFI
cdui		STRING	3	DUI

Figure 4.4 UNIFY Database Schema for message dictionary.

4.2.4.4. The Software

The message system programs (in executable files *jms* and *ams*) are written in the C programming language and include the UNIFY C-interface functions to access the database. The main purpose of *jms* is to allow a user to create a message interactively on terminal screen. The *ams* program is used to recieve data from a pipe and format it into a JINTACCS message. It is used by programs which query the C^2 database and generate messages automatically.

All of the information about message definitions is retrieved from the Message Database. A user may create a message simply by responding to a system-provided prompt for each data item. Messages are stored in multi-level doubly-linked lists while they are being constructed. This makes it possible to update a message dynamically during creation.

4.2.4.4.1. System Software Components

The system software has been implemented in modular fashion as well as in hierarchical structure to support many desirable programming concepts including modularity and portability. This is considered important since expansion is expected in later versions.

4.2.4.4.1. Message Handling Module

The major part of the system program has been written in a hierarchical fashion to conform to the structure of the messages. There are four major modules in the program. In order of invocation, they are:

message handler
set handler
field handler
dfi handler

The *message handler*, which is a part of the main routine, initiates the format for each message to be created by retrieving information from the message database. It also supplies the user prompts for input data, (e.g. message title) and begins to process interactively upon the user's response.

In the next stage, the *set handler* is called with the name of the message and retrieves the relevant collection of sets from the database. The *set handler* has three submodules, one for each type of set: linear, columnar, and free text. The *set handler* calls the *field handler* for each set to retrieve the collection of fields in the set.

The *field handler* calls the *dfi handler* to get the dfi (data field identifier) level information. The *dfi handler* supplies, for a particular dfi, the field descriptor or column header according to the type of the set, the data format, and left/right justification.

At each level in the message structure hierarchy, the user is asked to respond to system prompts for options or input data.

4.2.4.4.1. List Handling Module

In the list handling module, each message is held in the multi-level doubly-linked lists and modified dynamically at the user's command. The message-level, set-level, and field-level information is kept in level 0, level 1, and level 2 lists, respectively. A linked list is created at message creation time and its sub-level lists can be added, updated, and deleted on a real-time basis. This is of importance in the screen-oriented editor to be implemented in the next stage

of the system.

4.2.4.4.1. Message Display Module

The system gets the data from the linked lists and displays them on screen with indentations according to the level of lists. In later versions, this will be part of screen-oriented editor.

4.2.4.4.1. Message Format Converter

This system gets the message data from the linked lists and produces a message in JINTACCS format. Information from the Message Database is used to build a text file containing the message in JINTACCS format.

4.2.4.5. Getting Started with the Message System

To construct a message at the terminal using operator supplied data the **jtgen** command is used. The usage of this command is:

jtgen <*priority*> <*destination*>

where <*priority*> is a message priority and <*destination*> is a message destination identifier. This will start the program which will then prompt the user for all necessary information.

The system will ask for each item of information and then select the appropriate course of action. For example, each message may pertain to an Exercise or an Operation. The user will receive the prompt:

Enter EXER or OPER =>

The user's response may be one of the options or, since this set is not mandatory for all messages, a carriage return. If the operator chooses one of the options, the corresponding sequence of fields will be displayed. The default option, which is the message type other than EXERcise or OPERation, can be chosen by pressing the carriage return (CR) key without typing anything.

The message title will be prompted for as a field of the MSGID set

SET ID: MSGID
enter MESSAGE TYPE ===>

If the message title typed in is not a valid one, i.e. it is not in the message database, a warning message will be displayed to tell the user to reenter it. Once the system knows the name of the message to be created it can retrieve the information about that message from the Message Database and use it to generate the correct prompts.

The remaining sets and fields of the Initial Main Text (IMT) of the message will appear as prompts for the user. Once the IMT is complete, the system begins prompting for information specific to the chosen message. Information from the Message Database is used to determine the type of each set (linear, columnar, or free text) and the number and nature of the fields in the set. Appropriate prompts are presented using information from the data field identifier (dfi) for each field.

In general, fields can be skipped simply by pressing the CR key, except for mandatory fields. For a mandatory field, a warning message requires entry of data as prescribed. Another general rule is that any non-mandatory set can be terminated by typing double slashes (//) followed by a carriage return.

A created message is stored in a multi-level linked list. When it has been created successfully, the portion of the linked list for the message is displayed on screen for verification purposes. Finally, the message is converted into JINTACCS format for transmission.

It is desirable in many cases that a message be automatically created using C^2 database information without human intervention. Automatic message creation can be triggered by typing:

jtsend *<msgid>* *<priority>* *<destination>*

where *<msgid>* is a short message title and *<priority>* and *<destination>* are as described above.

4.2.4.6. Conclusions

In the development of the prototype system, a number of observations were made regarding the suitability of relational databases, ease of use of various systems, and future expansions of the system. These concerns are discussed below.

4.2.4.6.1. The Relational Model vs. the Hierarchical Model

For the implementation of the Message Database, a hierarchical system might be a better choice than a relational system for the following reasons:

- JINTACCS messages have been defined in hierarchical fashion. Most redundancies included in the relational database design could be reduced in the hierarchical database design model.
- The information in the Message Database is used primarily as a message dictionary. This information is relatively static, as message formats will not be added or modified on a daily basis.

- The information is always accessed in a hierarchical manner (fields within sets within messages) and there is no real need for the power of a relational query language.
- Speed of operation is desirable in the construction and processing of the messages.

This is not the case for the C^2 database:

- Command and control information, in general, may not be defined in a hierarchical fashion since any data objects required by CSS may be stored. These data items may be related to one another in many-to-many ways or may be totally unrelated.
- The data will be shipped around in JINTACCS messages between CSS units, subject to modification and aggregation on a daily basis.
- Operators may need to execute relational queries on the database to generate custom reports for decision support or message construction.
- The database should be easy to use since it is expected to be used by many different people at many different sites.

We conclude that, for the C^2 Database, the relational model is a better choice than other models.

It is desirable to minimize complexity in the system. The use of two separate database systems on the TACCS would increase complexity and might cause problems due to storage and memory limitations on the machine. Furthermore, the two different database models are not usually supported in a single commercial system. Thus, it was decided that the relational model be used for the whole system. The Message Database can be implemented in a relational system if a program is provided to use the message relations in the right hierarchical manner. It was necessary to add fields to some relations to determine precedence of sets within messages and fields within sets, since the concept of "item ordering" is not native to relational systems. Note that the amount of redundancy introduced in our relational database due to the use of the relational model is not too large.

4.2.4.6.2. Relational Database Management Systems

The selection of a suitable relational DBMS for the project required some time. In the end, we were limited by the lack of availability of systems which would run on all of the machines in the AIRMICS testbed. Early work on the message database was done using the INGRES relational database system on the VAX computer belonging to the School of Information and Computer Science at Georgia Tech. The first attempts at a prototype C^2 database were also made

using INGRES. When the UNIFY system was purchased and installed, the relational schemas developed under INGRES were ported over to UNIFY. In the process of porting the relational schemas and system software over to UNIFY, some observations about both systems were made.

INGRES provides Embedded QUEL (EQUEL), an embedded query language for use from user-written C programs. EQUEL statements, which are almost identical to QUEL, are preceded by the "##" and embedded in C programs. These statements are translated into standard C program statements by the EQUEL pre-processor. The translated statements define variables and execute functions from the INGRES library. This level is transparent to the programmer, who sees only the EQUEL statements. For its C language interface, UNIFY supplies only the library functions used to perform relational operations. The programmer must learn how to use these functions to accomplish his objectives. There is no high-level mechanism to perform relational operations in UNIFY from a user program.

UNIFY separates the C interface from its query language SQL. The UNIFY routines for database manipulation are provided as part of UNIFY host language functions, which includes the interface between user programs, CRT terminals, the printer, and the database. Since the number of database manipulation functions exceeds 40 and since these are not so well-defined as those in query languages, these may be harder to understand than embedded query languages. It is also a burden for the user to master another set of database manipulation functions as well as a query language.

We consider the availability of an embedded query language for the relational database system on the TACCS to be highly desirable. Such a feature facilitates development, extension, and maintenance of a database.

4.2.4.6.3. The Software

The system software has been implemented in modular structure as well as in hierarchical structure to support many desirable programming concepts including modularity and portability. These design features should be considered highly important since, in later versions, expansions and/or changes may be attempted to the existing modules to include more desirable features.
Also, changes in the message definitions should not affect the software at all since they will be made by the changes to the Message Database dictionary. Furthermore, the whole DBMS may be substituted for another new DBMS by changing only the code of database manager embedded in the message handling module.

Use of a multi-level linked list as the data structure for message storage provides flexibility. It maintains the message data dynamically during the editing process in screen-oriented fashion. The lists have double links between nodes both horizontally and vertically, i.e. they have double links between nodes in the same level as well as between the parent node and the child nodes in two different levels. The intra-level, doubly-linked list is capable of any kind of modification (i.e. insert, delete, and update) within the list, while the inter-level doubly-linked lists allow modification of the parent node for any child node and vice versa. Also, the C programming language is well-suited to the implementation of multi-level linked lists, especially when many pointers are employed.

4.3. TACCNET Installation and Operation

This chapter details the steps necessary to install the programs and data files associated with the TACCNET software and to maintain and operate the system once installed. The first section, entitled "Installation and Configuration", explains the steps necessary to bring a new computer into the network. The second section, "Initialization", lists the commands and their parameters for starting and stopping the network software from the TACCNET administrator's account. The section named "Monitoring" describes the process of detecting the arrival of new messages, the generation of outgoing messages, and the occurrence of errors at all levels of operation. "Maintenance" covers making modifications of system data files, queue clean-up procedures, and problem resolution. This chapter is concerned specifically with software implemented under the UNIX operating system.

In the following sections, a file pathname starting with an ellipsis ("...") indicates a path relative to the root of the TACCNET directory structure, named during installation. For example, if the software were installed in the directory **/usr/taccnet**, the file **.../tables/sites** would be referenced as **/usr/taccnet/tables/sites**. Do not type the ellipses in any path names.

4.3.1. Installation and Configuration

The TACCNET system can be installed easily on either an Onyx or an IBM PC/AT. Other UNIX machines might require some minor modifications, but these will not be discussed here. Before installation, the system manager must decide upon a root directory under which the queues will reside, a network name for the node being installed, and several other parameters.† He must also allocate at least one *dialin* port and at least one *dialout* port for connections to modems and phone lines. The first steps of the installation must be done from the system administration account (**root**). The following steps should be followed to install TACCNET on a new machine:

1. Create a root directory (e.g. /usr/taccnet) for TACCNET, change to that directory, and load the TACCNET system from diskette. (See **tar(1)** in the UNIX System III manual for information on loading from tape or diskette.)

2. Edit the file **/etc/group** and add a group named **taccnet** to that file. The actual group number is unimportant and may be set to any value that is valid for the local environment.

3. Change the group ownership of all files in the TACCNET directory structure to be **taccnet**. An example method for doing this is to change to the chosen root directory for the system and then to type the command, "**chgrp taccnet * */***".

---

† A single physical machine might take on the role of several network nodes if the TACCNET directory structure is installed under different root directories, each configured as a new node.

4. Place an entry in **/etc/passwd** for this node. Set its name to be the name chosen for the node being installed. Set its group number to be the same as that of the **taccnet** group created in step 2 above. Set its home directory to be the TACCNET root directory, and set its startup shell to be the the *iocontrol* program (specifically, ".../bin/iocontrol"). An example **/etc/passwd** entry is the following:

cosmos::113:100::/usr/taccnet:/usr/taccnet/bin/iocontrol

5. Duplicate the entry in **/etc/passwd** just added, adding the suffix "**.h**" to the login name. Change the startup shell for this login to be the special Honeywell interface, ".../bin/iocontrolh". The new entry should look something like the following:

cosmos.h::113:100::/usr/taccnet:/usr/taccnet/bin/iocontrolh

6. Change the password for both of these logins to be the name of the node. For example, node *cosmos* would have logins **cosmos** and **cosmos.h**, each with a password of **cosmos**. Some UNIX systems will not permit short passwords. If the password is rejected because it is too short, type it several times until it is accepted. If it is never accepted, choose a valid password, and inform all other nodes of the new password.

7. Create an administration account for use in editing configuration files and for starting and stopping the TACCNET system which has a group number the same as that of the **taccnet** group. You may create several such accounts, if required. Use this new account to complete the remaining steps.

8. Define the environment variable MASTERQ to be the full pathname of the TACCNET root directory you have chosen. Insert this definition in the shell initialization file for all accounts created in Step 7. (For *csh*, use the command "setenv MASTERQ ...", and place it in the the file **.cshrc** of the user's home directory. For *sh*, use the assignment "MASTERQ=..." followed by the command "export MASTERQ", placed in the file **.profile** of the user's home directory.)

9. Edit the file **.../tables/myname** and replace the text there with the name of the node being installed. The system uses this as its node name during conversations with other systems. Do the same with the file **.../bin/unitid**, as it is used by the *server* during automatic message generation.

10. Edit the file **.../tables/ports** to reflect the ports that have been set aside for *dialout* by the system. The structure of this file is explained in the section "Maintenance" below. There should be at least one port reserved for dialing out. Each *dialout* line must be connected to a Hayes Smartmodem. (It is unimportant which type of modem is used for *dialin*.)

11. If necessary, make changes to the files **.../tables/sites** and **.../tables/paths**, according to the instructions in "Maintenance" below, in order to describe the connections to other nodes in the network.

12. Make sure that there is at least one *dialin* port connected to the system, and that it is enabled for logins on a permanent basis. It is unimportant which type of modem is used for *dialin*.

13. Install the UNIFY Relational Database Management System according to the installation documentation for that system, and modify the database stored in **.../bin/unify.db** if desired to contain any current data for this site.

4.3.2. Initialization

Once the TACCNET system has been installed and all corresponding tables, modems, and phone lines have been correctly configured, the system will be operational. Typically, the user will wish to interact with the system (to start it, stop it, generate messages, monitor system log files, etc.) through the screen-oriented user interface, or *console* program. Information on this program can be found in the "User Interface" section of this document. The manual interface, which is command-driven and necessary in an environment that does not support cursor control or video displays, is described here. It is important to understand how to interact with TACCNET in this environment, since some elements of the screen-oriented interface require knowledge of the commands described below.

There are two modes in which the system operates: *one-pass* and *continuous operation*. In both modes, the Queue Manager System (*qms*) examines all possible system queues and invokes *callers* for each system which has messages pending. This sweep of system queues only occurs once in *one-pass* mode, where *continuous operation* mode causes the *qms* to sweep continuously, as long as the system is active. Similarly, the Message Processor (*msgproc*) and the Data Base Server (*server*) can either process the files in their respective queues in a single pass or continue to check their queues until instructed to stop, depending on the mode selected.

All three of the aforementioned programs should be running in *continuous operation* mode for the TACCNET system to be fully functional. Other configurations might be desired for testing or for causing a single transaction to occur under controlled circumstances. Shell files exist in the **.../bin** directory to aid in the starting and stopping of the TACCNET system, as well as other interactions with the system.
Within the **.../bin** directory, the user has access to the following commands:

cleanup - remove all files from all queues without processing.

unlock - remove all lock files.

taccnet - start the system (*qms, msgproc,* and *server*).

shutdown - stop all programs associated with the system.

These commands need no parameters, as they assume *continuous operation* mode and that the master (root) directory has been defined in the shell variable **MASTERQ**. The **cleanup** command will destroy data in all queues, causing unprocessed messages to be lost. This should only be done if the circumstances warrant such action. The **unlock** command should be typed just before starting the system or just after stopping the system, to make sure that

no lock files are still present. These lock files prohibit multiple invocations of TACCNET programs within the same directory structure. Typically, there will be no need for this command. It is only in the case of a complete system failure that it will be necessary. The **shutdown** command does not automatically remove these lock files.

To start the system under normal conditions, type **taccnet** from the **.../bin** directory.† Several other commands are available to selectively start and stop portions of the TACCNET system. These commands typically have the form:

command [-] [-dN].

The parameters are described for each command below. The optional dash (hyphen) after the command name sets the mode to *continuous operation.* Otherwise, the mode defaults to *one-pass.* The option "-dN" sets the optional *debug-level* for the system to the value of N. Values from 1 to 3 result in the printing of increasingly detailed debugging information in system log files. The debug option is not normally used. The following commands can be used to selectively start and stop portions of the system:

startsys - start the *qms* and *msgproc* programs using the root directory for the system as defined in the environment variable MASTERQ. Note that the *server* is **not** started with this command.

startserve - start the *server* program alone.

stopsys - terminate both *qms* and *msgproc.*

stopserve - terminate the *server.*

For example, to start the Data Base Server alone operating in *continuous operation* mode, one would type, "**startserve -**". To stop the server, one would simply type, "**stopserve**".

4.3.3. Monitoring

Most errors that occur during the execution of the **taccnet** system are handled automatically through retries. There are, however, some errors that cause data files not to be transmitted or database updates not to take place. These errors, along with recoverable errors and status information, are logged by all modules of the system to provide the operator with the ability to monitor the activity of the system.

---

† You should set your shell **path** environment variable to include the **.../bin** directory name so that you need not be in that directory to execute commands.

All modules log status and error information to files contained in the directory **.../log**. These files may be monitored using the **log** command, which is accessible from the **.../bin** directory. This command examines a file in the **.../log** directory, continuously displaying changes to it until the user presses the defined *intr* key (usually **DEL**) to interrupt the process.

See the section "User Interface" in this document for a way to monitor multiple log files in a screen-oriented, windowed fashion.

The following is a breakdown of the different files that can be monitored. The actual filenames in the directory are of the form **to log** below.

log qms - monitor the Queue Manager level of the system, displaying information on dispatching of *callers* to other systems.

log server - monitor the Database Server level, showing when updates and queries to the local database are made, as well as any errors associated therewith.

log msgproc - monitor the Message Processor by listing all messages processed by the system as they enter or prepare to leave this site.

log *sysname* - monitor the *iocontrol* program during its conversation with the remote system indicated by *sysname*. Indicate transmission errors, successful contact, termination, etc. *sysname* may be any valid remote system name defined in

Any attempt to monitor a system that has not yet created a log file will result in an error message saying that the log file cannot be opened. This means that there is currently no log file to examine. Either create the log file yourself, or wait and try the command later.

Monitoring log files can help the system administrator keep track of incoming files that might need manual processing. When new files enter the system, they are always placed in the queue for the Message Processor (**.../msgprocq**). The files are processed by *msgproc* if it is running, and placed in either a remote system's queue, such as **.../sysa**, the *server*'s queue (**.../serverq**), or the local system queue. If there is an error, files are placed in the directory **.../errorq**. Messages with bad headers or headers that contain sites which are unreachable are placed in the error queue. It is the TACCNET system administrator's duty to periodically examine these queues for possible problems and resolve them either by editing the files and moving them to their proper places, or by removing them.

4.3.4. Maintenance

Associated with the TACCNET software are several data files which contain information about remote system names, phone numbers, retry times, and local system parameters. Editing these files can change such parameters and enable the system administrator to keep configuration information current and consistent with changes at other sites. This section details the layout of these data files so that changes can be made to them using any text editor. It is suggested that changes be made only when the system is not active. The section entitled "User Interface" discusses one method for maintaining these configuration files.

All of these files should be examined before the system is started to make sure they are proper. If the system is aborted, there is a chance that some status information will be incorrect when the system is restarted. The shell-file command **cleanup**, mentioned previously, avoids most of these problems by restoring **.../tables/sites** and **.../tables/ports** from their respective **.save** files. Figure 4.5 is a diagram of the TACCNET system directory structure once installed in a given root directory. The following sections explain file formats and maintenance information for the files depicted in the diagram.

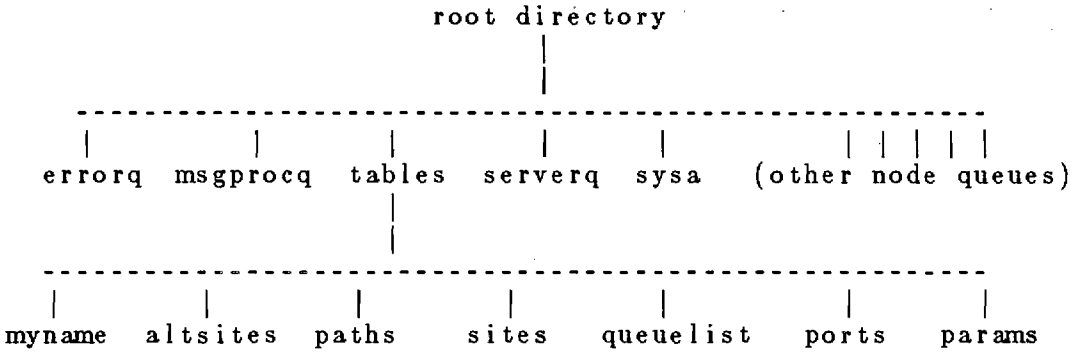


Figure 4.5 TACCNET directory structure

4.3.4.1. The Site Table

The file **.../tables/sites** contains a list of all sites that can be reached through this node in the network. For each site, information is stored about the site's phone number(s) and current status. The file is a standard ASCII text file containing several lines for each entry. The first line, which must begin with a colon (":"), specifies the site name, an up-flag, a retry-count, an absolute retry time in seconds (set and reset only by TACCNET software), and a system type flag to distinguish whether the system is UNIX ("U"), GCOS ("G"), or locally emulated ("E"). Last on this line is the password for access to the remote

system. This is usually the same as the system's name, except when operating system restrictions prohibit certain types of passwords. All of these fields are separated by spaces.

The remainder of an entry varies depending on the system type. If the system described is not locally emulated, then the remaining lines contain phone numbers in the order of preference, with each phone number on a line by itself. If the system type is "E" (locally emulated), then the next line describes the complete pathname on the local machine of the TACCNET root directory for the emulated node. Figure 4.6 shows a sample Site Table file.

```
:sysa 1 0 000000000 U sysa
3136
:xenair 1 0 000000000 E xenair
/tmp/taccnet
:honey 0 3 4168731203 G diffpwd
98944310
:sysd 1 0 000000000 U sysd
2414
:taccsb 2 2 593281304 U taccsb
8030
```

Figure 4.6 Site table file

Making changes to this file requires saving this file as both **.../tables/sites** and **.../tables/sites.save**. The second file is used as a backup of the default configuration in case it is ever desired to reset the various flags contained therein.

Site names cannot be any longer than eight characters and must all be unique.

The up-flag is composed of an ASCII character indicating the status of the remote system, as determined by communications failures. A value of "1" indicates that the corresponding node appears to be up. A value of "0" means that the node was determined to be down, based on the downed-node algorithm. A value of "2" indicates that the node appears down or busy, and that one or more retries will be attempted before the site is declared down.

The retry-count field will contain the ASCII representation of an integer designating the number of times the corresponding site has been called with no success. This is used to determine when a node should be marked as "down".

The retry time is the ASCII-encoded decimal representation of the last time the corresponding site was successfully contacted if the site appears to be active. Otherwise, if the site is down or being retried, this is the earliest time to attempt to reconnect to the site.

4.3.4.2. The Path Table

The file **.../tables/paths** contains routing information for systems than cannot be reached directly through this system. This information is used by the Message Generator (**genmsg**) to determine a path to a site. A path must be specified for each site in the Site Table, even if the site is directly accessible. The Path Table is composed of records indicating all paths to a particular site name, structured such that the site name is preceded by a colon (":"), and the associated paths are listed on a separate lines, each composed of site names separated by exclamation points ("!"), where the sites are traversed left to right.

Figure 4.7 is an example path table to illustrate the format.† If a site is directly accessible by this system, it should have an entry in the Path Table with its path set to its site name, with no exclamation points.

```
:sysa
sysa
xenair!sysa
:xenair
xenair
sysa!xenair
sysd!xenair
sysa!sysd!xenair
:honey
honey
:sysd
sysd
sysa!sysd
xenair!sysd
xenair!sysa!sysd
```

Figure 4.7 Path table file.

† Under the current implementation, the first path listed for a site is always taken as the "shortest" path.

4.3.4.3. The Port Table

The file **.../tables/ports** contains information about which I/O ports are to be used as *dialout* ports by the TACCNET system. It is used by the *qms* to determine which ports to use and to keep track of ports currently in use.

For each port defined as a *dialout* port, there is an entry in the Port Table. Each line of the table forms an entry, where the first character is a colon (":"), followed by the pathname of the output port (usually **/dev/ttynn**, where *nn* is a port number). After the port name are two fields indicating the name of the system currently being dialed through that port and the priority level of the transmission if the port is in use. If the port is not in use, the first of these fields is the ASCII string "**free**". The availability flag is an "**A**" when the port is "available". When the port is unavailable, it is either an "**R**" if the line is being used for "routine" messages, or a "**P**" if the line is being used to send one or more "priority" messages.

Figure 4.8 illustrates the format of the Port Table file. Note that the first entry indicates that port **/dev/tty00** is currently in use by a *caller* for system **xenair** for a routine message or set of messages. The second entry shows port **/dev/tty01** to be available for *dialout* to any system.

Since this file indicates the current status of all *dialout* ports, it might need to be reset if the system is aborted altogether, as mentioned above. Therefore, changes to this file should also be made to the file **.../tables/ports.save** so that it can serve as a backup file from which the **.../tables/ports** is reloaded before the system is restarted.

```
:/dev/tty00 xenair R
:/dev/tty01 free A
```

Figure 4.8 Port table file.

4.3.4.4. The Alternate Sites File

Whenever the Message Processor determines that a site is down, it attempts to send the enqueued messages for that site to its defined *backup* site. The file **.../tables/altsites** contains a list of all sites and their backups. The format of this file is similar to the others. Each site is preceded by a colon (":"), followed by a colon and any alternate sites in the order in which they should be attempted. Figure 4.9 shows a sample Alternate Sites file.

```
:sysa: xenair
:sysb: xenair
:sysd: sysa
:xenair: sysc
:sysc: dmmc1
```

Figure 4.9 Alternate sites file.

4.3.4.5. The Queue List

The file **.../tables/queuelist** contains a list of queues (directories) for use by some of the shell scripts to maintain and monitor the system. The shell script **cleanup**, which removes all files from the system queues after aborting the system, operates on the queues listed in this file. The shell script **qstat**, which examines the contents of the main system queues for display on the operator's console, also uses this file. The Queue List should contain the names of the major system queues, such as the site queues, message processor queue, priority queue, server queue, and error queue.

The Queue List file is a list of queue names, each on a new line, where a queue name is the simply the name of the node it represents. The complete path need not be specified, only the name of the queue relative to the TACCNET root directory (MASTERQ). Figure 4.10 illustrates the structure of this file.

```
priority
sysa
sysb
sysc
xenair
msgprocq
serverq
errorq
```

Figure 4.10 Queue list file.

4.3.4.6. The System Parameters File

All configurable system parameters (such as retry delays, packet lengths, and polling intervals) can be adjusted by entries in the System Parameters file, **.../tables/params**. This file is a standard ASCII file containing parameter-value pairs, each pair on a line by itself, associating the given value with the corresponding parameter.

System parameters, which invariably have defaults if not specified in this file, may be tuned by making changes to or adding to the System Parameters file. Because of the relationships among the elements of this system, the System Parameters file contains parameters for all components of the TACCNET system, including the *qms*, the *iocontrol* system, the *caller*, the *msgproc* system, and the *server*. Each of these programs examines this file upon startup, causing any modified parameters to be recognized throughout its execution. Making changes to this file thus requires shutting down the affected component and restarting it to incorporate the new parameters.

Figure 4.11 depicts an example System Parameters file, suggesting some typical values for the parameters listed. An explanation of all system parameters and their functions is given in Table 4.1 below.

```
blocklen 1024
blocklen(gcos) 64
timeout 8
timeout(gcos) 30
preemption 1
archiving 0
forwarding 1
maxcalls 4
maxretransmit 5
downdelay 200
retrydelay 60
qmspoll 30
msgprocpoll 15
serverpoll 120
```

Figure 4.11 System Parameters file.

Parameter	Range	Default	Action Taken
archiving	1 or 0	1	Archive(1) or discard(0) all messages after they have been processed on this system based upon this value.
blocklen	1-2000 (bytes)	140	When transmitting data to another UNIX-based TACCNET system, use packets containing this number of data bytes.
blocklen(gcos)	1-64 (bytes)	64	When transmitting to a Honeywell GCOS system, use packets containing this number of data bytes.
downdelay	0-32767 (sec)	180	If a remote node is determined to be down, do not try to call it again for this many seconds.
forwarding	1 or 0	1	Enable(1) or disable(0) alternate site re-routing of messages to downed sites, based upon this value.
maxcalls	1-100 (calls)	3	If a site cannot be contacted, do not try to call more than this number of times before declaring the corresponding remote node to be "down".
maxhangup	1-20 (tries)	5	Do not try to read the modem's response to a hangup command more than this number of times.
maxretransmit	1-20 (tries)	5	If a packet is transmitted in error, try to retransmit it this many times before declaring a transmission error.
msgprocpoll	0-32767 (sec)	30	Have the *msgproc* program wait for this many seconds between polls of **msgprocq**.
preemption	1 or 0	1	Allow(1) or disallow(0) interruption of transmission between packets. Interruption is always obeyed between transmission of complete messages.
qmspoll	0-32767 (sec)	60	Have the *qms* program wait for this many seconds between polls of the system queues in *continuous operation* mode.
retrydelay	0-32767 (sec)	60	If a remote node could not be reached, but is not considered "down", do not try to call it again for this many seconds.
serverpoll	0-32767 (sec)	60	Have the *server* program wait for this many seconds between polling of its queue for database update or query messages.
timeout	5-32767 (sec)	10	During communications with another UNIX-based TACCNET node, wait this many seconds before deciding that no response was made to a transmission.
timeout(gcos)	20-32767 (sec)	30	During communications with a Honeywell GCOS system, wait this many seconds before deciding that a response was not made to a transmission.

Table 4.1 Summary of TACCNET system parameters

4.3.5. User Interface

To permit a single user to operate, maintain, and interact with the TACCNET system, a general-purpose user interface, called *console*, is provided. This interface is screen-oriented and menu-driven, so it must be invoked from a video display terminal with cursor addressability. The *console* program will automatically operate within the TACCNET root directory, as defined either by the environment variable MASTERQ or by a command-line parameter at invocation. This program resides in the directory **.../bin.**

Essential to the User Interface is the organization of the terminal screen into windows. The terminal screen is divided into five windows: a **command menu** window, located at the bottom of the screen; two **queue monitor** windows, at the top of the screen; and two **status log** windows, in the center. These windows are *static* in that they occupy fixed positions on the screen, but all can be closed or opened at will by the user, with the exception of the **command menu** window. Figure 4.12 shows the relative positions of the windows on the standard *console* screen.

The **command menu** window displays the commands available to the user for modifying the state of the User Interface and for interacting with the TACCNET system. There are two command menu levels corresponding to the two functional modes. The user can move between alternative menus by pressing the *slash* key ("/"). The user can choose a menu item in one of two ways: he can type the single-digit number above the item he desires (without pressing any other keys), or he can select the item using the *space* and *backspace* keys to move between items and the *carriage-return* key to choose an item once selected. An explanation of each menu item (command) is given in the following subsection.

The **queue monitor** windows, which are located side-by-side at the top of the screen, constantly display the contents of the selected system queues. As message files appear and disappear from these queues, their contents are updated on the screen so that the user can observe the flow of messages through the system. There are only two such windows, so the user can change the queues to be monitored using the **open** command described below.

The **status log** windows are as wide as the screen and are positioned one atop the other. These windows constantly display the contents of specific system log files (see the section on "Monitoring") as requested by the user. Since there are only five lines in each of these windows, only the last three to four lines of a log file are displayed when such a window is first opened. After that, any new log information placed in a monitored file will be displayed in the window, and the contents will be scrolled when necessary to present the new status lines.

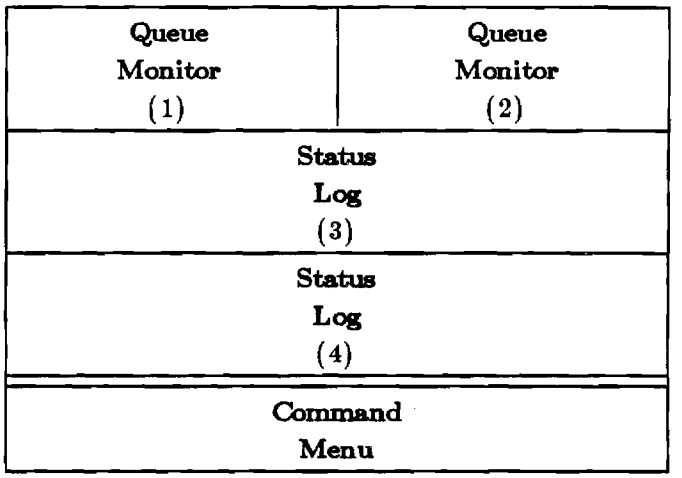


Figure 4.12 Layout of the *console* screen

Since there are two of these windows, the user can monitor status information pertaining to two systems or system functions at the same time. To change the names of the log files to be monitored in these windows, the user can execute the **open** command described below.

4.3.5.1. Invoking the User Interface

The *console* program can by invoked on a cursor-addressable terminal with the command "**console**". Several options can be applied to this command to vary the initial environment of the User Interface. These options, described below, can be entered in any order on the command line according to the following syntax:

console [*pathname*] [*-option* [*value*] ...]

The optional *pathname* tells the User Interface where the TACCNET root directory structure is located. This is in case the user wishes to monitor a TACCNET system which is situated in a directory other than the defined MASTERQ directory. If this parameter is not given, the *console* program will try to execute in the directory defined by the environment variable MASTERQ. If there is no such variable, the Interface will assume the default directory, "**/usr/taccnet**".

There are several possible *options* which can be given on the command line to set up the system for the user.† Associated with some options are *values*, which

† Typically, no options are required, since the *console* program will resume operations under the same configuration as the most recent execution.

can be names of files or system queues.
The following are the possible command-line options and their values for the *console* program:

-f [*conf*]	Pre-load the configuration file *conf* to define a saved system state. If *conf* is not specified, use the file "**.config**".
-i	Use only some *inverse video* functions; this terminal does not perform inverse video well.
-I	Use absolutely no *inverse video*; this terminal cannot support inverse video.
-n	Do not prompt the user to start the TACCNET system if it is down.
-s *sysname*	Pre-open windows 1 and 3 to monitor the system given by *sysname*.
-x	Do not use the previous system configuration, and do not automatically save the configuration after this session.

4.3.5.2. User Interface Menu Commands

While presenting the contents of system queues and log files in its four static windows, the *console* program displays a command menu at the bottom of the screen with which the user can control the operation of both the TACCNET system and the User Interface. Because a typical terminal screen is small and too many menu items might be confusing to the novice user, the set of possible user commands is divided between two menus. These menus are divided by function, so that one menu is for **TACCNET interaction** and the other is for **User Interface interaction.**

The **TACCNET interaction** menu contains commands that primarily affect and interact with the TACCNET system. The **User Interface interaction** menu avails the user of commands to tailor and manipulate the operations of the *console* program environment, including opening and closing windows and redrawing the screen. A description of the commands in each of these menus is given below. When the *console* program is first invoked, the **TACCNET interaction** menu is available at the bottom of the screen. The user can alternate between this menu and the other by simply pressing the slash key ("/"). See the previous subsection for information on selecting menu items to execute User Interface commands.

4.3.5.2.1. TACCNET Interaction Commands

The following is a list of the menu items available from the **TACCNET interaction** menu of the *console* user interface. Detailed usage of each command will not be given at this time.

admin	Edit a configuration file in **.../tables** using the editor defined by the UNIX environment variable "EDITOR".
command	Execute a single-line TACCNET command using the shell defined by the UNIX environment variable "SHELL". Any UNIX command can also be entered.
ports	Display the contents of the file **.../tables/ports**.
shell	Run an interactive shell using the shell defined in the UNIX environment variable "SHELL". (The *console* will return when the user exits the shell.)
sites	Display the contents of the file **.../tables/sites**.
status	Display graphic representation of queues and ports for system monitoring. Return to main menu by choosing "Status" option again.
exit	Terminate the User Interface, returning to the user's shell.

4.3.5.2.2. User Interface Interaction Commands

The following is a list of the menu items available from the **User Interface interaction** menu of the *console* program. These commands primarily affect the configuration of windows for monitoring the system.

clear	Erase the contents of a Status Log window so that new status information will appear alone.
close	Remove a window and its contents from the screen, marking it inactive. Use **open** to redefine a closed window.
get	Retrieve a configuration from a previously saved *console* session. Open and repaint windows according to this configuration.
new	Define a new system to be monitored in windows 1 and 3, such that the system's queue is displayed in window 1, and its status log is monitored in window 3. (This is a fast way to open two windows at once.)
open	Create a window (or modify an existing one) to monitor a system queue or status log file.
redraw	Refresh the screen in case of lost characters or display problems. Same as typing **ctrl-L** from either menu.
save	Write a file from the current configuration defining the windows which are open. Use **get** to retrieve this information later.
speed	Set the *polling speed* of the *console* in seconds. The polling speed is the amount of time the program waits between updates to open windows.
view	Examine the contents of a file in a queue associated with an open queue window. Uses the UNIX environment variable "VIEWER" as the file perusal program.
exit	Terminate the User Interface and return to the user's shell.

5. Recommendations and Conclusions

In this report we have discussed the major design considerations identified by the project team during the development of the prototype TACCNET system for the CSSCS. Some of these considerations are general and will apply to any information and communications systems developed for use in the CSSCS environment. Others are dependent on the specific functional requirements of the proposed system. It is critical to define and describe the complete set of functions to be performed by the system so that such design considerations may be discussed and resolved before the system is built.

One theme which has been present throughout our work and in this report is the importance of portability. At this time it appears likely that the CSSCS will run on a small computer using UNIX or one of its derivatives. The specific hardware and UNIX version are not yet defined. Since one of the prime features of UNIX and the C language is portability, it would be a serious mistake to write hardware or version dependent code in the development of CSSCS software. With a small amount of extra work one can develop code which is easily ported to any UNIX-derived system, regardless of the hardware chosen. The TACCNET system has been run successfully on seven different versions† of UNIX and on five different machines‡.

The second half of the document has been a description of the TACCNET prototype developed as part of the CSSCS AED program. This system is described to illustrate the main issues in CSSCS communications and is not to be considered as a fieldable system. It is a starting point for further development. It is not expected that the reader of this report will be fully able to understand and operate the TACCNET system. It will probably be necessary to study the system source code in order to fully understand the system. The TACCNET development team at Georgia Tech will be happy to answer any questions and provide any assistance necessary.

Appendix I to this report contains copies of viewgraphs used in a presentation about TACCNET. These will be helpful in understanding the system.

Appendix II contains high level data flow diagrams for the major TACCNET system components. These will aid in understanding the interactions of the TACCNET subsystems.

---

† The versions are: AT&T System Vr2; AT&T System III; IBM XENIX 1.0; SCO XENIX V; ONYX Onix V; PC/IX; DISTRIX 2.0

‡ The machines are: IBM PC/XT; IBM PC/AT; Burroughs B26; ONYX; AT&T 3b2

6. References

[1] ACCS-A3-400-004 (Interface Specification For) Maneuver Control Element Interface With Combat Service Support Control Element, August 1984.

[2] ACCS-A3-400-005 (Interface Specification For) Air Defense Control Element Interface With Combat Service Support Control Element, 9 November 1984.

[3] ACCS-A3-400-008 (Interface Specification For) Combat Service Support Control Element Interface With Intelligence/Electronic Warfare Control Element, 9 November 1984.

[4] ACCS-A3-400-009 (Interface Specification For) Combat Service Support Control Element Interface With Fire Support Control Element, 9 November 1984.

[5] ACCS-A3-500-003 Army Command and Control Systems Character Oriented Message Format Standards, June 84.

[6] ACCS-A3-500-003 Army Command and Control Systems Character Oriented Message Format Standards, Supplement 1, June 84.

[7] JINTACCS Technical Interface Design Plan, Volume VIII, Combat Service Support, October 1984.

[8] JINTACCS Technical Interface Design Plan, Volume VIII, Combat Service Support, Appendix E, COM Text Formatting Rules, June 1984.

[9] "Human Factors In The Display Of JINTACCS Messages", Tech. Rep. #USAISEC-RARA-85-2, D. Sharpe and A. Badre, 9 October 1985.

[10] "An Analysis of the Data Processing Requirements of CSSCS", Tech. Rep. #USAISEC-ASBG-85-1, M. Graham, 20 September 1985.

[11] CSSCS C^2 Information Requirements, 4 October 1982.

[12] "Of JINTACCS and JABBERWOCKS", Maj. J. Morrison and Maj. R. Case, *Signal*, vol. 38, no. 3, pp 55-58, November, 1983.

[13] "JINTACCS: getting the message across", Maj. A. Schenk, *Army Communicator*, Winter, 1986, pp 12-20.

[14] "How Secure is Secure?", G. Grossman, *UNIX Review*, August, 1986, pp 50-63.

[15] CSSCS Advanced Experimental Demonstrations, Final Technical Report for 1983-84, A. Jensen, W. Putnam, S. Goldberg, Georgia Institute of Technology, July, 1984.

[16] CSSCS Advanced Experimental Demonstrations, Final Technical Report for 1984-85, A. Jensen, W. Putnam, S. Goldberg, Georgia Institute of Technology, December, 1985.

Appendix I - TACCNET Presentation Materials

CSSCS Advanced Experimental Demonstrations

1983 - 1986

Development of a Tactical Army Command and Control Network

(TACCNET)

Objectives:

To examine issues related to information transfer among loosely-coupled, occasionally connected, heterogeneous, asynchronous networks of networks.

To develop a prototype Combat Service Support Computer System and a prototype Command and Control Database to be used in the exploration of CSS information processing requirements.

Approach

Iterative Refinement

- Develop expertise
- Design and build prototype
- Demonstrate capabilities
- Examine and refine

Experimental Demonstrations

- Advanced Experimental Demonstration (AED)
- Demonstrate capabilities
- Highlight issues
- Incorporate previous work
- Provide recommendations for future work

End Product

- Working, portable, full-featured prototype
- Documentation of issues and concerns
- Specification for interim, fieldable system

Plans

- Finalize and document TACCNET prototype

- Explore JINTACCS message processing issues
  - JINTACCS grammar or definition language
  - Functional definition of messages
  - Message processing tools

- Design Command and Control Database
  - Top-down design approach
  - Analysis of intended usage/user requirements
  - Determine structure and content from usage requirements
  - Interface with JINTACCS

- Convert to ADA

Accomplishments

- Information transfer among network of widely differing machines (S/1, CDC, IBM 4300, Vax/Unix, PC) over a variety of links (3780, BISYNC, asynchronous dialup, token ring)
- Prototype TACCNET using PC/Unix and Honeywell/GCOS featuring automated routing, failure detection, and rerouting
- Extended TACCNET featuring database backup and recovery, file transfer, message processing, and screen-oriented user interface
- C^2 Database with automated JINTACCS interface
- JINTACCS screen-oriented, automated composition tool
- Source-level system portability
- Simulation of CSSCS network (SLAM)

Status

- Completed and installed a well-defined, fully featured prototype TACCNET for CSSCS environment

- Developing detailed specification of TACCNET system design and implementation

- Beginning first year of two-year investigation of automated JINTACCS message processing

CSSCS Environment

- Nodes subject to catastrophic failures
- Nodes are physically mobile but logically static
- Frequent, expected, but unpredictable reconfiguration
- Nodes are loosely coupled and occasionally connected
- Machines are physically small (microcomputers)
- Communications links are undetermined (media transparency required)
- On-demand communication links
- Time constraints/priority messages
- Most messages in JINTACCS format
- Well-defined hierarchy of nodes

Why TACCNET?

Why not use *uucp*, Kermit, or other widely available data transfer systems?

- No existing product conforms to CSSCS environmental constraints (rerouting, failure management, JINTACCS message handling, time constraints, priority messages, observance of node hierarchy, etc.)

Why UNIX?

Advantages:

- Availability on many different architectures
- Portability (many machines in desired size class)
- Good environment for software development
- Convenient file structure (i.e., directories as queues)
- Process control and inter-process communication
- Multi-user, multi-tasking system
- Standard, portable high-level language (C)

Disadvantages:

- Unix is a "moving boundary"
- Not "friendly" to naive user
- Many variants in distribution
- Missing features (such as file locking)
- Security

TACCNET Capabilities and Functions

- Heterogeneous communications
  - Media transparency
  - Error-detecting protocol with retransmission
  - Logging of connections, errors, and message transfers
  - Bidirectional, on-demand links
  - Tunable parameters (i.e., speed, packet size, retry delays)
  - Remote system identification
  - Broadcast and message rejection
  - Failure detection and management
- File transfer
- Electronic mail
- JINTACCS to and from C2 database
- Automated JINTACCS message composition interface
- Distributed C2 database backup and recovery
- Single machine emulation of multiple nodes
- Network management functions via messages
- Dynamic network configuration
- Screen-oriented, menu-driven user interface
- Message forwarding/holding
- Store-and-forward message transfer
- Automatic routing via shortest path

TACCNET

Objectives:

- Pass JINTACCS messages
- Detect and handle failures
- Automatic (re)routing
- Dynamic network configuration
- Messages to and from C^2 database
- Database backup and recovery
- User interface
- JINTACCS message composition aids

TACCNET

Constraints:

- Ordinary telephone lines
- 1200 bps transmission rate
- Auto-dial / auto-answer modems
- Media transparency
- TACCS/UNIX, DAS3/GCOS

TACCNET

Additional Features:

- Error detection and recovery
- Data transparency
- Binary data transfer
- Store and forward capability
- Priority message scheduling
- On-line JINTACCS message dictionary
- Password security
- File transfer
- Electronic mail
- Multiple node emulation
- Tunable system parameters
- Portability (all code written in C language)
- Menu-driven system interface

TACCNET Composition

Communications

qms	-	scheduling
caller	-	connections
iocontrol	-	transmission

Message Generation and Processing

genmsg	-	generation
msgproc	-	processing

Database Operations

jms	-	message composition
server	-	messages into C^2 DB
build	-	messages from C^2 DB

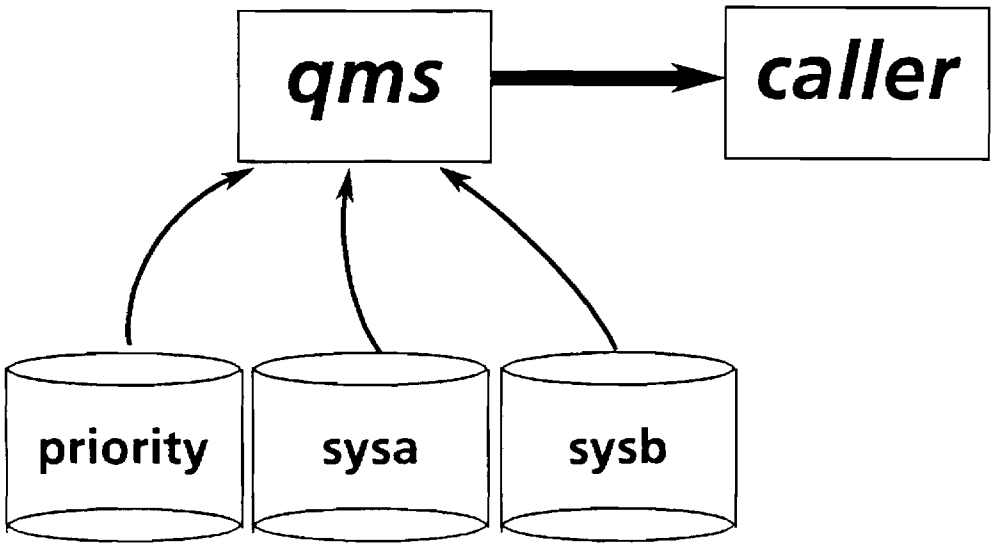
User Interface

console	-	system administration

Communications

qms

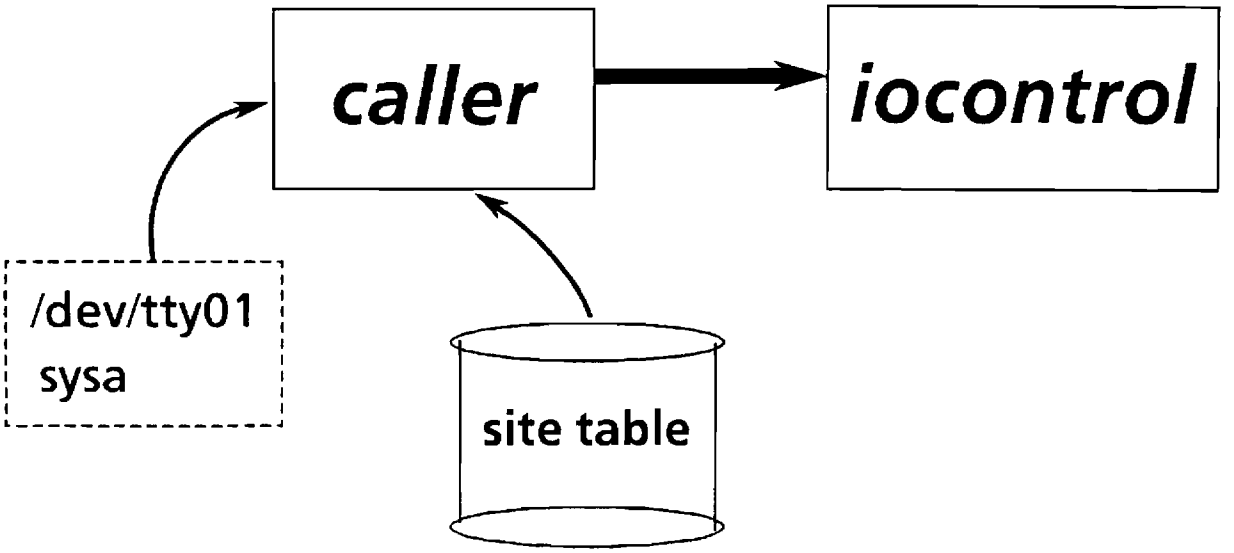
- Runs in background (sleep or cron)
- Scans priority queue first, then system queues in order taken from site table
- Invokes *caller*
- Handles preemption for priority messages



Communications

caller

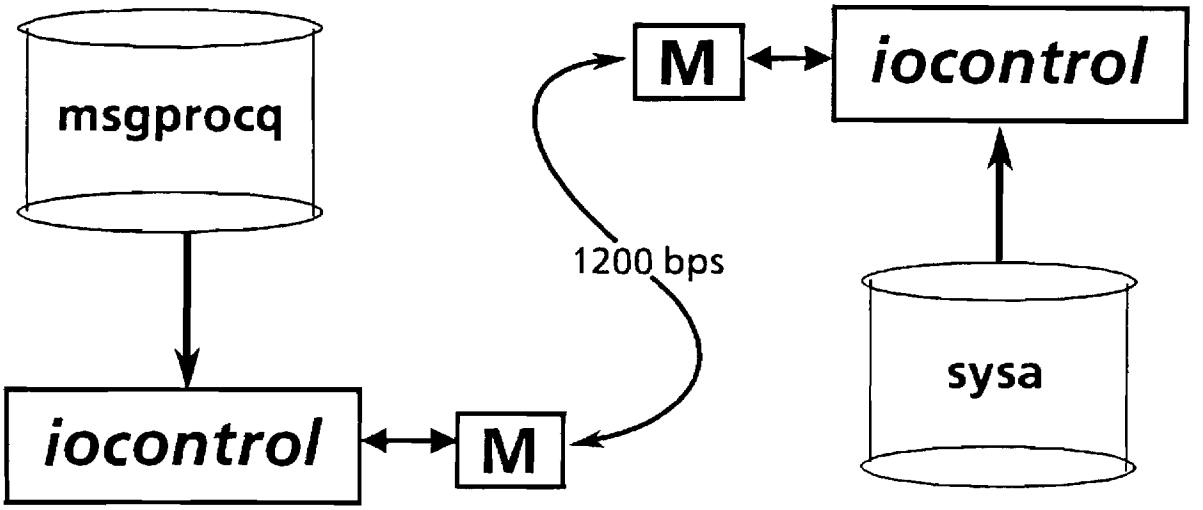
- Validates site information
- Establishes connection
- Handles connection failures
- Starts *iocontrol* process
- Handles transmission failures
- Releases port and system queue
- Updates site table



Communications

iocontrol

- Transmits / receives files
- Gets files from system queue
- Puts files into message processor queue
- Error detection / correction
- Priority preemption
- Data transparency / binary data transfer



Communications

Transmission Protocol

- Similar to BSC (Stop & Wait, Window = 1)
- Data packets / control packets
- ASCII control codes

  DLE - Data Link Escape
  STX - Start of Text
  ETX - End of Text
  ETB - End Text Block
  EM - End of Message
  EOT - End of Transmission
  ACK - Acknowledge
  NAK - Negative Acknowledge
  CAN - Cancel

- Packets "punctuated" with **CR** for GCOS

Communications

Packet Formats

- Data Packets

  12 bytes for frame, text block is variable length (tunable parameter)

- Control Packets

  Always 4 bytes

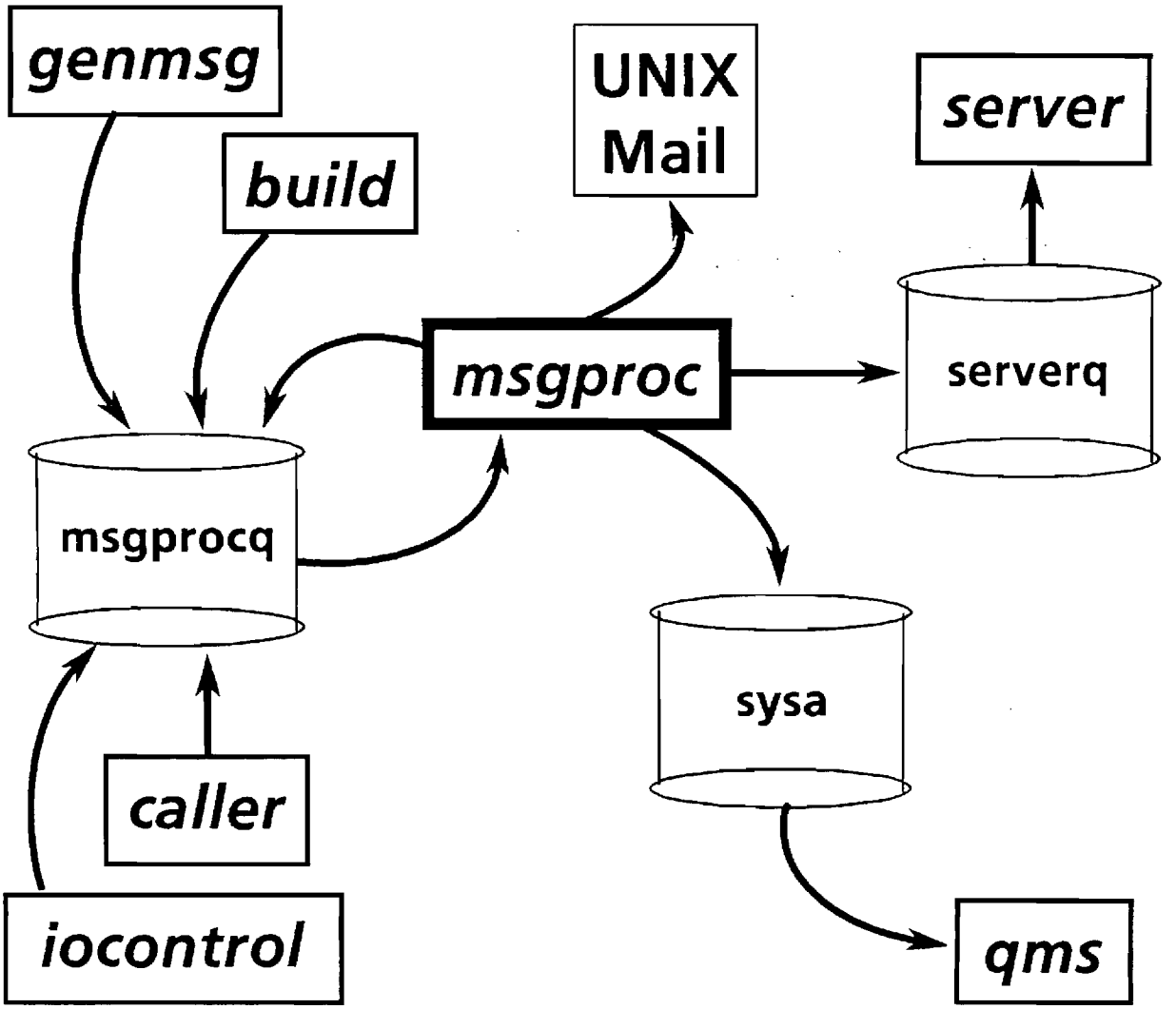
Data packet format:

Control packet format:

Message Processing

msgproc

- All messages pass through msgproc
- Processing is based on message type
- Routing is based on message header



Message Processing

msgproc

- Message file name indicates type

  Format:

  T*sysname*XXXXX

  Where:

  T = message type

  sysname = originating node

  XXXXX = hex timestamp

- Valid message types:

  A - Administrative
  C - Courtesy copy
  E - Undeliverable
  H - Bad header
  M - New message
  N - Rejected message
  P - Priority message
  R - Routine message
  S - Invalid path
  U - User mail

Message Processing

msgproc

- Routing based on message header

  >priority [c-flag]
  >source path
  >destination path

- Path format:

  site[!site!site...][!user]

  Where:

  site is a valid node ID;

  user is either "net.adm", "server", or a valid user on the node

- Message may have multiple headers (first is current)

Database Design

Message Dictionary

Hierarchical system may be preferred:

- JINTACCS message defined in hierarchical fashion
- Message database is primarily used as (static) message dictionary
- Information accessed hierarchically (fields within sets within messages)
- Speed of operation is desirable
- No need for relational query capability
- Reduced redundancy

Database Design

Command and Control (C^2)

Relational may be preferred:

- Information **not** defined in hierarchical manner
- Data items may be related in *many-to-many* fashion
- Database contents are dynamic
- Relational query capability desirable
- User-oriented interface necessary

First cut: extract from JINTACCS messages and normalize

Database Operations

jms

- User interface for message composition
- Uses message dictionary to build prompt panels
- Builds message in JINTACCS format and submits to *msgproc*
- User can review, edit, or save message during composition
- New messages are easily added for automated composition assistance

Database Operations

server (automated message posting)

- Reads JINTACCS messages
- Extracts variable data into file
- Builds UNIFY update in file
- Calls UNIFY to enter data update
- Old data overwritten by new

Limitations:

- C source module for each message
- Needs embedded query language

Database Operations

build (automated message generation)

- Given: message ID
  destination
  priority
- Executes UNIFY query, capturing data into file
- Reads data from file, puts into JINTACCS template in new file
- Submits new message file to *msgproc* for input to system
- Table-driven; does not use message dictionary

Limitations:

- C source module for each message
- Need embedded query language

Database Operations

Backup and Recovery

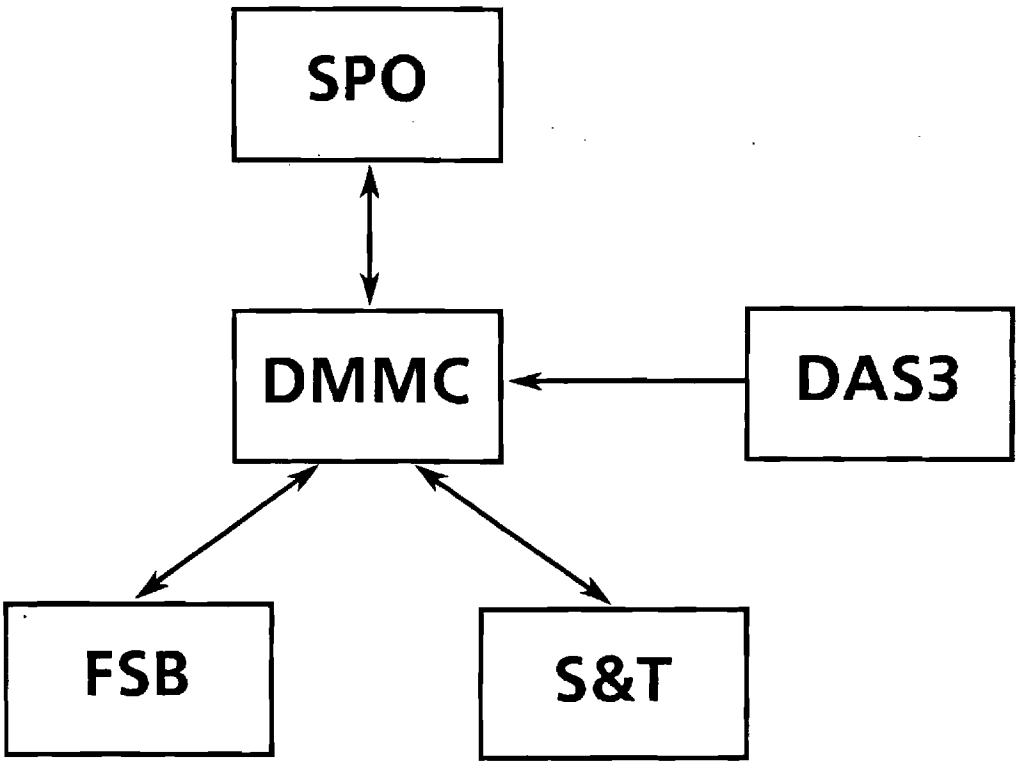
- Uses TACCNET to copy snapshot of database to remote node(s)
- Broadcast messages used to retrieve messages sent after snapshot
- Backup and recovery procedures initiated by user or *cron*

To recover from a failure:

- The snapshot is retrieved from one of the remote backup sites,
- A broadcast message is sent to the network requesting retransmission of all messages sent to the failed site after the snapshot was made.

Network Simulation

- SLAM simulation on CDC Cyber
- Low, moderate, and high traffic
- Assume basic network segment
- 1 - 3 ports (*dialin* and *dialout*)

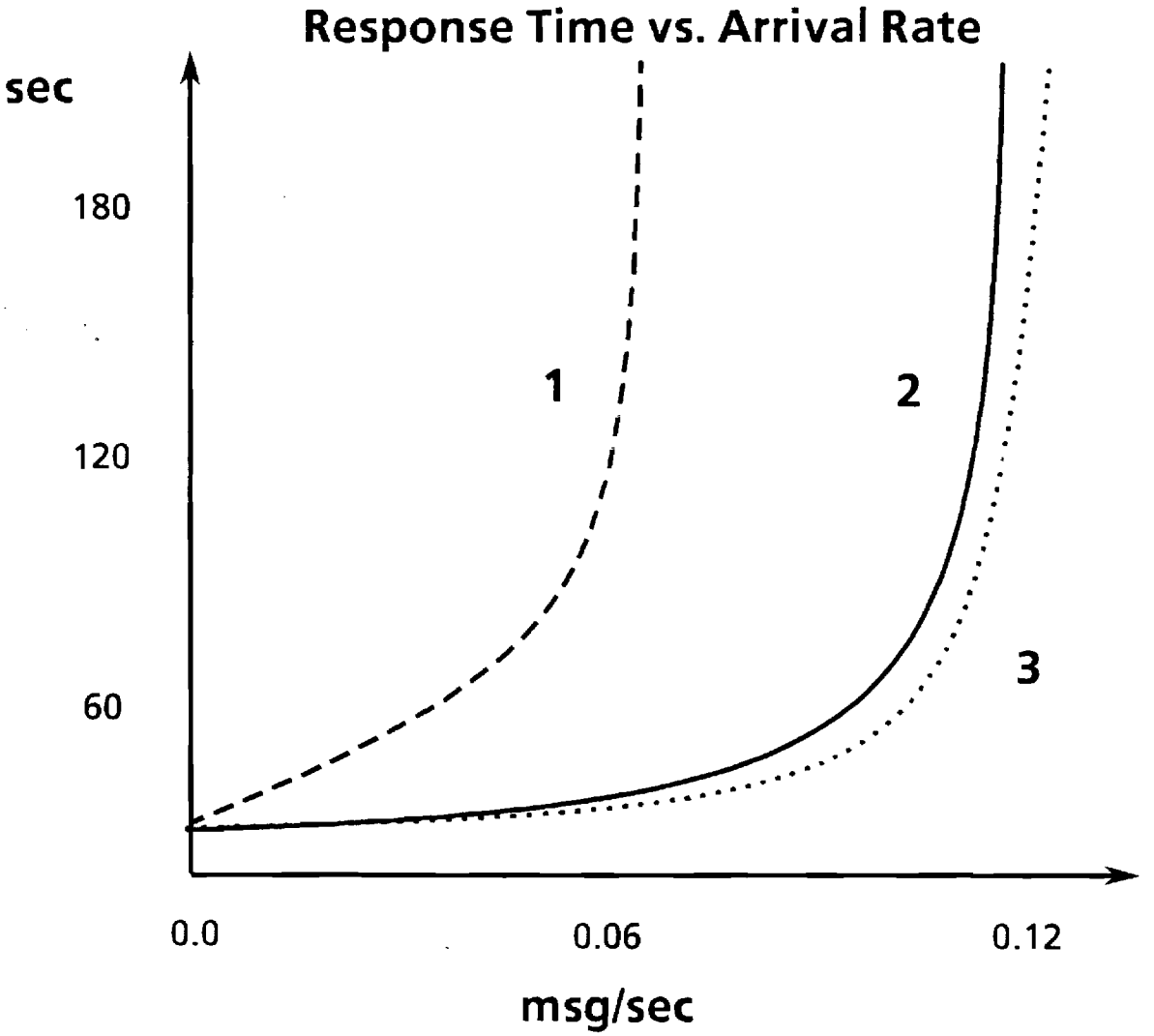


Basic network segment

Network Simulation

Results

- Bottleneck at DMMC
- 2 *dialin*, 2 *dialout* gives best results
- 1 *dialin*, 1 *dialout* is OK for leaf nodes



Further Investigation

JINTACCS processing

Message grammar, parsing, functional description

Security

Data and network security

Expert Systems

Structured format with ambiguities; message processing

Voice Technology

Voice / data interface for composition and display

User Interface / Tools

Edit / display messages during creation; insulation from JINTACCS

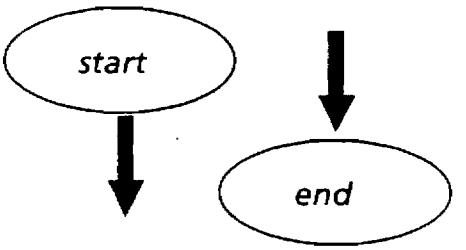
Distributed Database

Consistency, redudancy, fault tolerance

Appendix II - TACCNET Data Flow Diagrams

Legend

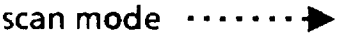
Module entry and exit points



Flow of program control

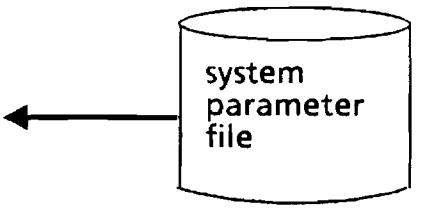
Flow of data to/from disk files or program modules

Command line arguments from user or parent program



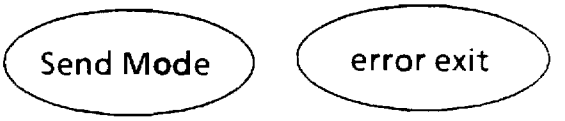
Program control statements

Flow of data to/from tables or messages (files)

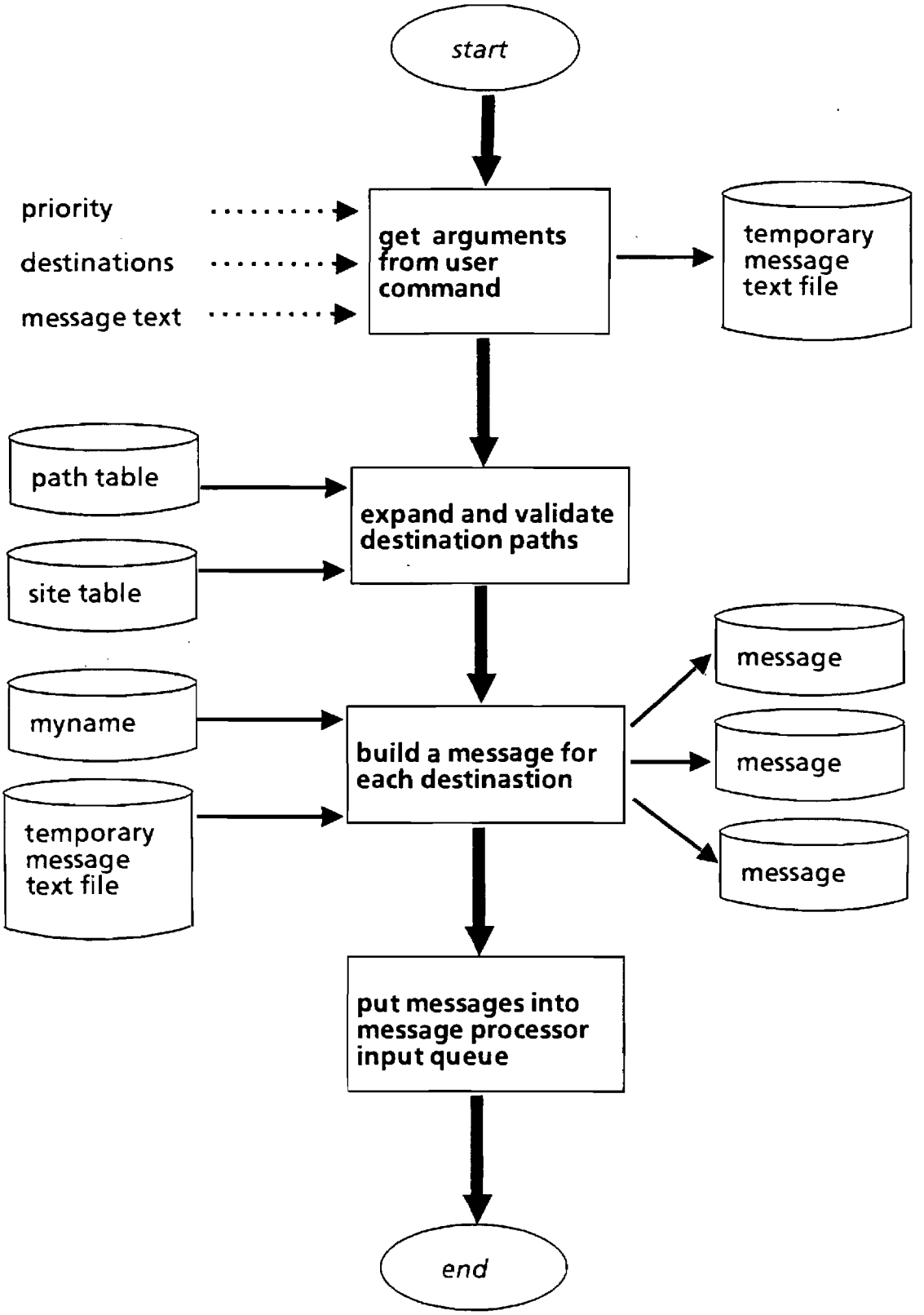


Flow of data (files) to/from directories (queues)

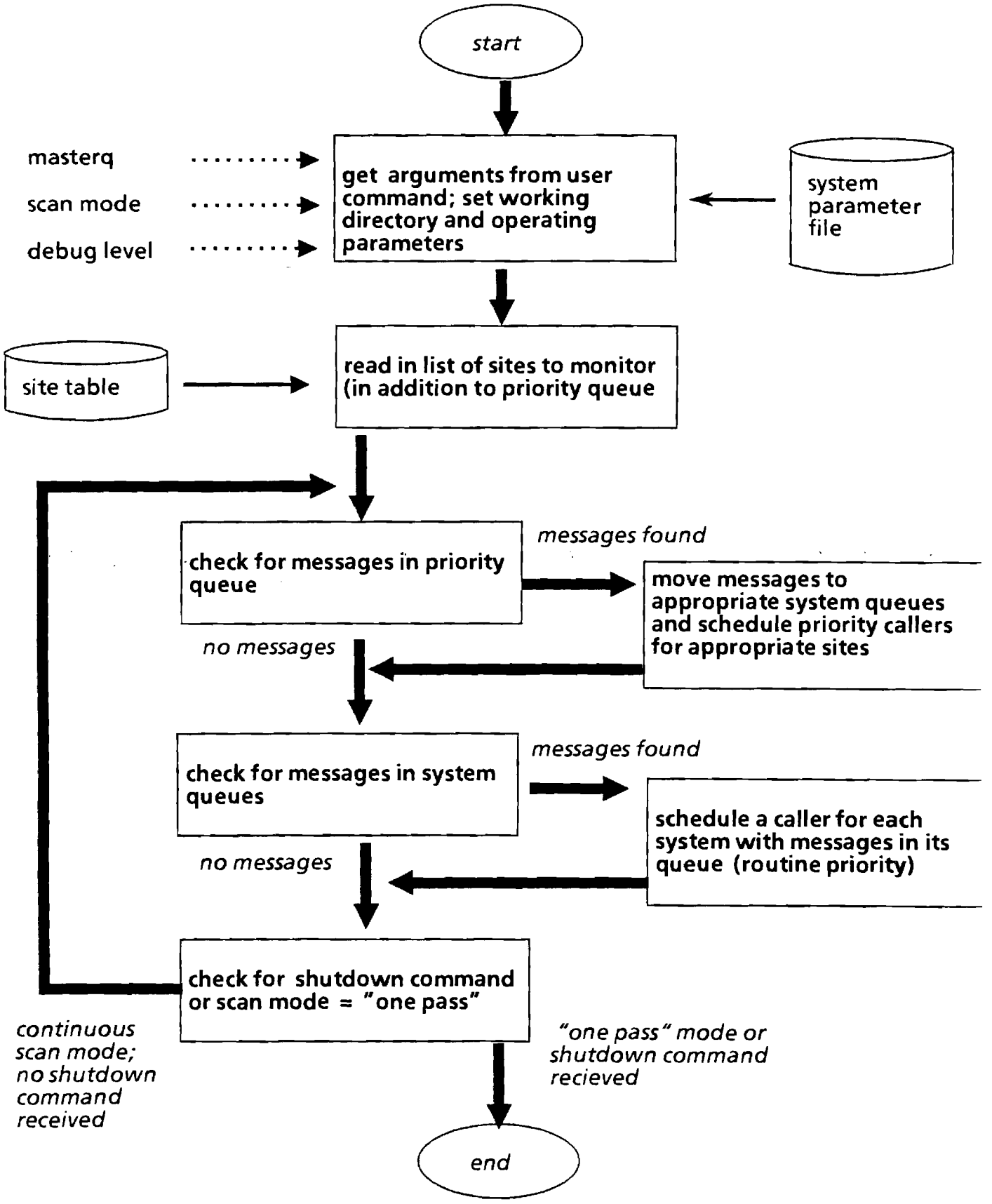
Special entry and exit points for errors and procedure calls



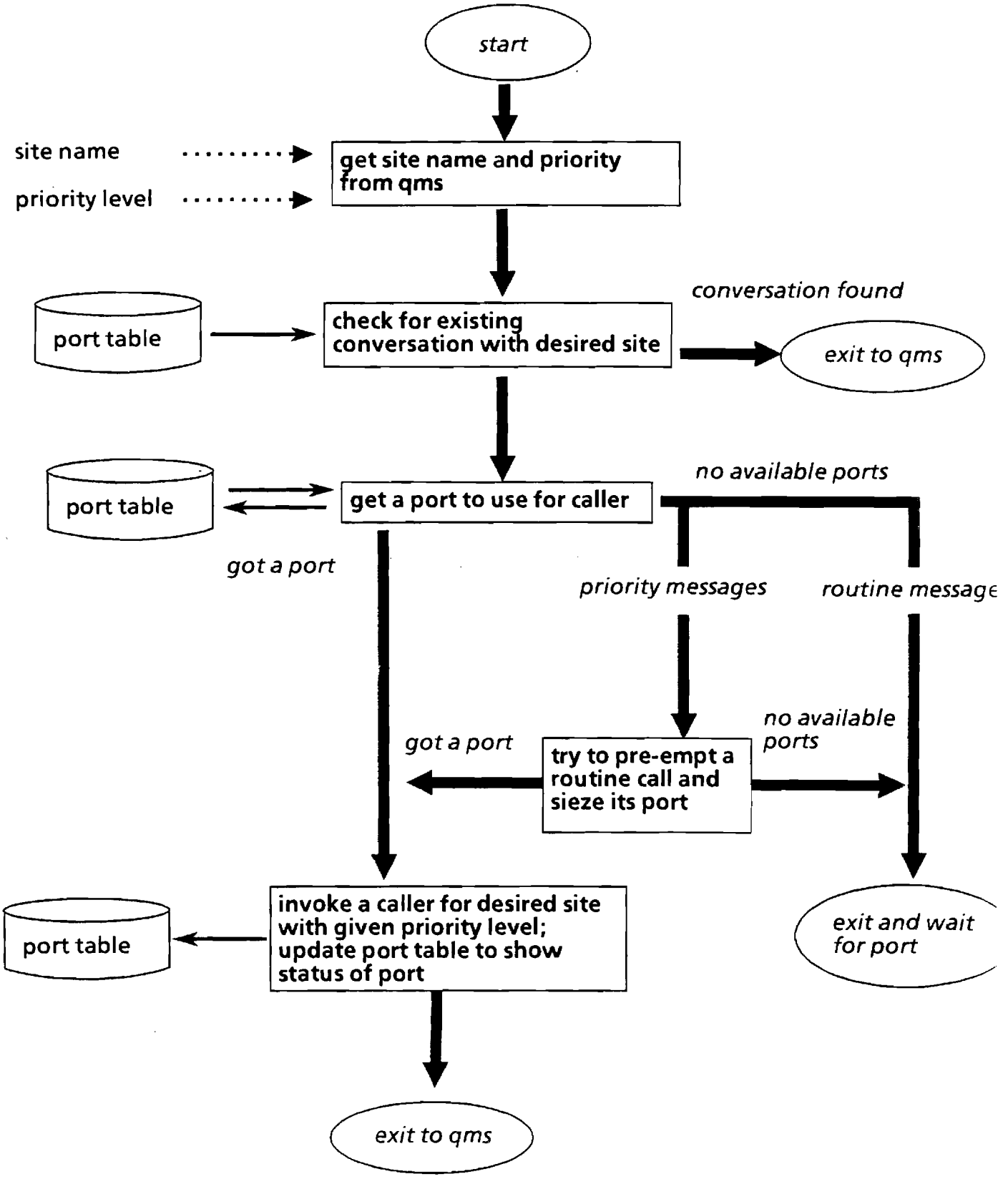
Genmsg


start
priority
destinations
message text
get arguments from user command
temporary message text file
path table
site table
expand and validate destination paths
myname
temporary message text file
build a message for each destinastion
message
message
message
put messages into message processor input queue
end

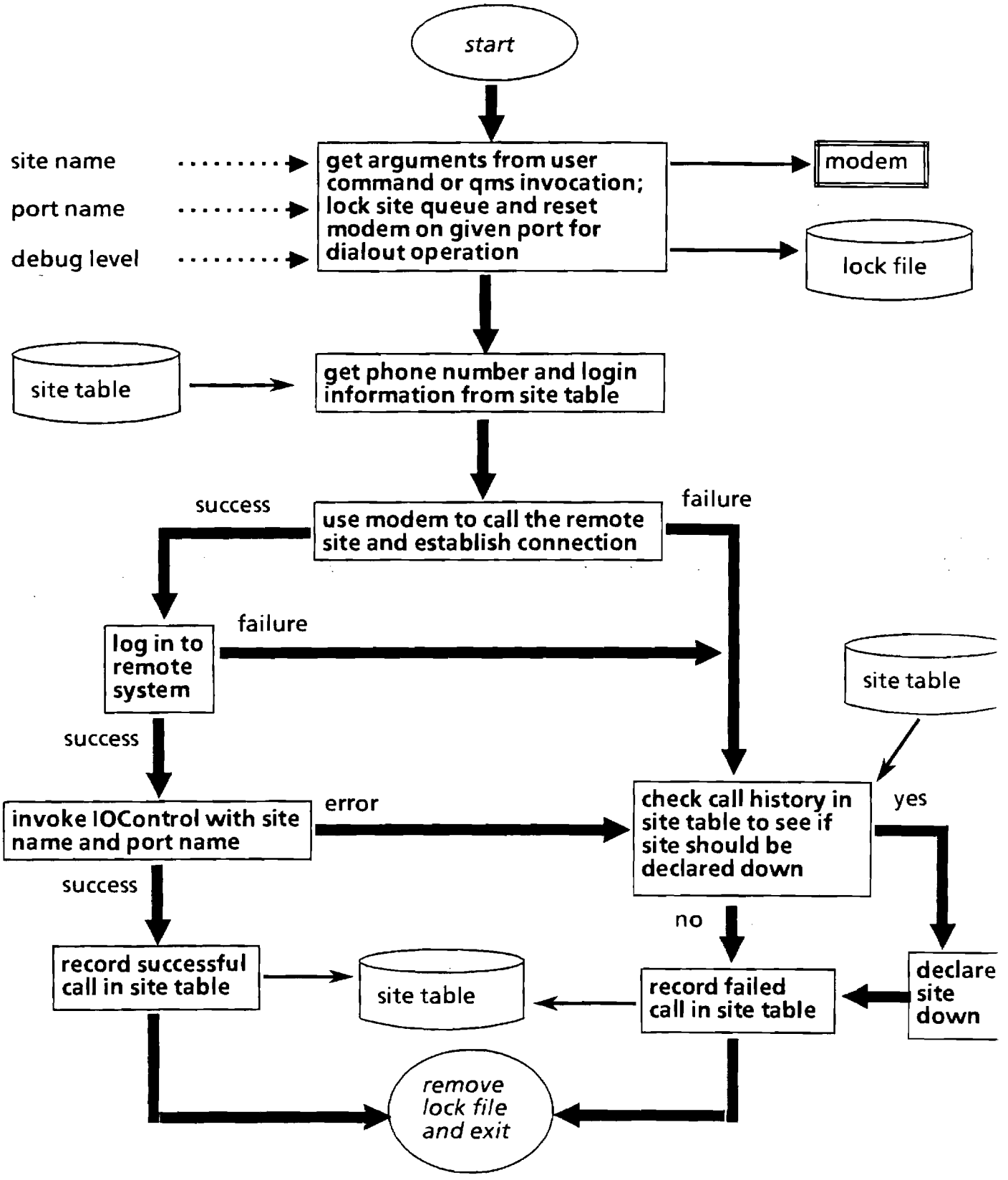
QMS


start
masterq
scan mode
debug level
get arguments from user command; set working directory and operating parameters
system parameter file
site table
read in list of sites to monitor (in addition to priority queue
check for messages in priority queue
messages found
move messages to appropriate system queues and schedule priority callers for appropriate sites
no messages
check for messages in system queues
messages found
schedule a caller for each system with messages in its queue (routine priority)
no messages
check for shutdown command or scan mode = "one pass"
continuous scan mode; no shutdown command received
"one pass" mode or shutdown command recieved
end

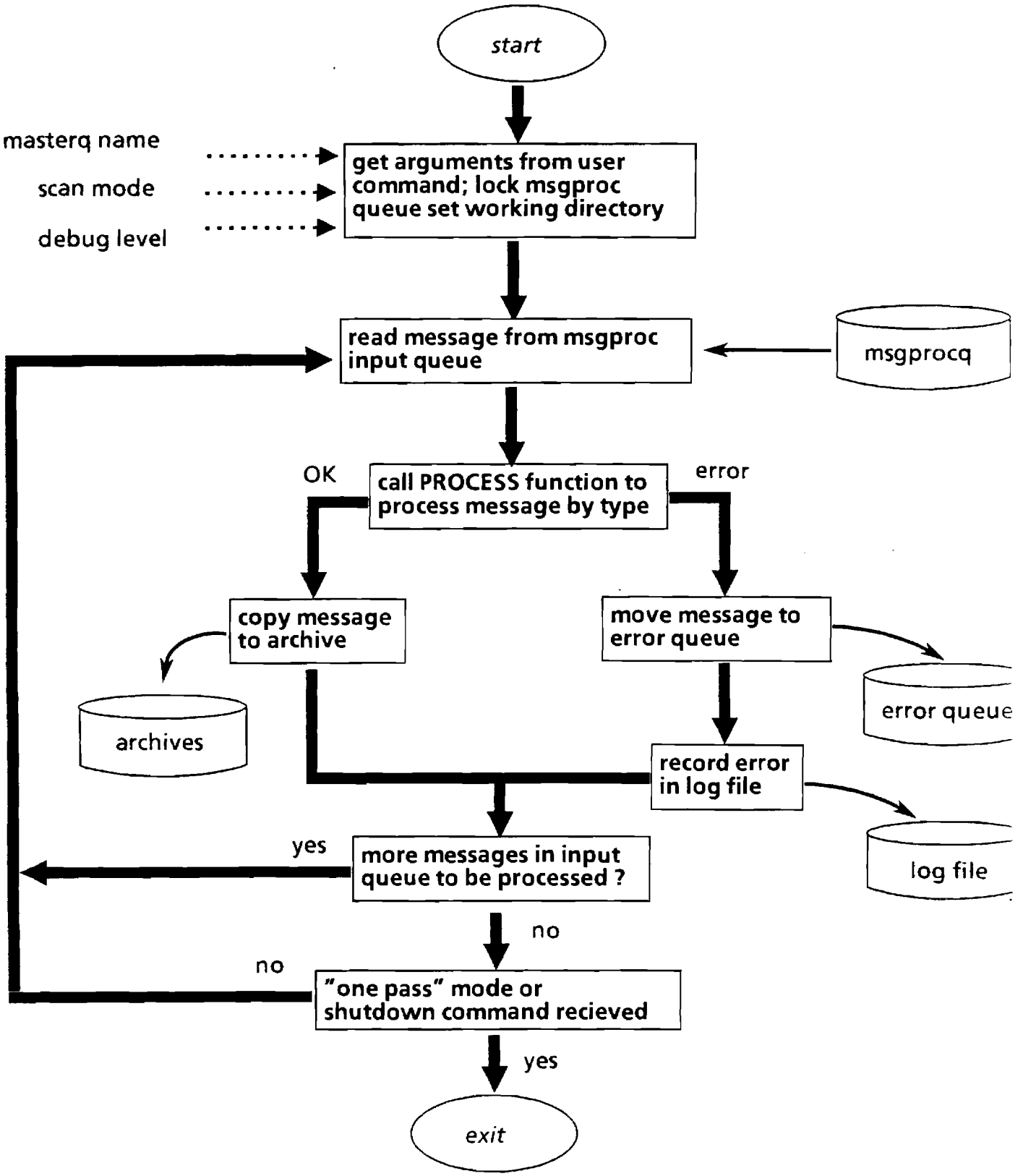
Schedule


start
site name
priority level
get site name and priority from qms
port table
check for existing conversation with desired site
conversation found
exit to qms
port table
get a port to use for caller
no available ports
got a port
priority messages
routine message
got a port
try to pre-empt a routine call and sieze its port
no available ports
port table
invoke a caller for desired site with given priority level; update port table to show status of port
exit and wait for port
exit to qms

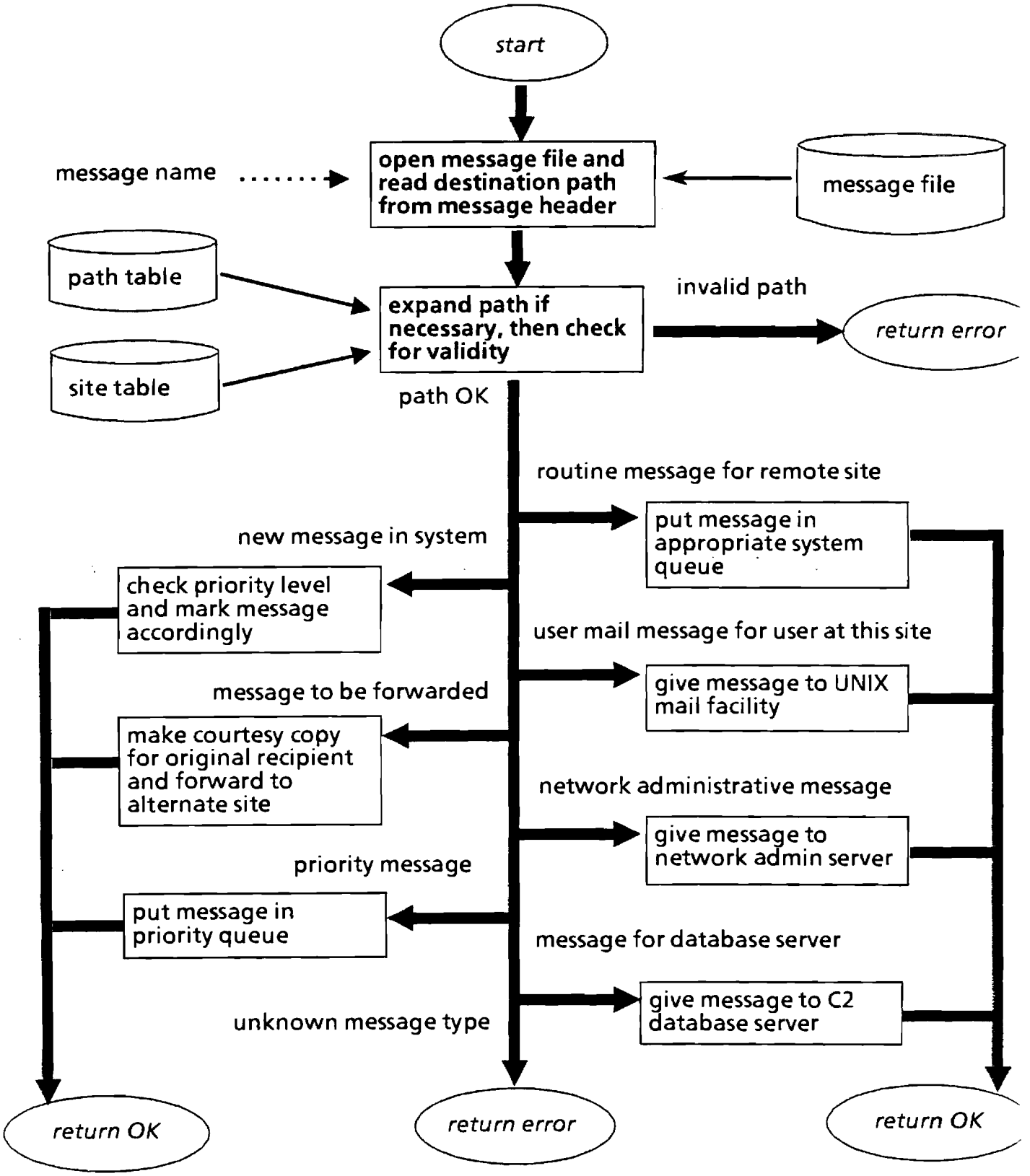
Caller


start
site name
port name
debug level
get arguments from user command or qms invocation; lock site queue and reset modem on given port for dialout operation
modem
lock file
site table
get phone number and login information from site table
success
use modem to call the remote site and establish connection
failure
log in to remote system
failure
site table
success
invoke IOControl with site name and port name
error
check call history in site table to see if site should be declared down
yes
success
no
record successful call in site table
site table
record failed call in site table
declare site down
remove lock file and exit

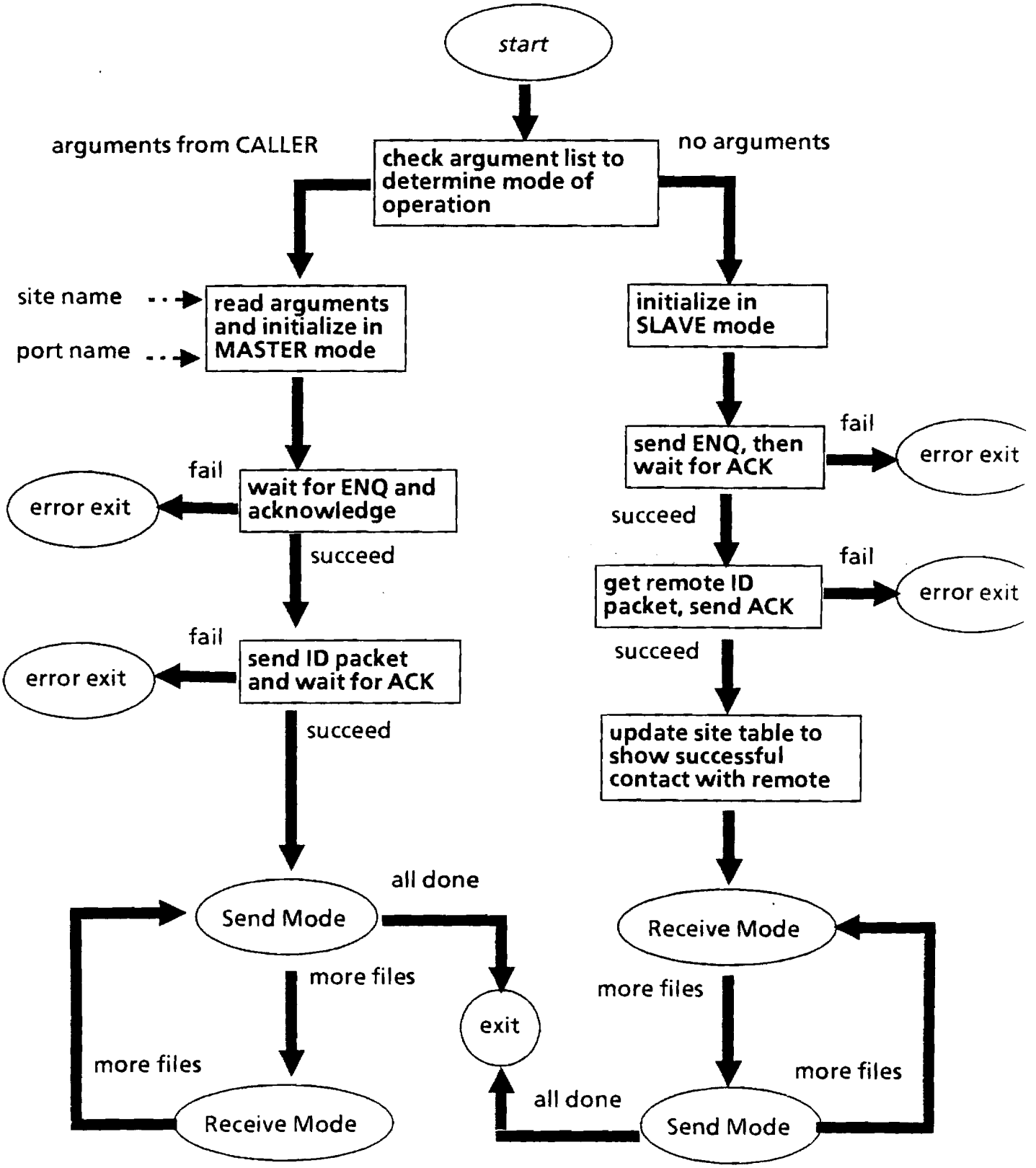
Msgproc


start
masterq name
scan mode
debug level
get arguments from user command; lock msgproc queue set working directory
read message from msgproc input queue
msgprocq
OK
call PROCESS function to process message by type
error
copy message to archive
archives
move message to error queue
error queue
record error in log file
log file
yes
more messages in input queue to be processed ?
no
no
"one pass" mode or shutdown command recieved
yes
exit

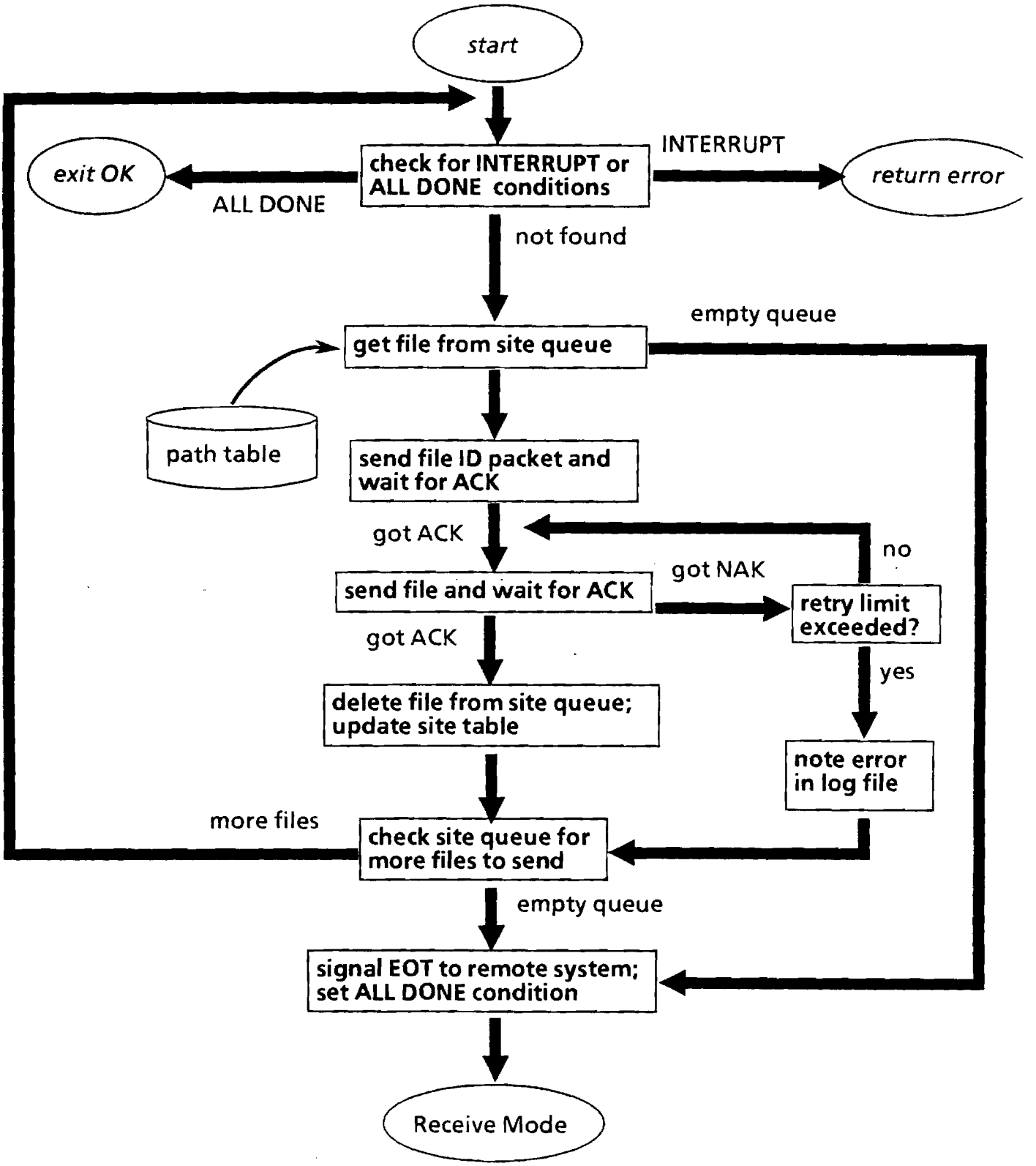
Process


start
message name
open message file and read destination path from message header
message file
path table
site table
expand path if necessary, then check for validity
invalid path
return error
path OK
routine message for remote site
put message in appropriate system queue
new message in system
check priority level and mark message accordingly
user mail message for user at this site
give message to UNIX mail facility
message to be forwarded
make courtesy copy for original recipient and forward to alternate site
network administrative message
give message to network admin server
priority message
put message in priority queue
message for database server
give message to C2 database server
unknown message type
return OK
return error
return OK

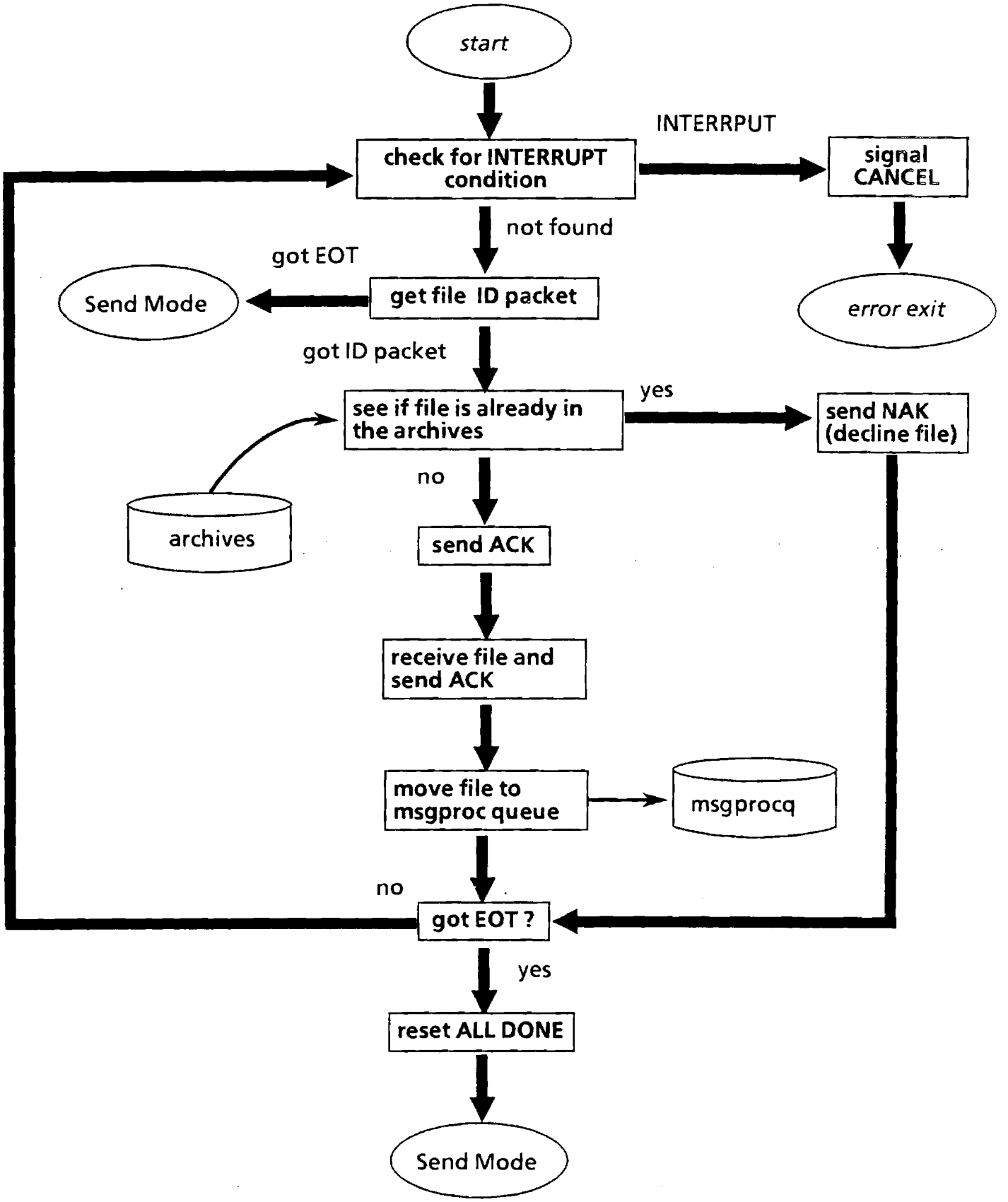
IOControl


start
arguments from CALLER
check argument list to determine mode of operation
no arguments
site name
port name
read arguments and initialize in MASTER mode
initialize in SLAVE mode
fail
wait for ENQ and acknowledge
error exit
succeed
send ENQ, then wait for ACK
fail
error exit
succeed
fail
send ID packet and wait for ACK
error exit
succeed
get remote ID packet, send ACK
fail
error exit
succeed
update site table to show successful contact with remote
all done
Send Mode
more files
more files
Receive Mode
exit
Receive Mode
more files
more files
all done
Send Mode

Send Mode


start
check for INTERRUPT or ALL DONE conditions
INTERRUPT
return error
exit OK
ALL DONE
not found
empty queue
get file from site queue
path table
send file ID packet and wait for ACK
got ACK
got NAK
no
send file and wait for ACK
retry limit exceeded?
got ACK
yes
delete file from site queue; update site table
note error in log file
more files
check site queue for more files to send
empty queue
signal EOT to remote system; set ALL DONE condition
Receive Mode

Receive Mode



C Language Program Listings for the TACCNET System

by

Principal Investigator:
Alton P. Jensen, Professor

Project Manager:
William O. Putnam, Research Scientist II

Project Staff:
Steven L. Goldberg, Research Assistant
Hong S. Shinn, Graduate Research Assistant

School Of Information and Computer Science
Georgia Institute Of Technology
Atlanta, Georgia 30332

Presented to:

U.S. Army Institute for Research In Management
Information, Communications, and Computer Science
(AIRMICS)

December 30, 1986

Contract No. DAHC0-85-C00012
Research Project No. G36-633

Introduction

This volume contains the C program source code listings for the TACCNET system. There are eleven sections to this volume: one for each of the TACCNET programs, one for the common functions, and one for the UNIX† shell command files used to control the system. Each section is independently numbered and indexed. A table of contents appears at the beginning of each section. The TACCNET system is divided into nine programs, each of which is composed of a number of functions ‡. Some functions are used in more than one program and are therefore grouped in a library of common functions. Each program has a primary function called *main()* in the file with the same name as the program.

All C language source file names end in the suffix ".c". Files whose names end in ".h" are header files containing global constants and data structure definitions. Files whose names end in ".e" contain declarations of external variables to be used by subprograms. Each TACCNET program has a corresponding command file called "Makefile" which contains lists of module dependencies and instructions for compiling and linking the program. The "Makefile" is used by the UNIX program maintainer **make** to build a system of executable programs.

The reader of this document is expected to be familiar with the contents of the TACCNET technical specification titled *Considerations in the Design and Development of a Combat Service Support Computer System* wherein the usage and operation of these programs is explained.

It will be difficult for persons not familiar with the C programming language to use this document. It will also be helpful to be familiar with the UNIX operating system under which TACCNET was developed. The definitive reference for the C language is *The C Programming Language* by Brian Kernighan and Dennis Ritchie (Prentice-Hall, 1978). There are numerous introductory books on the C language and the UNIX operating system available.

---

† UNIX is a trademark of AT&T Bell Laboratories.

‡ In the C programming language subprograms are called functions. A program must consist of at least one function but may call many other functions. Functions may call other functions and may even call themselves recursively.

Shell Command Files

This section contains the listings of the shell command files used to start, stop, and control the TACCNET system. These command files contain small programs written in the Bourne Shell command language. The Bourne Shell is standard on all UNIX systems and licensed derivatives. It is described in the UNIX operating system command reference under the title **sh**.

```
#
# Abort - stop TACCNET communications, queueing, and db server systems
#
stopsys
stopserve
exit 0
```

```
MASTERQ=${MASTERQ:-"/usr/taccnet"}


export MASTERQ
echo cleaning up $MASTERQ
cd $MASTERQ

: clear out the work directories listed in $MASTERQ/tables/queuelist
for i in ¢cat tables/queuelist¢
do
   rm -f $i/* $i/.??* >/dev/null 2>&1
done

: initialize the port and site tables
cp tables/sites.save tables/sites
cp tables/ports.save tables/ports
```

```
MASTERQ=${MASTERQ:-"/usr/taccnet"}
cd $MASTERQ/tables
cat ports
```

```
MASTERQ=${MASTERQ:-"/usr/taccnet"}
cd $MASTERQ/tables
cat sites
```

```
MASTERQ=${MASTERQ:-"/usr/taccnet"}
export MASTERQ
cd $MASTERQ/bin
if [ $# = "0" -o $# = "1" ]
then
   echo "usage: $0 priority dest [dest ...]"
   exit 1
fi
ams | (cd $MASTERQ; bin/genmsg $* 2>/dev/null)
```

```
MASTERQ=${MASTERQ:-"/usr/taccnet"}
:
export MASTERQ
cd $MASTERQ/bin

if [ $# -lt 2 ]
then
   echo "usage: $0 priority dest [dest ...]"
   exit 1
fi

file=/tmp/$$
ams >$file

while [ $file ]
do
        vi $file
        echo -n "Send, Edit, or Abort? "
        while read i
        do
                case $i in
                        A|a) rm $file; exit;;
                        S|s) break 2;;
                        E|e) break;;
                        *) echo -n "enter s, e, or a: "
                esac
        done
done

cd $MASTERQ;
bin/genmsg $* <$file 2>/dev/null
```

```
MASTERQ=${MASTERQ:-"/usr/taccnet"}
export MASTERQ
cd $MASTERQ/bin
if [ $# != "4" ]
then
   echo "usage: jtsend msgname unitid priority destination"
   exit 1
fi
build $* 2>/dev/null
```

```
MASTERQ=${MASTERQ:-"/usr/taccnet"}
cd $MASTERQ/log
tail -f $1*.log
```

```
MASTERQ=${MASTERQ:-"/usr/taccnet"}
export MASTERQ
cd $MASTERQ
bin/genmsg $* <&0 >&1
```

```
MASTERQ=${MASTERQ:-"/usr/taccnet"}
cd $MASTERQ
for i in ¢cat tables/queuelist¢
do
   echo $i:
   lc -aF $i
done
```

```
MASTERQ=${MASTERQ:-"/usr/taccnet"}
ABORTFILE=".abort"
export MASTERQ
#
# Stop the TACCNET system by placing the abort file ('.abort') in the
# top-level queue of the system to be shut down.  The file will be
# removed the next time the system is activated.
#
cd $MASTERQ
echo Shutdown: signalling TACCNET system shutdown
touch $ABORTFILE
```

```
if [ $1 ]
then

   MASTERQ=${MASTERQ:-"/usr/taccnet"}
   cd $MASTERQ/bin
   PATH=$PATH:$MASTERQ/bin
   export MASTERQ PATH
   echo starting server in $MASTERQ

   : start the server and record the pid for stopsys

   nohup server $MASTERQ/bin $MASTERQ/$1 $2 > nohup.out &
   echo >$MASTERQ/bin/.spids $!

else
   echo 'usage: startserve sitename'
   exit 1
fi
```

```
MASTERQ=${MASTERQ:-"/usr/taccnet"}
export MASTERQ
echo starting message system in $MASTERQ
cd $MASTERQ/bin
PATH=$PATH:$MASTERQ/bin

: start the message  processor and record the pid for stopsys
nohup msgproc $MASTERQ $* > nohup.out &
echo >>$MASTERQ/bin/.pids $!

: start the queue manager/scheduler and record the pid for stopsys
nohup qms $MASTERQ $* > nohup.out &
echo >>$MASTERQ/bin/.pids $!

touch /usr/spool/uucp/LCK..transmit
```

```
MASTERQ=${MASTERQ:-"/usr/taccnet"}
export MASTERQ
cd $MASTERQ/bin

if [ ! -r .spids ]
then
   echo "$MASTERQ/bin/.spids file not found"
   exit
fi

if [ ! -s .spids ]
then
   echo "$MASTERQ/bin/.spids file is empty"
   exit
fi

: kill the processes listed in the .spids file and remove the file
echo killing server in $MASTERQ
kill -15 ¢cat .spids¢
rm -f .spids >/dev/null 2>&1
```

```
MASTERQ=${MASTERQ:-"/usr/taccnet"}
export MASTERQ
cd $MASTERQ/bin

if [ ! -r .pids ]
then
   echo "$MASTERQ/bin/.pids file not found"
   exit
fi

if [ ! -s .pids ]
then
   echo "$MASTERQ/bin/.pids file is empty"
   exit
fi

: kill the processes listed in the .pids file and remove the file
echo killing message system in $MASTERQ
if kill -15 ¢cat .pids¢
then
   rm -f .pids >/dev/null 2>&1
   cd $MASTERQ
:  rm -f *.LCK >/dev/null 2>&1
:  rm -f tables/*.LCK >/dev/null 2>&1
   rm -f /usr/spool/uucp/LCK..transmit >/dev/null 2>&1
fi
```

```
:
#
#  Start TACCNET communications, queueing, and db server systems
#
MASTERQ=${MASTERQ:-"/usr/taccnet"}
export MASTERQ
cd $MASTERQ/bin
startsys - $*
startserve serverq -
sleep 1
exit 0
```

```
MASTERQ=${MASTERQ:-"/usr/taccnet"}
export MASTERQ
cd $MASTERQ
DATA1=unify.db
DATA2=file.db
PATH1=$MASTERQ/bin/$DATA1
PATH2=$MASTERQ/bin/$DATA2
MYNAME=¢cat bin/unitid¢

if [ $# -ne 1 -a $# -ne 2 ]
then
   echo usage: $0 sitename
   exit 1
fi

RMASTERQ=$MASTERQ
if [ $# -eq 2 ]
then
   RMASTERQ=$2
fi
BACKDIR=$RMASTERQ/backups

if [ ! -r $PATH1 ]
then
   echo cannot read $PATH1
   exit 1
fi

if [ ! -r $PATH2 ]
then
   echo cannot read $PATH2
   exit 1
fi

if fgrep :$1 $MASTERQ/tables/sites >/dev/null
then

   { echo save $BACKDIR/$MYNAME/$DATA1 ; cat $PATH1 ;}¦bin/genmsg 2 $1!net.adm;
   retcode=$?
   if [ $retcode -ne 0 ]
   then
       echo backing up $PATH1: return code $retcode
   fi

   { echo save $BACKDIR/$MYNAME/$DATA2 ; cat $PATH2 ;}¦bin/genmsg 2 $1!net.adm;
   retcode=$?
   if [ $retcode -ne 0 ]
   then
       echo backing up $PATH2: return code $retcode
   fi

else
   echo site $1 not found in $MASTERQ/tables/sites
   exit 2
fi
```

```
MASTERQ=${MASTERQ:-"/usr/taccnet"}
export MASTERQ
cd $MASTERQ
MYNAME=¢cat bin/unitid¢
DATA1=unify.db
DATA2=file.db
BACKDIR=$MASTERQ/backups

if [ $# -ne 1  -a $# -ne 2 ]
then
   echo usage: $0 sitename
   exit 1
fi

RMASTERQ=$MASTERQ
if [ $# -eq 2 ]
then
   RMASTERQ=$2
fi
PATH1=$RMASTERQ/backups/$MYNAME/$DATA1
PATH2=$RMASTERQ/backups/$MYNAME/$DATA2

if [ ! -d $BACKDIR -o ! -w $BACKDIR ]
then
   echo cannot access $BACKDIR
   exit 1
fi

if [ -f $BACKDIR/$DATA1 ]
then
   echo file $BACKDIR/$DATA1 already exists
   exit 1
fi

if [ -f $BACKDIR/$DATA2 ]
then
   echo file $BACKDIR/$DATA2 already exists
   exit 1
fi


if fgrep :$1 $MASTERQ/tables/sites >/dev/null
then

   echo transfer $PATH1 $BACKDIR/$DATA1 ¦ bin/genmsg 2 $1!net.adm;
   retcode=$?
   if [ $retcode -ne 0 ]
   then
       echo restoring $DATA1: return code $retcode
   fi

   echo transfer $PATH2 $BACKDIR/$DATA2 ¦ bin/genmsg 2 $1!net.adm;
   retcode=$?
   if [ $retcode -ne 0 ]
   then
       echo restoring $DATA2: return code $retcode
   fi

else
   echo site $1 not found in $MASTERQ/tables/sites
```

```
    exit 2
fi
```

```
MASTERQ=${MASTERQ:-"/usr/taccnet"}
cd $MASTERQ
rm -f *.LCK
rm -f tables/*.LCK
rm -f bin/*.LCK
```

Common Functions

This section contains common functions used by many different programs and functions in the TACCNET system. If the source code for a function is not given in the separate program listings it will be in this section.

```
common:
        make stripme.o myname.o remove.o fileopen.o filenq.o newfile.o\
             datetime.o validsite.o readsite.o lockfile.o frename.o\
             dequeue.o getdir.o writelog.o givetomp.o ports.o state.o\
             receive.o receiveh.o archive.o putsite.o abort.o

links:
        ln net.h iocontrol.h iocontrol.e wait.h sysdef.h retcodes.h\
           datetime.o dequeue.o getdir.o fileopen.o frename.o givetomp.o\
           lockfile.o myname.o ports.o readsite.o receive.o state.o\
           putsite.o validsite.o writelog.o ../caller
        ln net.h sysdef.h datetime.o filenq.o fileopen.o lockfile.o\
           myname.o newfile.o readsite.o stripme.o validsite.o\
           writelog.o ../genmsg
        ln net.h sysdef.h abort.o datetime.o dequeue.o getdir.o filenq.o\
           fileopen.o frename.o lockfile.o myname.o newfile.o readsite.o\
           remove.o stripme.o validsite.o writelog.o archive.o ../msgproc
        ln net.h wait.h sysdef.h abort.o datetime.o dequeue.o getdir.o\
           fileopen.o frename.o givetomp.o lockfile.o myname.o ports.o\
           readsite.o state.o stripme.o validsite.o writelog.o putsite.o ../qms
        ln net.h iocontrol.h iocontrol.e wait.h sysdef.h retcodes.h abort.o\
           datetime.o dequeue.o getdir.o filenq.o fileopen.o lockfile.o\
           myname.o readsite.o receive.o receiveh.o remove.o validsite.o\
           putsite.o writelog.o archive.o ../iocontrol
        ln net.h sysdef.h abort.o datetime.o dequeue.o getdir.o fileopen.o\
           frename.o lockfile.o remove.o writelog.o ../server
        ln wait.h lockfile.o readsite.o ports.o myname.o fileopen.o ../console


stripme.o: net.h stripme.c
        cc -c -O stripme.c

myname.o: net.h  myname.c
        cc -O -c myname.c

remove.o: net.h remove.c
        cc -O -c remove.c

fileopen.o: net.h fileopen.c
        cc -O -c fileopen.c

filenq.o: net.h filenq.c
        cc -O -c filenq.c

newfile.o: net.h newfile.c
        cc -O -c newfile.c

datetime.o: net.h datetime.c
        cc -c -O datetime.c

lockfile.o: net.h lockfile.c
        cc -c -O lockfile.c

frename.o: net.h frename.c
        cc -c -O frename.c

dequeue.o: net.h dequeue.c
        cc -c -O dequeue.c

getdir.o: net.h getdir.c
```

```
        cc -c -O getdir.c

writelog.o: net.h writelog.c
        cc -c -O writelog.c

readsite.o: net.h readsite.c
        cc -c -O readsite.c

validsite.o: net.h validsite.c
        cc -c -O validsite.c

givetomp.o: net.h givetomp.c
        cc -c -O givetomp.c

ports.o: net.h ports.c
        cc -c -O ports.c

state.o: net.h state.c
        cc -c -O state.c

receive.o : receive.c iocontrol.h
        cc -O -c receive.c

receiveh.o : receive.c iocontrol.h
        mv receive.o .receive.o
        cc -O -c -DFORHONEY receive.c
        cp receive.o receiveh.o
        rm receive.o
        mv .receive.o receive.o

archive.o: archive.c net.h
        cc -O -c archive.c

putsite.o: net.h putsite.c
        cc -c -O putsite.c

abort.o: net.h abort.c
        cc -c -O abort.c
```

```
/* Global variable descriptions */

extern int ModemFd; /* Current file descriptor for modem port */
extern int SeqNo;  /* Current packet sequence number for transmit or receive */
extern int DebugLevel;  /* Runtime debug level (0 = normal) */
extern int MasterMode;  /* Flag indicating that IOCONTROL invoked as master */
/* */
```

```
/* mode definitions for iocontrol.c */

#define HANGUP     -1
#define MASTERINIT  0
#define SLAVEINIT   1
#define SENDMODE    2
#define RECEIVEMODE 3

/* error codes and corresponding messages for exit(2) system call */

#include "retcodes.h"

/* miscellaneous constants */

#define MAXPACKET 2000    /* Maximum length of a packet */
#define MAXRETRY 10       /* Maximum retransmits or re-receives */
#define MAXDELAY 10       /* Maximum wait time for incoming character */

#ifdef FORHONEY
#define BLOCKLENGTH 64    /* Number of characters in a block */
#else
#define BLOCKLENGTH 140
#endif

#define ETXOFFSET 7       /* Position of etx/etb char in received packet */
#define PKTOVERHEAD 14+6  /* Overhead for control information in a packet */
#define REJECT 2          /* Reject return code for old messages */

/* ASCII control codes used throughout */

#ifdef PROTOCOL_DBG

#define STX '<'
#define ETX 'X'
#define EOT 'O'
#define ENQ 'E'
#define ACK 'A'
#define DLE '.'
#define NAK 'N'
#define ETB 'B'
#define CAN 'C'
#define EM '>'

#else

#define STX (char) 0x02   /* Start of text */
#define ETX (char) 0x03   /* End of text */
#define EOT (char) 0x04   /* End of transmission */
#define ENQ (char) 0x05   /* Enquire */
#define ACK (char) 0x06   /* Positive acknowledge */
#define DLE (char) 0x10   /* Data-link escape */
#define NAK (char) 0x15   /* Negative acknowledge */
#define ETB (char) 0x17   /* End of text block */
#define CAN (char) 0x18   /* Cancel session (abort) */
#define EM  (char) 0x19   /* End of message */

#endif

#define MYEOT (char) 0x74 /* Return code saying we were interrupted */
#define MYCAN (char) 0x78 /* Return code saying we were aborted */
```

```
/* iocontrol function headings for calls */

int   CheckSum ();                          /* CheckSum (address, length) */
FILE *CreateFile ();                        /* CreateFile (name, queue) */
void  Exit ();                              /* Exit (errcode) */
unsigned char *GetBlock ();                 /* GetBlock (length, end) */
int   GetFile ();
int   GetHeader ();                         /* GetHeader (filename) */
unsigned char *GetPacket ();                /* GetPacket (length) */
int   Preemption ();                        /* Preemption (queuename) */
int   Receive ();
int   Send ();                              /* Send (address, length) */
int   SendBlock ();                         /* SendBlock (data, length, end) */
int   SendByte ();                          /* SendByte (byte) */
int   SendEnq ();
int   SendFile ();                          /* SendFile (pathname, messagename) */
int   SendHeader ();                        /* SendHeader (messagename) */
int   SendName ();
int   SendPacket ();                        /* SendPacket (data, length, end) */
int   WaitAck ();
int   WaitName ();

/* common (global) variables are in the file iocontrol.e ('e'xternal) */
```

```
#include "sysdef.h"
#include <stdio.h>
#include <sys/types.h>
#include <sys/ioctl.h>
#include <termio.h>
#include <fcntl.h>
#include <ctype.h>

#ifdef XENIX_V                  /* System V XENIX define NULL differently */
#undef NULL
#define NULL 0
#endif

#define UMASK 002

/* return codes */

#define GOOD         0
#define BUSY         2
#define ERR         -1

#define GCOSLOGIN    "LOGIN "
#define UNIXLOGIN    "ogin: "
#define UNIXPASSWORD "assword:"
#define GCOS         'G'
#define UNIX         'U'
#define EMULATED     'E'

#define NETADMIN  "net.adm" /* name of network administrator process   */

#define TRUE  1
#define FALSE !TRUE

#define UP    TRUE
#define RETRY UP+1
#define DOWN  UP-1

/* Message Type Descriptors                                            */

#define SERVERTYPE   'J'        /* JINTACCS message for server         */
#define MESSAGETYPE  'M'        /* normal message type                 */
#define USERTYPE     'U'        /* user mail message type              */
#define ERRORTYPE    'E'        /* message is mangled somehow          */
#define NAKTYPE      'N'        /* message got a NAK on transmission   */
#define ADMINTYPE    'A'        /* network adminsitrative message      */
#define REROUTETYPE  'F'        /* message is to be rerouted           */
#define CCTYPE       'C'        /* courtesy copy message               */
#define NOHEADTYPE   'H'        /* message with bad header             */
#define NOPATHTYPE   'S'        /* message with invalid path specification */
#define PRIORTYPE    'P'        /* high priority message               */
#define ROUTINETYPE  'R'        /* routine message                     */
#define AVAILABLE    'A'        /* port available flag                 */

/* Network Administrative Commands                                     */

#define ADD       'a'           /* add a site to the site table        */
#define CHANGE    'c'           /* change existing site                */
#define DELETE    'd'           /* delete a site from the table        */
#define EXAMINE   'e'           /* get the status of a site            */
#define RECOVER   'r'           /* recover old messages from archives  */
```

```
#define SAVE      's'          /* save message contents into named file   */
#define TRANSFER  't'          /* request a copy of a file be transferred */

/* Convenient function definitions */

#define EQUALS !strcmp
#define SkipEOL(fd) {int c; while(((c=getc(fd))!='\n') && (c!=EOF));}




#define FlushModemInput(fd) ioctl(fd,TCFLSH,0)
#define NOW time((long *)0)

/* System constants */

#define DOT        '.'
#define FIELDMARK  ':'
#define SEPCHAR    '!'
#define DELCHAR    '*'
#define HEADERLINE '>'
#define CRET       '\r'
#define NL         '\n'
#define TAB        '\t'
#define BLANK      ' '
#define CR         "\015"
#define ONE        "1"
#define ZERO       "0"
#define CCPOS       3        /* position of the CCFLAG in Priority line */

/* Hayes Smartmodem 1200 commands and responses */

#define ESCAPESTR "+++"
#define HANGUPCMD "ATH\r"
#define RESETCMD  "ATZ\r"
#define NOANSWER  "3"
#define DIALSTR   "ATD"
#define SETUP     "\rAT E0 Q0 V0\r"
#define ATTENTION "AT\r"
#define OK '0'

/* files and directories used by the system */

#define MASTERQ       "."
#define BIN           "bin"
#define PRIORQ        "priority"
#define MSGPROCQ      "msgprocq"
#define SERVERQ       "serverq"
#define ERRORQ        "errorq"
#define ARCHIVEQ      "archive"
#define PATHTABLE     "tables/paths"
#define SITETABLE     "tables/sites"
#define NEWSITETABLE  "tables/sites.new"
#define OLDSITETABLE  "tables/sites.old"
#define PORTTABLE     "tables/ports"
#define ALTSITES      "tables/altsites"
```

```
#define MYNAME          "tables/myname"
#define PARAMFILE       "tables/params"
#define PASSWDFILE      "/etc/passwd"
#define INTFILE         ".Interrupt"

/* programs loaded by the system */

#define IOCONTROL       "iocontrol"
#define IOCNTRLH        "iocontrolh"
#define CALLER          "caller"
#define QMS             "qms"
#define MSGPROC         "msgproc"
#define GENMSG          "genmsg"
#define SERVER          "server"

/* system constants */

#define MAXDOWNSITES    20
#define MAXALTSITES     20
#define MAXPHONENUMS    4
#define PROGNAMELEN     14
#define SITENAMELEN     14
#define FILENAMELEN     40
#define PATHNAMELEN     80
#define LINELEN         128

typedef char filename [FILENAMELEN];
typedef char pathname [PATHNAMELEN];
typedef char sitename [SITENAMELEN];

typedef struct          /* site table entry */
   {
    sitename SiteName;                  /* name of the site                */
    short    Status;                    /* up, down, priority, busy        */
    short    NumCalls;                  /* number of times we called so far */
    long     TimeToCall;                /* don't call before this time     */
    char     SysType;                   /* operating system type           */
    filename Password;                  /* password for taccnet login      */
    pathname PhoneNum [MAXPHONENUMS+1]; /* array of phone numbers to try   */
   } site ;




typedef struct
   {
    int  Priority;       /* Message priority class (1-6)       */
    int  CCFlag;         /* Courtesy copy flag (TRUE, FALSE)   */
    char *DestSite;      /* Destination site name              */
    char *DownSites[MAXDOWNSITES+1]; /* List of previously-tried sites */
   } header ;
```

```
typedef struct          /* port table entry */
   {
    char *Port;    /* port name (unix path name)                          */
    char *Site;    /* remote site name, if connected                      */
    int  State;    /* port state (Available, Routine, Priority)           */
   } portentry ;



typedef portentry *portlist;

/* Unix calls.  Unix is a registered trademark of ATT Bell Laboratories   */

char *calloc ();
char *ctime ();
char *getenv ();
char *malloc ();
char *strcpy ();
char *strrchr ();
long time ();
unsigned sleep ();
FILE *popen ();


/* System functions                                                       */

char *StripMe ();       /* Gets first site from full path (a!b!c!d)       */
char *GetPrompt ();     /* Reads prompt message from modem                */
site *ValidSite ();     /* Validates site and returns pointer to site entry*/
site *ReadSite ();      /* read site entry from stream into struct        */
FILE *NewFile ();       /* creates new file in named queue, returns name  */
char **GetDir ();       /* read directory file into an array of filenames */
char *DeQueue ();       /* get first message name from the filename array */
FILE *FileOpen ();      /* open a file in a queue and return Fd           */
char *MyName ();        /* returns the name of this site                  */
char *GetLine ();       /* read a line of input and trim off the newline  */
portlist *GetPorts();   /* read the PORTTABLE into a structure for use    */
char **GetSites ();     /* read the SITETABLE and get a list of site names */
```

```
#ifndef GOOD
#define GOOD            0
#endif

#define INTERRUPTED     1
#define ABORTED         2
#define LOSTCONTACT    10
#define BADCONNECTION  20
#define BADREMOTENAME  30
#define FATAL          40
#define INTERNALERROR  50
#define RECOVERABLE    60

#define MLOSTCONTACT       "lost contact with remote"
#define MBADCONNECTION     "could not connect with remote"
#define MBADREMOTENAME     "remote name not defined"
#define MFATAL             "fatal error"
#define MINTERNALERROR     "internal error"
#define MRECOVERABLE       "problem on local system"
```

```
/* This file define the type of UNIX system to be used. */
/* Un-comment one and ONLY one of the following lines:  */

#define XENIX     /* for microsoft XENIX systems */
/*#define XENIX_V  /* for SCO Xenix V; define both this and XENIX */
/*#define DISTRIX  /* for Convergent Technologies DISTRIX systems */
/*#define ONYX     /* for ONYX systems             */
```

```
/*
 * Structure of the information in the first word returned by
 * wait.  If w_stopval==WSTOPPED, then the second structure
 * describes the information returned, else the first.
 */
union wait      {
        int     w_status;               /* used in syscall */
        /*
         * Terminated process status.
         */
        struct {
                unsigned short  w_Termsig:7;    /* termination signal */
                unsigned short  w_Coredump:1;   /* core dump indicator */
                unsigned short  w_Retcode:8;    /* exit code if w_termsig==0 */
        } w_T;
        /*
         * Stopped process status.
         */
        struct {
                unsigned short  w_Stopval:8;    /* == W_STOPPED if stopped */
                unsigned short  w_Stopsig:8;    /* signal that stopped us */
        } w_S;
};

#define WSTOPPED        0177    /* value of s.stopval if process is stopped */
```

```
#include "net.h"

#define ABORTFILE ".abort"

/*
   Abort - Return whether or not the file ".../.abort" exists,
           indicating that the current system should be shut down.
           The .abort file will be deleted only once the system is
           restarted.
*/

int Abort (masterq)

char *masterq;  /* Queue in which ABORTFILE might exist */

{
   pathname AbortFile;  /* File name for abort file */
   register int i,Fd;
   int Result;

   sprintf (AbortFile, "%s/%s", masterq, ABORTFILE);

   Fd = open (AbortFile, O_RDONLY);

   Result = (Fd != ERR);  /* Return TRUE if file exists */

   if (Result)
      close (Fd);         /* Free the file descriptor */

   return (Result);
}
```

```
#include "net.h"

extern int Archiving;  /* Flag permitting archiving of all messages */

/*      Archive - copy file in given queue into archive directory  */

int Archive (FileName, Queue)

char *FileName;
char *Queue;

{
   int c;
   int Oflags = O_WRONLY | O_CREAT | O_EXCL;
   int Mode = 0664;
   int Fd;
   FILE *OldFp;
   FILE *NewFp;
   pathname Path;

   if (!Archiving)
      return (TRUE);

   sprintf (Path, "%s/%s", Queue, FileName);     /* name of input file  */

   if ((OldFp = fopen (Path, "r")) == NULL)      /* read from this file */
      {
       WriteLog ("Archive: can't read", Path, "-", "file not archived");
       return (FALSE);
      }

   sprintf (Path, "%s/%s", ARCHIVEQ, FileName); /* name of archive file  */

   if ((Fd = open (Path, Oflags, Mode)) == ERR) /* see if it is already there */
      {
       WriteLog ("Archive:", Path, "already", "in archive");
       fclose (OldFp);
       return (ERR);    /* indicate to calling function that file exists */
      }

   if ((NewFp = fdopen (Fd, "w")) == NULL)  /* open as a stream for writing */
      {
       WriteLog ("Archive: can't write", Path, "-", "file not archived");
       fclose (OldFp);
       return (FALSE);
      }

   while ((c = getc(OldFp)) != EOF)             /* copy the file */
      putc (c, NewFp);

   fclose (OldFp);                              /* close the files */
   fclose (NewFp);
   close (Fd);

   return (TRUE);

}
```

```
#include "net.h"
#include <time.h>

/*
   DateTime - Construct a string containing the date-time header
              for each line to be written to the log file.
*/

int DateTime (Str)

char *Str;

{
   long BDate;
   struct tm *Time;     /* Structure for detailed date-time information */

   struct tm *localtime();

   BDate = time ((long *) 0);

   Time = localtime (&BDate);  /* Get date-time information */

   sprintf (Str, "%.2d/%.2d/%.2d %.2d:%.2d:%.2d", Time->tm_mon+1,
            Time->tm_mday, Time->tm_year, Time->tm_hour, Time->tm_min,
            Time->tm_sec);

   return (0);
}
```

```
#include "net.h"

/*
   DeQueue - Return the name of the first message in the input queue
             opened via GetDir. Return NULL if the queue is empty.
*/

char *DeQueue (MsgQueueName)

char *MsgQueueName;

{
   static filename *MsgFiles;    /* Pointer to list of files in the queue */
   static int CurFilePtr = 0;    /* Index of current filename */
   static pathname QueueName;    /* Name of the current queue */

   /* if list exhausted or new queue, read again for more file names */
   if ((!EQUALS(MsgQueueName, QueueName)) || (CurFilePtr == 0) ||
       (*MsgFiles [CurFilePtr] == NULL))
      {
       MsgFiles = GetDir (MsgQueueName);  /* reload directory listing */
       strcpy (QueueName, MsgQueueName);  /* reset queue name */
       CurFilePtr = 0;                    /* reset list index */
      }

   if (MsgFiles == NULL)
      return (NULL);  /* Return end of list condition */

   return (MsgFiles [CurFilePtr++]);  /* Point to next file for later */
}
```

```
#include "net.h"

/*
   FileNQ - Make a file named by FileName visible in the directory given
            in Queue by removing the '.' in front of its name. Return
            TRUE or FALSE result.
*/

int FileNQ (FileName, Queue)

char *FileName;
char *Queue;

{
   pathname OldName;    /* Storage for old filename */
   pathname NewName;    /* Storage for new filename */

   sprintf (OldName, "%s/.%s", Queue, FileName);
   sprintf (NewName, "%s/%s", Queue, FileName);

   unlink (NewName);  /* Make sure the file doesn't already exist */

   if (link (OldName, NewName))   /* Rename the file, removing the first char */
      {
       WriteLog ("FileNQ: can't link", OldName, "to", NewName);
       return (ERR);
      }

   if (unlink (OldName))     /* Get rid of invisible directory entry */
      {
       WriteLog ("FileNQ: can't unlink", OldName, "", "");
       return (ERR);
      }

   return;
}
```

```
#include "net.h"

/*
   FileOpen - Open the file FileName in the queue QueueName by constructing
              the full pathname of the file given these two components.
              Return a file descriptor to the file, or NULL if the file
              could not be opened.
*/

FILE *FileOpen (FileName, QueueName, FileType)

char *FileName;
char *QueueName;
char *FileType;

{
   pathname TempFileName;  /* Complete pathname of file to be opened */

   sprintf (TempFileName, "%s/%s", QueueName, FileName);

   return (fopen (TempFileName, FileType));
}
```

```
#include "net.h"

int FRename (Path1, Path2)

char *Path1;
char *Path2;

/* Renames Path1 to Path2 */

{
   if (link (Path1, Path2) == ERR)
       return (ERR);

   if (unlink (Path1) == ERR)
       return (ERR);

   return (GOOD);
}
```

```
#include "net.h"
#include <sys/dir.h>

#define NFILES 96   /* Max files per directory */
#define Size (sizeof (struct direct))

char **GetDir (Dir)
char *Dir;

{
   long lseek ();
   struct direct DirEntry;
   register int Num = 0;
   static filename Entries [NFILES];
   int DirFd; /* Directory file descriptor */

   if ((DirFd = open(Dir, O_RDONLY)) == ERR)
      {
       WriteLog ("GetDir:", "unable to open", Dir, "directory");
       return (NULL);  /* Return empty list if can't open directory */
      }

   if (lseek (DirFd, 32L, 0) == (long) ERR)    /* Skip . and .. */
      {
       close (DirFd);
       return (NULL);
      }

   while (read (DirFd, &DirEntry, Size) != 0)
       if ((DirEntry.d_ino != (ino_t) 0) && (DirEntry.d_name [0] != DOT))
          {
           strcpy (Entries [Num], DirEntry.d_name);
           Num++;
          }

   close (DirFd);

   if (Num > 0)
      {
       Sort (Entries, Num);
       *Entries [Num] = NULL;   /* Set the last one to NULL */
       return (Entries);
      }
   else
       return (NULL);
}




Sort (Entries, Num)
filename Entries [];
register int Num;
```

```
{
   register int Gap, i, j;
   filename Temp;

   for (Gap = Num / 2; Gap > 0; Gap /= 2)
       for (i = Gap; i < Num; i++)
           for (j = i - Gap; j >= 0; j -= Gap)
               {
                if (strcmp (Entries [j], Entries [j+Gap]) <= 0)
                    break;
                strcpy (Temp, Entries [j]);
                strcpy (Entries [j], Entries [j+Gap]);
                strcpy (Entries [j+Gap], Temp);
               }
}
```

```
#include "net.h"

extern int ForwardFlag;  /* Flad indicating forwarding is enabled */

int GiveToMP (SysName)

char *SysName;

{
/* Move all messages that are not courtesy copies over to the msgproc queue. */
/* We have to look out for case where nothing remains in the queue except CC */
/* messages. DeQueue() will keep giving us the names of the CC files forever */
/* if we don't detect this state and exit the loop. We do this by saving the */
/* name of the first CC encountered and checking it against later CC files.  */
/* When it is see for the second time, we will have sent all the messages to */
/* be rerouted except for the remaining CC files, which we must leave in the */
/* queue.                                                                    */

char *MessageFile;
pathname NewPath;
pathname OldPath;
pathname SaveName;
register int Count = 0;
register int Done = FALSE;
char     Number[10];

   if (!ForwardFlag)     /* Don't do forwarding if disabled */
      return (GOOD);

   strcpy (SaveName, "");  /* set strlen(SaveName) to 0 */
   while ( ((MessageFile = DeQueue(SysName)) != NULL) && (!Done) )
      if (MessageFile[0] != CCTYPE)  /* mark non-CC files for forwarding */
         {
          sprintf (OldPath, "%s/%s", SysName, MessageFile);
          sprintf (NewPath, "%s/%c%s", MSGPROCQ, REROUTETYPE, MessageFile+1);
          FRename ( OldPath, NewPath );
          WriteLog ("GiveToMP: moved", OldPath, "to", NewPath);
          ++Count;
         }
      else
         {
          if (strlen(SaveName) == 0) /* remember name of first CC message */
             strcpy (SaveName, MessageFile);
          else /* see if we are back to the first CC message */
             Done = EQUALS (SaveName, MessageFile);
         }

   if (Count > 0)
      {
       sprintf (Number, "%d", Count);
       WriteLog ("GiveToMP: forwarded", Number, "messages", "");
      }

   return (GOOD);
```

```
#include "net.h"

int Lock (FileName)

/*  returns ERR if the file is already locked, GOOD otherwise */

char *FileName;

/*
   Simple file locking mechanism using open(). We try to lock FileName by
   creating a file called FileName.LCK. Setting Oflags O_CREAT and O_EXCL
   causes open() to return ERR if the file already exists.
*/

{
   pathname LockFile;
   register int i,Fd;
   int Oflags = O_CREAT | O_EXCL;   /* return ERR if file exists */
   int Mode = 0664;

   sprintf (LockFile, "%s.LCK", FileName);

   /* try and lock the file several times before giving up */
   for (i = 0; i < 10; i++)
       if ((Fd = open (LockFile, Oflags, Mode)) == ERR)
           sleep (2);
       else
           break;

   if (Fd == ERR)
      return (ERR);       /* file was already locked */
   else
      {
       close (Fd);        /* don't forget to close the file */
       return (GOOD);     /* file is now locked */
      }

}




int UnLock (FileName)

/* returns ERR if the file could not be unlocked, GOOD otherwise */

char *FileName;

{
   pathname LockFile;
   int Result=0;

   sprintf (LockFile, "%s.LCK", FileName);
```

```
  Result = unlink (LockFile);

  if (Result == ERR)
     WriteLog ("UnLock: can't unlock", LockFile, "", "");

  return (Result);
```

```
#include "net.h"

/*
   MyName - Return the name of this site, found in the file MYNAME.
*/

char *MyName ()

{
   sitename TempSiteName;    /* Temporary storage for site name */
   FILE     *MyNameFd;       /* MYNAME file descriptor */
   char     *RetPtr;         /* Pointer to return to caller */

   if ((MyNameFd = FileOpen (MYNAME, MASTERQ, "r")) == NULL)
      {
       fprintf (stderr, "MyName: I don't know my own name.\n");
       fclose (MyNameFd);
       return ((char *) NULL);  /* Try to recover */
      }

   fscanf (MyNameFd, "%s", TempSiteName);  /* Get this site's name */

   fclose (MyNameFd);

   RetPtr = malloc (strlen (TempSiteName) + 1);   /* Get perm. storage */
   strcpy (RetPtr, TempSiteName);

   return (RetPtr);
}
```

```
#include "net.h"

/*
        NewFile - build a file name in the directory given by Queue, with
                  a '.' in front of the name to make it invisible. Return
                  the name of the file in FileName. Open the file and
                  return a file descriptor.  The file name will be
                  composed of the system name plus a date-time stamp.
*/

FILE *NewFile (FileName, Type, Queue)

char *FileName;      /* pointer to new file name */
int   Type;          /* character to indicate message type */
char *Queue;         /* directory in which to put the file */

{
   pathname PathName;      /* pointer to the full path name of the file */
   char *ThisName;         /* pointer to name of this site */
   long int AbsTime;

   ThisName = MyName ();  /* Get this site's name */

   AbsTime = NOW;

   sprintf (FileName, ".%c%s%x%.4x", Type, ThisName,
               (int) ((AbsTime & (long) 0x0f0000) >> 16),
               (int) (AbsTime & (long) 0x00ffff));
   sleep (1);  /* Ensure that clock increments */

   sprintf (PathName, "%s/%s", Queue, FileName);

   free (ThisName);

   return (fopen(PathName, "w"));
}
```

```
#include "net.h"
main()
{
   printf ("Now:   %.9ld\n", NOW);
}
```

```
#include "net.h"

void PSFree ();

int TakePort (Port, Site, Type)

char *Port;
char *Site;
int  Type;

/*
   Get the port list and rewrite the PORTTABLE indicating that the named
   Port is being used to call the named Site with a transmission of the
   given Type.  Assume that the PORTTABLE is already LOCKED.
*/

{
   portlist *PortList;
   register int i, Found=FALSE;

   if ((PortList = GetPorts ()) ==  NULL)
      {
       WriteLog ("TakePort: can't read port list to mark", Port, "in use", "");
       return (ERR);
      }

   for (i=0; (PortList[i] != NULL); i++)
      if (EQUALS(PortList[i]->Port, Port))
          {
           Found = TRUE;
           break;
          }

   if (!Found)
      {
       WriteLog ("TakePort: can't find", Port, "in", PORTTABLE);
       UnLock (PORTTABLE);
       PSFree (PortList);
       return (ERR);
      }

   free (PortList[i]->Site);
   PortList[i]->Site = malloc (strlen (Site)+1);
   strcpy (PortList[i]->Site, Site);
   PortList[i]->State = Type;

   if (PutPorts (PortList) ==  ERR)
      {
       WriteLog ("TakePort: can't write port list to mark", Port, "in use", "");
       PSFree (PortList);
       return (ERR);
      }

   PSFree (PortList);
   return (GOOD);
```

```
int FreePort (Port)

char *Port;

/*
   Finds the named Port in the PORTTABLE and changes the port entry to
   indicate that the port is no longer in use.
   Assumes the PORTTABLE is UNLOCKED and does its own locking.
*/

{
   portlist *PortList;
   register int i, Found=FALSE;

   if ((Lock (PORTTABLE)) == ERR)
      {
       WriteLog ("FreePort: Can't lock", PORTTABLE, "", "");
       return (ERR);
      }

   if ((PortList = GetPorts ()) == NULL)
      {
       WriteLog ("FreePort: can't read port list to mark", Port, "in use", "");
       UnLock (PORTTABLE);
       return (ERR);
      }

   for (i=0; (PortList[i] != NULL); i++)
      if (EQUALS(PortList[i]->Port, Port))
         {
          Found = TRUE;
          break;
         }

   if (!Found)
      {
       WriteLog ("FreePort: can't find", Port, "in", PORTTABLE);
       UnLock (PORTTABLE);
       PSFree (PortList);
       return (ERR);
      }

   free (PortList[i]->Site);
   PortList[i]->Site = malloc (strlen ("free")+1);
   strcpy (PortList[i]->Site, "free");
   PortList[i]->State = AVAILABLE;

   if (PutPorts (PortList) == ERR)
      {
       WriteLog ("FreePort: can't write port list to mark", Port, "in use", "");
       UnLock (PORTTABLE);
       PSFree (PortList);
       return (ERR);
      }

   UnLock (PORTTABLE);
   PSFree (PortList);
   return (GOOD);
}
```

```
int PutPorts (PortList)

portlist *PortList;

/*
   Writes the contenets of PortList over the current PORTTABLE.
   Assumes the PORTTABLE is LOCKED.
*/

{
    FILE *PortTableFd;
    register int i;

    if ((PortTableFd = fopen (PORTTABLE, "w")) == NULL)
       {
        WriteLog ("PutPorts: can't open", PORTTABLE, "for update", "");
        return (ERR);
       }

    for (i=0; PortList[i] != NULL; i++)
        fprintf (PortTableFd, "%c%s %s %c\n", FIELDMARK, PortList[i]->Port,
                 PortList[i]->Site, PortList[i]->State);

    fclose (PortTableFd);

    return (GOOD);
}




portlist *GetPorts ()

/*
  Reads the PORTTABLE and builds an structure of port entries.
  Uses malloc() to get storage for the struct.
  Returns a pointer to the struct in PortList.
  Returns NULL if there is an error, pointer to list if all is well.
  End of array is marked by PortList[n] == NULL.
  Assumes PORTTABLE is LOCKED.
*/

{
#define MASK 0x00ff

   portlist *PortList;
   FILE     *PortTableFd;
   char     TempBuf[LINELEN+1];
   sitename Site;
   pathname Port;
   int      State=0;
   register int i, c, NumPorts;
   char     x;  /* Added - SLG */
```

```
    if ((PortTableFd = fopen (PORTTABLE, "r")) == NULL)
        {
         WriteLog ("GetPorts: Can't open", PORTTABLE, "", "");
         return (NULL);                          /* indicate failure */
        }

    /* count the ports so we can malloc() for PortList */
    for (NumPorts=0; (c=getc(PortTableFd)) != EOF; NumPorts++)
        SkipEOL (PortTableFd);
    PortList = (portlist *) malloc ((NumPorts+1) * sizeof(portentry *));

    rewind (PortTableFd);

    for (i=0; (((c=getc(PortTableFd)) != EOF) && (i < NumPorts)); i++)
        {
         PortList[i] = (portentry *) malloc (sizeof (portentry));
         fscanf (PortTableFd, "%s %s %c", Port, Site, &x);
         PortList[i]->Port = malloc (strlen(Port)+1);
         strcpy (PortList[i]->Port, Port);
         PortList[i]->Site = malloc (strlen(Site)+1);
         strcpy (PortList[i]->Site, Site);
         State = (int) x;
         /* State &= MASK;  -- Bill: this does not work on ONYX */
         PortList[i]->State = State;
         SkipEOL (PortTableFd);       /* goto next entry */
        }

    PortList[i] = NULL;  /* set end of list marker */

    fclose (PortTableFd);

    return (PortList); /* good return code */
}




int ValidPort (PortName)

char *PortName;

/*  Returns TRUE if PortName is defined in PORTTABLE, FALSE otherwise. */

{
    portlist *PortList;
    register int i, Found;

    if ((Lock (PORTTABLE)) == ERR)
        {
         WriteLog ("ValidPort: Can't lock", PORTTABLE, "", "");
         return (ERR);
        }

    if ((PortList = GetPorts ()) == NULL)
```

```
      {
       WriteLog ("ValidPort: can't read port list", "", "", "");
       UnLock (PORTTABLE);
       return (ERR);
      }

   for (i=0, Found=FALSE; ((!Found) && (PortList[i] != NULL)); i++)
      if (EQUALS(PortList[i]->Port, PortName))
         Found = TRUE;

   PSFree (PortList);
   UnLock (PORTTABLE);

   return (Found);
}




void PSFree (PortList)

portlist *PortList;  /* Structure to be freed */

{
   int i;

   for (i = 0; PortList[i] != NULL; i++)
       {
        free (PortList[i]->Port);
        free (PortList[i]->Site);
        free (PortList[i]);
       }

   free (PortList);
}
```

```
#include "net.h"

extern int DebugLevel;

int  PutSite (SiteEntry)

site     *SiteEntry;

/* Looks up SiteEntry in SITETABLE and replaces its status information */
/* with the info in structure pointed to by SiteEntry.  If the site is not */
/* defined in the SITETABLE, an ERR will be indicated by the return    */
/* code. If the copy succeeds, the storage used for the site entry will be */
/* be released.                                                        */

{
   FILE     *SiteTableFd;
   register int c;
   register int n=0;
   long     offset;
   char     Name [SITENAMELEN];

   if ((Lock (SITETABLE)) == ERR)
      {
       WriteLog ("PutSite: can't lock", SITETABLE, "", "");
       exit (1);
      }

   if ((SiteTableFd = fopen (SITETABLE, "r+")) == NULL)
      {
       WriteLog ("PutSite: Can't open", SITETABLE, "", "");
       UnLock (SITETABLE);
       return (ERR);
      }

   /* scan through file until site found or EOF reached */

   getc (SiteTableFd);  /* skip first colon */
   do {
       fscanf (SiteTableFd, "%s", Name);  /* get a site  */
       if (EQUALS (SiteEntry->SiteName, Name))  /* it is the one we seek */
          {
           /* copy the site status information */
           offset = ftell (SiteTableFd);

#ifdef XENIX_V
           fclose (SiteTableFd);
           SiteTableFd = fopen (SITETABLE, "r+");
#endif

           fseek (SiteTableFd, ++offset, 0);  /* reset for output */
           fprintf (SiteTableFd, "%.1d %.1d %.9ld", SiteEntry->Status,
                   SiteEntry->NumCalls, SiteEntry->TimeToCall);
           fclose (SiteTableFd);
           if (DebugLevel)
             WriteLog ("PutSite:", SiteEntry->SiteName, "has been modified",
                       "");
           UnLock (SITETABLE);
           return (GOOD);  /* exit and indicate successful copy */
          }
       else
```

```
        /* advance to start of next site entry */
        do {
           SkipEOL (SiteTableFd);       /* skip to next line */
           c = getc (SiteTableFd);      /* read first character */
        } while ((c != FIELDMARK) && (c != EOF));

     } while (c != EOF);

  /* we wind up here if the site is not defined */

  WriteLog ("PutSite:", SiteEntry->SiteName, "not defined in",
            SITETABLE);
  fclose (SiteTableFd);
  UnLock (SITETABLE);

  return (ERR);         /* exit and indicate error */
```

```
#include "net.h"

site *ReadSite (Fd)

FILE *Fd;

/*
        ReadSite - read a site entry from the stream Fd into a static site
                   structure declared here; return pointer to the struct.
*/

{
   static site SiteStore;   /* Define entire structure as static */
   site *Site;
   int c;
   int i=0;
   static int KStatus, KNumCalls;   /* Temporary 'read' variables */
   static long KTimeToCall;
   static char KSysType;
   pathname TempBuf;

   Site = &SiteStore;

   fscanf (Fd, "%s", TempBuf);
   strcpy ( Site->SiteName, TempBuf );

   /* Read structure entries using temporary automatic variables so
      that this will work with all flavors of C.  Some C's don't
      guarantee structure element alignment, causing the expression
      &(struc->elem) to yield an incorrect value.                    */

   fscanf (Fd, "%d %d %ld %c %s", &KStatus, &KNumCalls, &KTimeToCall,
                                  &KSysType, TempBuf);

   Site->Status = (short) KStatus;      /* Copy aligned data into structure */
   Site->NumCalls = (short) KNumCalls;
   Site->TimeToCall = KTimeToCall;
   Site->SysType = KSysType;

   strcpy ( Site->Password, TempBuf );

   SkipEOL (Fd); /* phone numbers begin on next line */

   c = getc (Fd);  /* read in the array of phone numbers */
   while ((c != NL) && (c != BLANK) && (c != FIELDMARK) &&
          (c != EOF) && (i <= MAXPHONENUMS))
      {
       ungetc (c, Fd);
       fscanf (Fd, "%s", TempBuf);
       strcpy (Site->PhoneNum [i], TempBuf);
       ++i;
       SkipEOL (Fd);      /* go to next line             */
       c = getc (Fd);     /* and get the first character */
      }

   *Site->PhoneNum [i] = '\0';       /* mark end of list */

   return (Site);
}
```

```
#include "net.h"
#include <signal.h>
#include "iocontrol.h"
#include "iocontrol.e"

extern int TimeOut;  /* Maximum delay before timeout should occur */
extern int HoneyTime;

/*
   Receive - get a byte from the remote system (via the modem), waiting
             a maximum of TimeOut seconds for the byte.  If a character
             arrives, return it to the caller.  If a byte is not
             received within the threshold, return the error condition.
*/

int Receive ()

{
   int alrmint();

   int i = 0;          /* Initialize wait to 0 */
   char *cptr;         /* Character to read from port */
   unsigned char c;    /* Character to return */

   cptr = malloc (1); /* Just allocate a byte */

   signal (SIGALRM, alrmint);

#ifndef FORHONEY
   alarm (TimeOut);         /* Set timeout interval */
#else
   alarm (HoneyTime);       /* Honeywell requires much longer timeout */
#endif

   if (read (ModemFd, cptr, 1) > 0)
      {
       alarm (0);           /* reset timer */
       c = *cptr;           /* Get that unsigned 8-bit character */
       free (cptr);         /* Return storage */
       return ((int) c);    /* We got a character */
      }

   alarm (0);
   free (cptr);
   return (ERR);       /* Return ERR if timeout or error on read */
}




int alrmint ()
{
   return (ERR);
}
```

```
#include "net.h"

#define FATAL 40

int Remove (FileName, QueueName)

filename FileName;
pathname QueueName;

{
   pathname TempFileName;   /* Complete path name of file to remove */

   sprintf (TempFileName, "%s/%s", QueueName, FileName);

   if (unlink (TempFileName) == ERR)
      {
       WriteLog ("Remove: FATAL: can't delete", TempFileName, "", "");
       exit (FATAL);  /* This is fatal.. cannot dequeue! */
      }
}
```

```
#include "net.h"

extern int DebugLevel;

   /*
   This routine will check to see that the site entry pointed to by Site
   is not declared down. If it is, the routine will look at the
   TimeToCall field and see if it is time to try again.
   */

int State (Site) /* returns the value of Site->Status (UP, RETRY, or DOWN) */

site *Site;

{

   if (Site->Status != UP)
      {
       if (NOW >= Site->TimeToCall)
          {
           if (DebugLevel)
              WriteLog ("State: retry time reached for", Site->SiteName,
                        "", "");
           Site->Status = UP;
           Site->TimeToCall = 0;
          }
      }

   if (DebugLevel > 1)
      switch (Site->Status)
         {
          case UP    : WriteLog ("State:", Site->SiteName, "is", "UP");
                       break;
          case RETRY : WriteLog ("State:", Site->SiteName, "is", "RETRY");
                       break;
          case DOWN  : WriteLog ("State:", Site->SiteName, "is", "DOWN");
                       break;
          default : WriteLog ("State:", Site->SiteName, "status undefined", "");
         }

   return (Site->Status);
}
```

```
#include "net.h"


char *StripMe (Path, FirstSite)

char *Path, *FirstSite;

/* Removes the first site name from Path and returns a pointer to the */
/* rest of the path. If there is only one site name left in Path, a   */
/* NULL is returned. Path is not altered, and FirstSite points to the */
/* site name that has been stripped off the front of the path.        */

{
   register int i = 0;

   while ((Path [i] != SEPCHAR) && (Path [i] != '\0'))
      {
       FirstSite [i] = Path [i];
       i++;
      }

   FirstSite [i] = '\0';

   return ((Path [i] == '\0') ? NULL : Path + i + 1);
}
```

```
#include "net.h"

site *ValidSite (Site)

char *Site;

/* Validates the existence of Site in SITETABLE and returns as  */
/* its value a pointer to a site table entry in a struct defined    */
/* by "site" in net.h                                               */
/* If the site is not found in the table, it returns NULL           */

{
   FILE     *SiteTableFd;
   site     *SiteEntry;
   register int c, n=0;
   long     Start;
   pathname TempBuf;

   if ((Lock (SITETABLE)) == ERR)
      {
       WriteLog ("ValidSite: Can't lock", SITETABLE, "", "");
       return (NULL);
      }

   if ((SiteTableFd = fopen (SITETABLE, "r")) == NULL)
      {
       WriteLog ("ValidSite: Can't open", SITETABLE, "", "");
       UnLock (SITETABLE);
       return (NULL);                          /* indicate failure */
      }

   /* skip down to beginning of first site entry */
   while ((c = getc (SiteTableFd)) != FIELDMARK) SkipEOL (SiteTableFd);

   do {
       Start = ftell (SiteTableFd); /* save pointer to start of entry */
       fscanf (SiteTableFd, "%s", TempBuf);  /* get a site name */
       if (EQUALS (TempBuf, Site))  /* it is the one we seek */
          {
           fseek (SiteTableFd, Start, 0); /* reset file pointer */
           SiteEntry = ReadSite (SiteTableFd); /* read in the site entry */
           fclose (SiteTableFd);
           UnLock (SITETABLE);
           return (SiteEntry);  /* return pointer to the site structure */
          }
       else
           do {
              c = getc (SiteTableFd);      /* read first character */
           } while ((c != FIELDMARK) && (c != EOF));

      } while (c != EOF);

   fclose (SiteTableFd);
   UnLock (SITETABLE);
   return (NULL);  /* indicate site not found */
```

```
#include "net.h"

extern pathname LogFile;

int WriteLog (P1, P2, P3, P4)

char *P1, *P2, *P3, *P4;

{
   FILE *LogFd;
   char Date [26];
   register int i=0;

   DateTime (Date);

   if ((LogFd = fopen (LogFile, "a")) == NULL)
         {
          fprintf (stderr, "WriteLog: Can't open %s.\n", LogFile);
          fprintf (stderr, "%s %s %s %s %s\n", Date, P1, P2, P3, P4);
          return (ERR);
         }

   setbuf (LogFd, (char *) NULL);

   fprintf (LogFd, "%s %s %s %s %s\n", Date, P1, P2, P3, P4);

   fclose (LogFd);

   return (GOOD);

}
```

QMS

This section contains the functions used only by the Queue Manager/Scheduler program (QMS).

```
qms : net.h qms.o schedule.o getsites.o ports.o getdest.o\
      movepri.o dequeue.o getdir.o lockfile.o writelog.o datetime.o\
      readsite.o stripme.o frename.o interrupt.o validsite.o myname.o\
      fileopen.o state.o givetomp.o abort.o putsite.o
        cc -o qms qms.o schedule.o getsites.o ports.o getdest.o\
      movepri.o dequeue.o getdir.o lockfile.o writelog.o datetime.o\
      readsite.o stripme.o frename.o interrupt.o validsite.o myname.o\
      fileopen.o state.o givetomp.o abort.o putsite.o
        strip qms

qms.o : net.h wait.h qms.c
        cc -c -O qms.c

schedule.o: net.h schedule.c
        cc -c -O schedule.c

getsites.o: net.h getsites.c
        cc -c -O getsites.c

getdest.o : net.h getdest.c
        cc -c -O getdest.c

movepri.o : net.h movepri.c
        cc -c -O movepri.c

interrupt.o: net.h interrupt.c
        cc -c -O interrupt.c
```

```
#include "net.h"

int GetDest (MsgFile, SiteName)
char *MsgFile;
char *SiteName;

/*
   Open the MsgFile and get the destination site from the path line.
   Check to be sure that the site is valid. Return ERR if anything is
   wrong, 0 if all is well. Return name of site in SiteName.
*/

{
   pathname Path;
   FILE *Fd;

   if ((Fd = fopen(MsgFile, "r")) == NULL)  /* open the message file */
      {
       WriteLog ("GetDest: can't open", MsgFile, "", "");
       return (ERR);
      }

   SkipEOL (Fd);                    /* skip over the priority line       */
   fscanf (Fd, "%*c%s", Path);      /* read in the full destination path */
   StripMe (Path, SiteName);        /* strip off the first site          */

   fclose (Fd);                     /* close the message file */

   if (ValidSite(SiteName) == NULL)  /* make sure the site is valid */
      return (ERR);

   return (0);                      /* good return code */

}
```

```
#include "net.h"

char **GetSites ()

/*
  Reads the SITETABLE and builds an array of site names. Uses malloc() to
  get storage for the list.  Returns a pointer to the list in SiteList.
  Returns NULL if there is an error, 0 if all is well.
  End of array is marked by SiteList[n] == NULL.
*/

{
   char **SiteList;
   char     TempBuf[LINELEN+1];
   sitename Site;
   FILE     *SiteTableFd;
   register int c, NumSites, i=0;

   if ((Lock (SITETABLE)) == ERR)
      {
       WriteLog ("GetSites: Can't lock", SITETABLE, "", "");
       return (NULL);
      }

   if ((SiteTableFd = fopen (SITETABLE, "r")) == NULL)
      {
       WriteLog ("GetSites: Can't open", SITETABLE, "", "");
       UnLock (SITETABLE);
       return (NULL);                          /* indicate failure */
      }

   /* count the sites so we can malloc() for SiteList */
   for (NumSites=0; ((c=getc(SiteTableFd)) != EOF); )
      {
       if (c == FIELDMARK)  /* then this line is a site entry */
          NumSites++;
       SkipEOL (SiteTableFd);
      }
   SiteList = (char **) malloc ((NumSites+1) * sizeof(char *));

   /* skip down to beginning of first site entry */
   rewind (SiteTableFd);
   while ((c = getc (SiteTableFd)) != FIELDMARK) SkipEOL (SiteTableFd);

   do {
       fscanf (SiteTableFd, "%s", TempBuf);  /* get a site name */
       SiteList[i] = malloc (strlen(TempBuf)+1);
       strcpy (SiteList[i++], TempBuf);
       do {
           SkipEOL (SiteTableFd);       /* goto next entry */
           c = getc (SiteTableFd);
          } while ((c != FIELDMARK) && (c != EOF));

      } while (c != EOF);

   SiteList[i] = NULL;  /* set end of list marker */
   fclose (SiteTableFd);
   UnLock (SITETABLE);

   return (SiteList);
```

```
}
#include "net.h"

int Interrupt (Queue)
char *Queue;

/*
   Write an interrupt flag in the named queue to cause the currently
   running IOControl to terminate.
*/

{
   pathname FileName;
   FILE *Fd;
   register int i=0;

   sprintf (FileName, "%s/%s", Queue, INTFILE);

   /* create the interrupt flag file */

   if ((Fd = fopen (FileName, "w")) == NULL)
      {
       WriteLog ("Interrupt: can't create", FileName, "", "");
       return (ERR);
      }
   else
      fclose (Fd);

   /* wait for Caller to remove the file after IOControl surrenders the port */

   while (((Fd = fopen (FileName, "r")) != NULL) && (i++ < 10))
        {
         fclose (Fd);
         sleep (15);       /* Could be made into a runtime parameter */
        }

   if (i >= 10)  /* then the file was never removed */
      {
       fclose (Fd);
       unlink (FileName);
       return (ERR);
      }

   return (GOOD);
}
```

```
#include "net.h"

int MovePRI (FileName, QueueName)
char *FileName;
char *QueueName;

/*
  Move the named file from the priority queue to the named system queue.
  Return ERR if there are any problems, 0 if all goes well.
*/

{
   pathname OldPath;
   pathname NewPath;

   sprintf (OldPath, "%s/%s", PRIORQ, FileName);
   sprintf (NewPath, "%s/%c%s", QueueName, PRIORTYPE, FileName+1);

   WriteLog ("MovePRI: moving", OldPath, "to", NewPath);
   return (FRename (OldPath, NewPath));

}
```

```
#include "net.h"
#include <signal.h>

#define ABORTFILE ".abort"

pathname LogFile;      /* global log file for QMS routines */
int DebugLevel = 0;    /* runtime debug level (0 = normal) */
int ForwardFlag = TRUE;  /* Set default forwarding of messages */

main (argc, argv)
int argc;
char **argv;

/*
   This is the main program for the communications system. It monitors
   the contents of the priority queue and the system queues, scheduling
   Callers to service the messages in those queues.

   It is invoked with the name of the directory to be used as the MASTERQ
   where all the other queues and directories are located.

   The program will check the queues for waiting messages, giving priority
   to those in the PRIORQ. It will call Schedule() to start a Caller
   process for each queue containing messages for transmission. It will
   continue to invoke Callers until all queues are serviced or there are
   no more available ports for the Caller to use.

   If there are no available ports and a Priority message is waiting, it
   will sieze a port from a Caller engaged in routine transmission. Callers
   engaged in priority transmissions will not be interrupted. If there is a
   Priority message for a site with which a Caller is currently engaged in
   a routine transmission, the Priority messages will be inserted into the
   transmission as soon as possible.

   The program may be run periodically by the cron process, or it may be
   invoked with the '-' option which will cause it to run continuously
   in the background. It creates a lockfile on the master directory to
   prevent multiple copies of the program from executing.

   If the file ".abort" ever appears in the master directory, the program
   will terminate and remove its associated lock file.
*/

{
   char *NextFile;
   char **QueueList;
   sitename Queue;
   pathname MasterQueue;
   pathname FileName;
   pathname LockName;
   register int i;
   register int Forever = FALSE;  /* assume single pass, option set below */
   char Level[3];              /* string version of debug level */
   FILE *ParamFileFp;          /* Parameter file pointer */
   char Key[20];
   int  Value;
   int  Fd;
   int PollDelay = 60;         /* delay between processing scans in "-" mode */

   void ShutDown ();
```

```
   signal (SIGTERM, ShutDown);

   umask (UMASK);

   /* validate and parse arguments */
   if ((argc < 2) || (argc > 4))
      usage (argv[0]);

   for (--argc; argc > 0; argc--)
      switch (argv[argc][0])
         {
          case '-': switch (argv[argc][1])
                       {
                        case '\0': Forever = TRUE;
                                   break;
                        case 'd' : sscanf (argv[argc]+2, "%d", &DebugLevel);
                                   sprintf (Level, "%d", DebugLevel);
                                   break;
                        default  : usage (argv[0]);
                       }
          default : strcpy(MasterQueue,argv[argc]); /* get working directory*/
         }

   /* set working directory */
   if ((chdir (MasterQueue)) != 0)
      {
       fprintf (stderr, "%s: invalid directory: %s\n", argv[0], MasterQueue);
       usage (argv[0]);
      }

   /* set the global LogFile for WriteLog() */
   sprintf (LogFile, "log/%s.log", QMS);

   if (Forever)
      {
       WriteLog ("QMS: activated in Scanner mode in", MasterQueue, "", "");

       if ((ParamFileFp = fopen (PARAMFILE, "r")) != NULL)
          {
           fscanf (ParamFileFp, "%s %d\n", Key, &Value);
           while ((!feof (ParamFileFp)) && (!ferror (ParamFileFp)))
              {
               if (EQUALS (Key, "qmspoll"))
                  PollDelay = Value;
               else if (EQUALS (Key, "forwarding"))
                  ForwardFlag = Value;

               fscanf (ParamFileFp, "%s %d\n", Key, &Value);
              }

           fclose (ParamFileFp);
          }
      }
   else
       WriteLog ("QMS: activated in One-Pass mode in", MasterQueue, "", "");

   if (DebugLevel)
      WriteLog ("QMS:", "system debug level is", Level, "");

   /* only allow one copy of QMS at a time */
```

```
    if (Lock(QMS) == ERR)
        {
         WriteLog ("QMS: can't lock", QMS, "(already running)", "");
         exit (ERR);
        }

    /* Remove system ABORT file */
    unlink (ABORTFILE);

    /* process each system queue */
    if ((QueueList = GetSites ()) == NULL)  /* uses malloc() to get storage */
        {
         WriteLog ("QMS: can't get site list - GoodBye!", "", "", "");
         UnLock (QMS);
         exit (1);
        }

    do {
         if (Abort (MasterQueue))     /* Always check for abort signal */
            ShutDown ();
         for (i=0; QueueList[i] != NULL; i++)
            {
             /* process all the messages in the priority queue first */
             if (DebugLevel > 2)
                WriteLog ("QMS: checking", PRIORQ, "", "");
             while ((NextFile = DeQueue (PRIORQ)) != NULL)
                {
                 if (DebugLevel)
                    WriteLog ("QMS: processing", PRIORQ, "", "");
                 sprintf (FileName, "%s/%s", PRIORQ, NextFile);
                 if (GetDest (FileName, Queue) != ERR)
                    if (MovePRI (NextFile, Queue) != ERR)
                       if (Schedule (Queue, PRIORTYPE) != GOOD)
                          WriteLog ("QMS: can't schedule", Queue,
                                    "for priority call", "");
                }

             if (DebugLevel > 1)
                WriteLog ("QMS: checking", QueueList[i], "", "");

             if (ValidSite(QueueList[i])->SysType == EMULATED)
                continue;

             sprintf (LockName, "%s.LCK", QueueList[i]);
             if ((Fd = open (LockName, O_RDONLY)) >= 0)
                {
                 close (Fd);
                 if (DebugLevel)
                    WriteLog ("QMS:", QueueList[i], "is locked", "");
                }
             else
                {
                 if ((NextFile = DeQueue (QueueList[i])) != NULL)
                    {
                     if (DebugLevel)
                        WriteLog ("QMS: processing", QueueList[i], "", "");
                     switch (Schedule (QueueList[i], ROUTINETYPE))
                        {
                         case BUSY : /* all ports are in use - wait for one */
                              if (DebugLevel)
                                 WriteLog ("Schedule: waiting for a Caller",
```

```
                                        "to terminate", "", "");
                            sleep (PollDelay);
                            break;

                        case GOOD : break;

                        case ERR :
                        default  :
                            WriteLog ("QMS: can't schedule", QueueList[i],
                                      "for routine call", "");
                            break;
                        }
                    }
                }

            if (Abort (MasterQueue))    /* Always check for abort signal */
                ShutDown ();
            }  /* end of for loop */

        if (Forever)    /* sleep between polls */
            sleep (PollDelay);

    } while (Forever);    /* run only once unless '-' option was set */

    free (QueueList);      /* release storage */

    WriteLog ("QMS: processing complete - normal termination", "", "", "");

    UnLock (QMS);  /* unlock the QMS guard */

    exit (GOOD);
}



int usage (Name)
char *Name;
{
    fprintf (stderr, "usage: %s directory [-]\n", Name);
    exit (1);
}
```

```
void ShutDown ()

{
   WriteLog ("ShutDown:", "operator requested system shutdown", "", "");
   UnLock (QMS);
   exit (0);
}
```

```
#include "net.h"

extern int DebugLevel;

int Schedule (SiteName, Type)
char *SiteName;
int  Type;

/*
   This routine will try to schedule a call to the named Site.  It will
   read the port table to see if there is already a conversation with
   that site in progress.  If not, it will try to get an available port
   for use.  If a port is available, a Caller program will be invoked
   with the Site name and Port Name and we return GOOD. If there is
   already a conversation in progress, we return GOOD. If there are no
   ports available we return ERR. If we cannot run a Caller, we return
   ERR.

   If the Type of call is "high priority" and a port is not available,
   we will attempt to seize a port from a currently operating Caller.
   When the caller is started, the Port Table will be marked to indicate
   the nature of the call (ROUTINE or RPIORITY).
*/

{
   portlist *PortList;  /* storage allocated by GetPorts() */
   pathname SeizePort;
   sitename SeizeSite;
   pathname PortName;
   pathname Caller;
   register int Talking = FALSE;
   register int Result;
   register int i;
   register int Condition;
   site     *Site;
   char     Response[10];
   char     *ThisName;    /* Pointer to the name of this system */

   void     PSFree ();    /* Routine to free storage for ports structure */
                          /* (Declared in common/ports.c)                */

   ThisName = MyName ();  /* We want to be able to deallocate the storage */

   if (EQUALS(SiteName, ThisName))
      {
       free (ThisName);
       return (GOOD);
      }

   free (ThisName);

   if ((Site = ValidSite (SiteName)) == NULL)  /* Site is invalid */
      {
       WriteLog ("Schedule:", "invalid site", SiteName, "");
       return (ERR);
      }
   else /* see if site is UP or DOWN */
      if ( (Condition = State (Site)) != UP )  /* could be RETRY or DOWN */
         if (Type != PRIORTYPE)
            {
```

```
            if (DebugLevel)
               WriteLog ("Schedule: recall time not reached for",
                         SiteName, "", "");
            if (Condition == DOWN) /* need to forward messages */
               GiveToMP (SiteName);

            /* Free storage for site some day */

            return (GOOD);
           }
         else
           {
            if (DebugLevel)
               WriteLog ("Schedule:", "scheduling priority call for",
                         SiteName, "(possibly down)");

            Site->TimeToCall = NOW;  /* Reset to try to call again */
            Site->Status = UP;       /* Force caller to poll site */

            PutSite (Site); /* Storage for site freed */
           }

   /* arrive here if site is UP or retry time has elapsed */

   if ((Lock (PORTTABLE)) == ERR)
      {
       WriteLog ("Schedule: Can't lock", PORTTABLE, "", "");
       return (ERR);
      }

   if ((PortList = GetPorts ()) == NULL)  /* uses malloc() to get storage */
      {
       WriteLog ("Schedule: can't get port list - GoodBye!", "", "", "");
       UnLock (PORTTABLE);
       return (ERR);
      }

   strcpy (SeizeSite, "");
   strcpy (SeizePort, "");
   strcpy (PortName, "");
   for (i=0; PortList[i] != NULL; i++)
      {
       /* if we are already talking to the Site we can quit */
       if (EQUALS(PortList[i]->Site, SiteName))
          {
           Talking = TRUE;
           break;
          }

       /* make note of first available port */
       if ((PortList[i]->State == AVAILABLE) && (strlen(PortName) == 0))
           strcpy (PortName, PortList[i]->Port);   /* copy port name */

       /* make note of first non-priority Caller */
       if ((PortList[i]->State != PRIORTYPE) && (strlen(SeizePort) == 0))
          {
           strcpy (SeizeSite, PortList[i]->Site);  /* copy site name */
           strcpy (SeizePort, PortList[i]->Port);  /* copy port name */
          }
      }
```

```
   PSFree (PortList);  /* release the storage */

   if (Talking)
      {
       if (DebugLevel)
          WriteLog ("Schedule: already talking to", SiteName,
                    "- no need to call", "");

       UnLock (PORTTABLE);
       return (GOOD);
      }

   if (strlen(PortName) == 0) /* no available port */
      if ((Type == PRIORTYPE) && (strlen(SeizePort) != 0)) /* can seize one */
         {
          WriteLog ("Schedule: interrupting", SeizeSite, "to get", SeizePort);
          strcpy (PortName, SeizePort);
          if (Interrupt (SeizeSite) == ERR)
             {
              WriteLog ("Schedule: can't interrupt", SeizeSite, "", "");
              UnLock (PORTTABLE);
              return (BUSY);
             }
         }
      else
         {
          if (DebugLevel)
             WriteLog ("Schedule: no ports available at this time", "", "", "");

          UnLock (PORTTABLE);
          return (BUSY);
         }

   /* mark the chosen port as IN USE */
   TakePort (PortName, SiteName, Type);
   UnLock (PORTTABLE);

   /* only one caller per system allowed */
   if (Lock(SiteName) == ERR)
      {
       WriteLog ("Schedule: can't lock", SiteName, "", "");
       FreePort (PortName);
       return (ERR);
      }

   /* build the shell command to execute CALLER */
   if (DebugLevel)
      sprintf (Caller, "%s/%s %s %s -d%d &", BIN, CALLER, SiteName, PortName,
                                             DebugLevel);
   else
      sprintf (Caller, "%s/%s %s %s &", BIN, CALLER, SiteName, PortName);

   /* start a Caller for the given Site on the desired Port */
   if ((Result = system (Caller)) != GOOD)
      {
       sprintf (Response, "%d", Result);
       WriteLog ("Schedule: system call returned", Response, "", "");
       FreePort (PortName);  /* free up the port */
       return (ERR);
      }
```

```
    if (DebugLevel)
       WriteLog ("Schedule: loaded Caller for", SiteName, "on", PortName);

    return (GOOD);
}
```

CALLER

This section contains the functions used only by the Caller program (CALLER).

```
caller: caller.o givetomp.o checkdown.o dial.o validsite.o putsite.o\
      openmodem.o getdir.o dequeue.o state.o hangup.o writelog.o frename.o\
      lockfile.o readsite.o readstr.o datetime.o receive.o sethayes.o\
      login.o myname.o fileopen.o ports.o getprompt.o endswith.o errmsg.o
        cc -o caller caller.o givetomp.o checkdown.o dial.o validsite.o\
      putsite.o openmodem.o getdir.o dequeue.o state.o hangup.o\
      writelog.o frename.o lockfile.o readsite.o readstr.o\
      datetime.o receive.o sethayes.o login.o myname.o fileopen.o\
      ports.o getprompt.o endswith.o errmsg.o
        strip caller

caller.o: net.h caller.c sysdef.h wait.h retcodes.h
        cc -c -O caller.c

checkdown.o: net.h checkdown.c
        cc -c -O checkdown.c

dial.o: net.h dial.c
        cc -c -O dial.c

openmodem.o: net.h openmodem.c
        cc -c -O openmodem.c

hangup.o: net.h hangup.c
        cc -c -O hangup.c

readstr.o: net.h readstr.c
        cc -c -O readstr.c

sethayes.o: net.h sethayes.c
        cc -c -O sethayes.c

login.o: net.h login.c
        cc -c -O login.c

getprompt.o: net.h iocontrol.h getprompt.c
        cc -c -O getprompt.c

endswith.o: net.h endswith.c
        cc -c -O endswith.c

errmsg.o: retcodes.h errmsg.c
        cc -c -O errmsg.c
```

```
extern int MaxCalls;
extern int RetryDelay;
extern int DownDelay;
```

```
#include "wait.h"
#include "net.h"
#include "retcodes.h"
#include <signal.h>

#define MODEMTIMEOUT 12

int ModemFd; /* OpenModem() will return a file descriptor for the modem       */
site *Site;  /* allocated by ValidSite()                                      */
pathname LogFile;  /* global LogFile for use by WriteLog()                    */
sitename SysName;  /* name of the system to be called                         */
pathname PortName; /* name of the port to use                                 */
pathname IntFileName; /* Name of interrupt file to remove                     */
int IntFlag = FALSE;  /* Flag indicating if interrupt occurred                */
int DebugLevel = 0;   /* Runtime debug level (0 = normal)                     */
int MaxCalls = 2;     /* Maximum retry-calls to a site before determined down */
int RetryDelay = 60;  /* Minimum retry time in seconds between calls to site  */
int DownDelay = 180;  /* Minimum delay in seconds before retry of downed site */
int MaxWait = 10;     /* Maximum retry for hangup response                    */
int TimeOut = 10;     /* Timeout (in seconds) for reading a single character  */
int HoneyTime = 30;   /* Timeout for Honeywell versions                       */
int ForwardFlag = TRUE;  /* Set default forwarding of messages                */
#ifdef XENIX
   FILE *File;  /* for resetting O_NDELAY under XENIX */
#endif


main (argc, argv)

int argc;
char ** argv;

/*
   The Caller program will take a system name and a port name as input
   parameters.  It will open the named port for dialout use and check for
   the presence of a modem.  It will get the connection information on the
   desired system and try to establish a connection with that system.  If
   that succeeds, it will call IOControl to send the files located in the
   directory and receive any datafiles from the remote system. Received file
   will be given to the Message Processor for disposition.

   Errors will be logged, and the program will terminate with either a good
   return code (0) or a bad return code (-1).

   If a remote site is called unsuccessfully, a record is kept in the site
   table.  When the number of unsuccessful attempts crosses a threshold
   value, the site is declared down and all traffic for the site is rerouted
   for a period of time.
*/

{
   pathname IOControl;
   union wait Status;
   register int pid;
   register int Condition;
   register int Connected=FALSE;
   FILE *ParamFileFp; /* Parameter file pointer (for retry information) */
   char Key[20];
   int Value;
   int MainTimeOut;
```

```
   void ShutDown ();  /* Define routine for system shutdown */
   char *ErrMsg ();   /* Define error message routine */

   umask (UMASK);

   /* get the arguments and prepare for action */
   if (argc != 3)
      if ((argc == 4) && (argv [3][0] == '-') && (argv [3][1] == 'd'))
         {
          argc--;      /* Remove last argument */
          sscanf (argv[3]+2, "%d", &DebugLevel);  /* Get debug level (-d#) */
         }
      else
         {
          fprintf (stderr, "usage: %s system port\n", argv[0]);
          Release (); /* release resources */
          UnLock (SysName);
          exit (ERR);
         }

   strcpy (SysName, argv[1]);
   strcpy (PortName, argv[2]);
   sprintf (LogFile, "log/%s.log", SysName);

   if (DebugLevel)
      WriteLog ("Caller:", "system debug level is", argv[3]+2, "");

   if (!ValidPort(PortName))
      {
       WriteLog ("Caller: port", PortName, "not found in", PORTTABLE);
       Release (); /* release resources */
       UnLock (SysName);
       exit (ERR);
      }

   /* open the modem port and make sure the modem is awake */
   if ((ModemFd = OpenModem(PortName)) == ERR)
      {
       WriteLog ("Caller: can't open", PortName, "to call", SysName);
       Release (); /* release resources */
       UnLock (SysName);
       exit (ERR);
      }

#ifdef XENIX
    /* reset O_NDELAY on XENIX system - see openmodem.c */
    if ((File = fopen (PortName, "r+")) == NULL)
       {
        WriteLog ("Caller: Can't turn off O_NDELAY on", PortName, "", "");
        Release ();
        UnLock (SysName);
        exit (ERR);
       }
     fprintf (File, "AT\r");
#endif

   if ((ParamFileFp = fopen (PARAMFILE, "r")) != NULL)
      {
       fscanf (ParamFileFp, "%s %d\n", Key, &Value);
       while ((!feof (ParamFileFp)) && (!ferror (ParamFileFp)))
```

```
        {
         if (EQUALS (Key, "maxcalls"))
            MaxCalls = Value;
         else if (EQUALS (Key, "retrydelay"))
            RetryDelay = Value;
         else if (EQUALS (Key, "downdelay"))
            DownDelay = Value;
         else if (EQUALS (Key, "maxhangup"))
            MaxWait = Value;
         else if (EQUALS (Key, "timeout"))
            TimeOut = Value;
         else if (EQUALS (Key, "timeout(gcos)"))
            HoneyTime = Value;
         else if (EQUALS (Key, "forwarding"))
            ForwardFlag = Value;

         fscanf (ParamFileFp, "%s %d\n", Key, &Value);
        }

     fclose (ParamFileFp);
    }

signal (SIGTERM, ShutDown);  /* Set interrupt handler */

/* Set timeout to specific value for interactions with modem */

MainTimeOut = TimeOut;
TimeOut = MODEMTIMEOUT;

/* put the modem in command mode and set the switches */
if (SetHayes (PortName) == ERR)
   {
    WriteLog ("Caller: can't configure modem on", PortName, "", "");
    Release (); /* release resources */
    UnLock (SysName);
    exit (ERR);
   }

FlushModemInput (ModemFd);

    if ((Site = ValidSite (SysName)) != NULL)  /* Site is valid */
      {

       if ( (Condition = State (Site)) == UP )
          {
           if (DebugLevel)
              WriteLog ("Caller: invoked for", SysName, "using", PortName);
           Connected = Dial (Site);
           if (Connected == TRUE)
               {
               if (DebugLevel)
                  WriteLog ("Caller: Connected to", SysName,
                            "-", "about to log in");

               TimeOut = MainTimeOut;   /* Restore main time-out value */
               if (Login (SysName, Site->Password, Site->SysType) == ERR)
                   {
                    if (CheckDown (Site))  /* record unsuccessful call */
                       GiveToMP (SysName); /* may need to forward mail */

                    UnLock (SysName);
```

```
        quit (ERR);
       }

    if ((pid = fork ()) == ERR)
       {
        WriteLog ("Caller: cannot fork - Goodbye!", "", "", "");
        UnLock (SysName);
        quit (ERR);
       }

    if (pid == 0) /* I am the child */
       {
        /* IOCONTROL does the transmitting and receiving */
        if (Site->SysType != GCOS)
           sprintf (IOControl, "%s/%s", BIN, IOCONTROL);
        else
           sprintf (IOControl, "%s/%s", BIN, IOCNTRLH);

        if (DebugLevel)
           execl (IOControl, IOControl, SysName, PortName,
                  argv[3], 0);  /* Pass debug level information */
        else
           execl (IOControl, IOControl, SysName, PortName, 0);

        WriteLog ("Caller: cannot exec", IOControl, "- Goodbye!",
                  "");
        UnLock (SysName);
        exit (ERR);
       }

    wait (&Status);           /* wait for IOCONTROL to complete */

    if (Status.w_S.w_Stopval == WSTOPPED)
       {
        /* This was a bad error; IOCONTROL did not exit cleanly */
        WriteLog ("Caller:", "IOControl aborted unexpectedly",
                  "", "");
        if (DebugLevel)
           WriteLog ("Caller: unlocking", SysName, "", "");
        if (UnLock (SysName) == ERR)
           WriteLog ("Caller: could not unlock", SysName,
                     "queue", "");
        quit (ERR);
       }
    else switch (Status.w_T.w_Retcode)
       {
        case GOOD :
           if (DebugLevel)
              WriteLog ("Caller: Conversation with", SysName,
                        "complete.", "");
           break;

        case INTERRUPTED :
           WriteLog ("Caller: Interrupted during",
                     "conversation with", SysName, "");
           IntFlag = TRUE;  /* Set interrupt flag for later */
           /* Construct interrupt file name to remove */
           sprintf (IntFileName, "%s/%s", SysName, INTFILE);
           break;

        case ABORTED :
```

```
                    WriteLog ("Caller:", "operator requested system",
                              "shutdown", "");
                    break;

                case ERR :
                default  :
                    WriteLog ("Caller:", IOCONTROL, "returned error:",
                              ErrMsg (Status.w_T.w_Retcode));
                    if (CheckDown (Site))  /* record unsuccessful call */
                       GiveToMP (SysName); /* may need to forward mail */
                    quit (ERR);
                }

            /* reset site table entry to indicate successful contact */

            if (!IntFlag)
                {
                 Site->Status = UP;       /* declare site up    */
                 Site->TimeToCall = NOW; /* reset time to call */
                 Site->NumCalls = 0;      /* reset retry count  */
                 PutSite (Site);          /* save entry & release storage */
                }

             quit (GOOD);          /* all is well - hang up phone & exit */
            }
         else          /* Connected=FALSE - remote system didn't answer */
            {
             WriteLog ("Caller: Can't connect to", SysName, "", "");
             UnLock (SysName);
             if (Connected == FALSE)     /* remote system didn't answer  */
               {
                if (CheckDown (Site))    /* if site is declared down     */
                   GiveToMP (SysName);  /* give messages to MSGPROC     */
               }
             else         /* Connected=ERR: modem didn't answer commands */
               {
                WriteLog ("Caller: we have a problem",
                          "with the modem at", PortName, "");
               }
            }
        }
      else    /* Condition != UP  ( may be DOWN or DELAY) */
          {
           if (Condition == DOWN)  /* may need to forward the messages */
              GiveToMP (SysName);
          }
     }
   else  /* ValidSite returned NULL */
     {
      WriteLog ("Caller: No entry for", SysName, "defined in", SITETABLE);
      /* take some action or notify operator of this situation */
     }

Release (); /* release resources */
exit (ERR); /* arrive here only if some error occurred above */
```

```
int quit (retcode)  /* hang up the phone and exit */

int retcode;

{
  TimeOut = MODEMTIMEOUT;
  if (!HangUp(Site->SysType))
      WriteLog ("Caller: could not hang up the phone", "", "", "");

  if (!IntFlag)
     Release ();                              /* release resources */
  else
     if (unlink (IntFileName) == ERR)          /* We must acknowledge */
        WriteLog ("Caller:", "could not remove", IntFileName, "");

  exit (retcode);
}
```

```
int Release ()

/*  free up the port and system queue taken by QMS for our use */

{
   if (DebugLevel)
      WriteLog ("Caller: freeing", PortName, "", "");
   if (FreePort (PortName) == ERR)
      WriteLog ("Caller: could not free", PortName, "after use", "");
}
```

```
void ShutDown ()

{
   WriteLog ("ShutDown:", "operator requested system shutdown", "", "");
   Release ();
   exit (0);
}
```

```
#include "net.h"
#include "params.e"

int CheckDown (RSite)

site *RSite;

{
/* Returns TRUE if site is declared DOWN, FALSE otherwise.              */
/* Sets callback time appropriately and updates number of failed calls. */
/* PutSite sets status and frees site entry storage.                    */

    RSite->NumCalls++;  /* increment the retry count */
    if (RSite->NumCalls > MaxCalls)
       {
        RSite->Status = DOWN;  /* declare the site down */
        RSite->NumCalls = MaxCalls-1;
        RSite->TimeToCall = NOW + DownDelay;
        PutSite (RSite);  /* save entry & release storage */
        WriteLog ("CheckDown:", RSite->SiteName, "is down.", "");
        return (TRUE);
       }
    else   /* RSite->NumCalls <= MaxCalls means try again later */
       {
        RSite->Status = RETRY;  /* declare the site delayed */
        RSite->TimeToCall = NOW + RetryDelay;
        WriteLog ("CheckDown:",RSite->SiteName,"temporarily down", "");
        PutSite (RSite);  /* save entry & release storage */
        return (FALSE);
       }
}
```

```
#include "net.h"

extern int  ModemFd;
extern int  DebugLevel;  /* Runtime debug level (0 = normal) */

int Dial (SiteInfo)

site *SiteInfo;
/* char *TelNums [20]; */

{
   char DialCmd [40];
   char Answer [10];
   register int i=0;

/* Using Hayes Modem - Minimal Code */

   FlushModemInput (ModemFd);

   write (ModemFd, ATTENTION, strlen(ATTENTION));

   if (ReadStr(Answer) == NULL)  /* modem never answered */
      {
       WriteLog ("Dial: modem does not respond - can't dial out","","","");
       return (ERR); /* don't count as failure, problem at this end */
      }

   if (!EQUALS (Answer, ZERO))
       WriteLog ("Dial: modem answered:", Answer, "to AT command", "");

   /* try calling each listed number for the site until one succeeds */
   for (i=0; *(SiteInfo->PhoneNum[i]) != '\0'; i++)
      {
       /* give the dial command and see what happens */
       sprintf (DialCmd, "%s%s\r", DIALSTR, SiteInfo->PhoneNum[i]);

       if (DebugLevel)
          WriteLog ("Dial: dialing", SiteInfo->PhoneNum[i],"", "");

       FlushModemInput (ModemFd);
       write (ModemFd, DialCmd, strlen (DialCmd));

       if (ReadStr(Answer) == NULL)
          {
           WriteLog ("Dial: no response to dial command,", "sending AT",
                     "", "");
           FlushModemInput (ModemFd);
           write (ModemFd, ATTENTION, strlen(ATTENTION));
           if (ReadStr (Answer) == NULL)
              {
               WriteLog ("Dial: no response to AT either, return ERR",
                         "", "", "");
               return (ERR); /* problem at our modem, not theirs */
              }
           else if (!EQUALS (Answer, NOANSWER))
              {
               WriteLog ("Dial: modem answered:", Answer, "", "");
               return (ERR);
              }
          }
```

```
    /* arrive here if ReadStr returned an Answer promptly */
    if (!EQUALS (Answer, ONE))
       {
        if (DebugLevel)
           WriteLog ("Dial: modem answered:", Answer, "", "");
        if (EQUALS (Answer, NOANSWER))
            WriteLog ("Dial: remote modem did not answer", "", "", "");
        else
           return (ERR);   /* may be a problem at our end - don't count  */
       }
    else   /* connection is established */
       return (TRUE);

   }

  if (i)  /* then the list was empty - i was never incremented */
   WriteLog ("Dial: none of the phone numbers worked", "", "", "");
  else
   WriteLog ("Dial: the phone number list was empty", "", "", "");

  return (FALSE);  /* none of the phone numbers worked */
```

```
#include "net.h"

int EndsWith (String, Target)
char *String;
char *Target;

{
   while ( *Target != '\0' )
       if (EQUALS(String, Target++))
          return (TRUE);

   return (FALSE);
}
```

```
#include "retcodes.h"

/*
   ErrMsg - return an error message to be printed to log file based
            on constant return codes.
*/

char *ErrMsg (code)

int code;  /* Code for which error message should be generated */

{
   char Temp[10];
   char *RetPtr;

   char *malloc ();

   switch (code)
      {
       case LOSTCONTACT   : return (MLOSTCONTACT);
       case BADCONNECTION : return (MBADCONNECTION);
       case BADREMOTENAME : return (MBADREMOTENAME);
       case FATAL         : return (MFATAL);
       case INTERNALERROR : return (MINTERNALERROR);
       case RECOVERABLE   : return (MRECOVERABLE);

       default            : sprintf (Temp, "%d", code);
                            RetPtr = malloc (strlen (Temp) + 1);
                            return (RetPtr);
      }
}
```

```
#include "net.h"
#include "iocontrol.e"

/*
   GetPrompt - read a string from the modem line and return a pointer to it.
               End of string will be indicated by a timeout on read operation.
               There may be embedded newlines or returns, but these will not
               mark the end of the string. If no characters are read, return
               a null string. Strip parity bit in case ISTRIP doesn't work.
*/

char *GetPrompt ()

{
   char *ptr;
   char TempBuf[80];
   int i=0;
   int c;

   while (i < 80)                          /* read until timeout or overflow */
       if ((c = Receive()) == ERR)         /* ERR means timeout, so exit */
           break;
       else if ((c&0x7f) != 0)
           TempBuf[i++] = (c & 0x7f);  /* buffer character, increment counter */

   TempBuf[i] = '\0';                      /* mark end of string */
   ptr = malloc (strlen(TempBuf)+1);   /* allocate storage */
   strcpy (ptr, TempBuf);              /* copy the string */

   return (ptr);                           /* return a pointer to the string */
}
```

```
#include "net.h"

extern int ModemFd;
extern int DebugLevel;

int HangUp (SysType)

char SysType;

/*
   Sends a hangup command to the modem
*/

{
   int     n,Done;
   char    FromModem[10];
   int     GoodHangup;

   FlushModemInput (ModemFd);

   sleep (3);
   write (ModemFd, ESCAPESTR, strlen(ESCAPESTR));
   sleep (2);

   write (ModemFd, "\r", 1);     /* Make sure modem is at beginning of line */

   sleep (1);

   FlushModemInput (ModemFd);

   write (ModemFd, HANGUPCMD, strlen(HANGUPCMD));

   GoodHangup = TRUE;

   if (ReadStr (FromModem) == NULL)
     {
      WriteLog ("HangUp: No response to hangup command", "", "", "");
      write (ModemFd, HANGUPCMD, strlen(HANGUPCMD));
      GoodHangup = FALSE;
     }
   else if (!EQUALS (FromModem, ZERO))
     {
      WriteLog ("HangUp: response to hangup command was", FromModem, "", "");
      write (ModemFd, HANGUPCMD, strlen(HANGUPCMD));
      GoodHangup = FALSE;
     }

   if (DebugLevel && GoodHangup)
      WriteLog ("HangUp: hung up the phone ok", "", "", "");

   write (ModemFd, RESETCMD, strlen(RESETCMD));  /* reset modem for uucp */

   return (TRUE);  /* hung up the phone OK */
}
```

```
#include "net.h"
#include "iocontrol.h"

#define WANTLOGIN 0
#define GOTLOGIN 1
#define WANTPASSWORD 2
#define GOTPASSWORD 3
#define CONNECTED 4

extern int ModemFd;
extern int DebugLevel;

int Login (Name, Password, SysType)

char *Name;
char *Password;
char  SysType;

{

char System[2];     /* print buffer for SysType */
char Command[40];   /* print buffer for talking to remote systems */
char *PromptStr;    /* prompt string from remote site: allocated by GetPrompt */
int Count = 0;      /* retry count for logging in to UNIX systems */
int State;          /* connection state for UNIX login automaton */

if (DebugLevel)
   WriteLog ("Login: Connected, about to log in as", Name, "", "");

switch (SysType)
  {

   case GCOS :      /* must change parity and deal with Honeywell */
      {
       struct termio TTYSet;
       int i;

       if (DebugLevel)
          WriteLog ("Login: system is GCOS", "", "", "");

       /* Reconfigure modem for even parity */

       if (ioctl (ModemFd, TCGETA, &TTYSet) == ERR) /* get old modem settings */
          {
           WriteLog ("Login:", "Can't get old modem settings", "",
                     "");
           return (ERR);
          }

       TTYSet.c_cflag |= PARENB;          /* Enable parity on Honeywell  */
       TTYSet.c_cflag &= ~PARODD;         /* Set parity to even          */
       TTYSet.c_cflag &= ~CSIZE;
       TTYSet.c_cflag |= CS7;             /* 7 data bits                 */

       if (ioctl (ModemFd, TCSETA, &TTYSet) == ERR)  /* set new attributes */
           {
            WriteLog ("Login: Can't set new modem attributes on", "",
                      "", "");
            return (ERR);
           }
```

```
    /* Done -- now continue login */

    write (ModemFd, CR, 1);           /* refresh login prompt */
    sleep (4);
    write (ModemFd, CR, 1);
    sleep (4);
    FlushModemInput (ModemFd);

    for (i = 0; i < strlen (Name); i++)
        Name[i] = toupper (Name[i]);

    for (i = 0; i < strlen (Password); i++)
        Password[i] = toupper (Password[i]);

    sprintf (Command, "%s %s\r", GCOSLOGIN, Name);
    write (ModemFd, Command, strlen (Command));  /* username */
    sleep (4);                        /* Wait for Password prompt */
    sprintf (Command, "%s\r", Password);
    write (ModemFd, Command, strlen (Command));  /* password */
    /* Assume good connection at this point */
    return (GOOD);
   }

case UNIX :      /* IOControl is the default shell */
   {
    if (DebugLevel)
       WriteLog ("Login: system is UNIX", "", "", "");

    sprintf (Command, "%s\r", Name);  /* build username response */
    FlushModemInput(ModemFd);
    write (ModemFd, CR, 1);           /* refresh login prompt */

    State = WANTLOGIN;  /* initial state */

    do {
        switch (State)
           {
            case WANTLOGIN:
                 PromptStr = GetPrompt();   /* get login prompt */
                 if (EndsWith(UNIXLOGIN, PromptStr))
                   State = GOTLOGIN;
                 else
                   {
                    ++Count;   /* increment retry count */
                    write (ModemFd, CR, 1);      /* refresh login prompt */
                    if (DebugLevel)
                       WriteLog("Login: wanted", UNIXLOGIN, "got",
                                PromptStr);
                   }
                 break;

            case GOTLOGIN:
                write (ModemFd, Command, strlen (Command));  /* send it */
                State = WANTPASSWORD;
                if (DebugLevel)
                   WriteLog ("Login:", "sent login", Command, "");
                break;

            case WANTPASSWORD:
                 PromptStr = GetPrompt();   /* get password prompt */
```

```
                    if (EndsWith(UNIXPASSWORD, PromptStr))
                       State = GOTPASSWORD;
                    else
                       {
                        ++Count;  /* increment retry count */
                        if (EndsWith(UNIXLOGIN, PromptStr))
                           State = GOTLOGIN;
                        if (DebugLevel)
                           WriteLog("Login: wanted", UNIXPASSWORD, "got",
                                    PromptStr);
                       }
                    break;

                case GOTPASSWORD:
                    sprintf (Command, "%s\r", Password); /* build response */
                    write (ModemFd, Command, strlen (Command));  /* send it */
                    if (DebugLevel)
                       WriteLog ("Login:", "sent password", Command, "");
                    State = CONNECTED;  /* we have sent our login sequence */
                    break;

               }
            free (PromptStr); /* release storage */
         } while ((State != CONNECTED) && (Count < MAXRETRY));

         FlushModemInput(ModemFd);

         if (State != CONNECTED)      /* could not log in to remote system */
            {
             WriteLog ("Login:", "could not log in to remote system", "", "");
             return (ERR);
            }

         return (GOOD);
        }
      default : /* undefined system type */
         {
          sprintf (System, "%c", SysType);
          WriteLog ("Login: undefined system type -", System, "", "");
          return (ERR);
         }
      }
}
```

```
#include "net.h"

extern int ModemFd;
struct termio TTYSet;

int OpenModem (PortName)

char *PortName;

/*
Open the named port for use by IOControl.  Make sure to set the line
parameters correctly: no buffering, 1200bps, ignore input parity, 7 bits out,
even parity out, no echo, etc. Returns ERR if the modem cannot be contacted
or ModemFd if all is well. Set global variable ModemFd.

The flag O_NDELAY must be set if the ioctl is to return instead of waiting
for the carrier (esp in XENIX). Hayes modem switches must be "uudduuud" for
use with this system(u=up, d=down). This allow use of modem for dialin and
dialout without special cables, jumpers, or switch manipulation.
*/

{
   register int OFlag = O_RDWR|O_NDELAY;

   if ((ModemFd = open (PortName, OFlag)) == ERR)
      {
       WriteLog ("OpenModem:", "Can't open", PortName, "");
       return (ERR);
      }

    if (ioctl (ModemFd, TCGETA, &TTYSet) == ERR) /* get old modem settings */
       {
        WriteLog ("OpenModem:", "Can't get old modem settings from", PortName,
                  "");
        return (ERR);
       }

    TTYSet.c_iflag &= ~INPCK;          /* Don't check input parity    */
    TTYSet.c_iflag |= ISTRIP;          /* Get rid of bit 8            */
    TTYSet.c_iflag &= ~ICRNL;          /* Don't convert CR to NL      */

    TTYSet.c_cflag &= ~PARENB;         /* No parity                   */
    TTYSet.c_cflag &= ~CSIZE;
    TTYSet.c_cflag |= CS8;             /* 8 data bits                 */

    TTYSet.c_cflag &= ~CBAUD;          /* Clear old baud-rate bits    */
    TTYSet.c_cflag |= B1200;           /* Set baud to 1200            */

    TTYSet.c_cflag |= CLOCAL;
    TTYSet.c_cflag |= HUPCL;

    TTYSet.c_lflag &= ~ICANON;         /* Don't want canonical input  */
    TTYSet.c_lflag &= ~ECHO;           /* No echo                     */

    TTYSet.c_oflag &= ~OPOST;          /* Don't post-process output   */

    TTYSet.c_cc [VMIN] = 1;            /* MIN = 1 char (no buffering) */
    TTYSet.c_cc [VTIME] = 0;           /* Expect data after 0 ms      */

    if (ioctl (ModemFd, TCSETA, &TTYSet) == ERR) /* set new modem attributes */
```

```
        {
         WriteLog ("OpenModem: Can't set new modem attributes on", PortName,
                   "", "");
         return (ERR);
        }

    /* normal UNIX systems use fcntl to reset O_NDELAY */
    /* this doesn't work for XENIX systems - see caller.c */
    if (fcntl(ModemFd, F_SETFL, (fcntl(ModemFd,F_GETFL,0) & ~O_NDELAY)) == ERR)
        {
         WriteLog ("OpenModem: Can't turn off O_NDELAY on", PortName, "", "");
         return (ERR);
        }

    return (ModemFd);
}
```

```
#include "net.h"

extern int MaxWait;  /* Maximum retry for hangup */

char *ReadStr (Str)

char *Str;

/*
   ReadStr() will read a string from the modem into the space pointed to
   by String. The input will be terminated by a CR or NL character. The
   calling routine must insure the presence of sufficient space at String.
   ReadStr() will return a pointer to String if it succeeds or the value
   NULL if it fails.
*/

{
   register int n;
   register char *Ptr = NULL;
   char *GetStr();

   for (n=0; ((n < MaxWait) && (Ptr == NULL)); n++)
      Ptr = GetStr(Str);

   return (Ptr);
}




char *GetStr (String)

char *String;

{
   char *Ptr;

   Ptr = String-1;                              /* point to start of storage */
   do {
       Ptr++;
       if ((*Ptr = Receive()) == ERR)           /* get character from modem */
             return (NULL);                     /* bad return - timed out   */
      } while ((*Ptr != CRET) && (*Ptr != NL));

   *Ptr = '\0';    /* replace newline or cret with null */

   return (String);  /* good return - pointer to string */
}
```

```
#include "net.h"

extern int ModemFd;
extern int DebugLevel;

int SetHayes (PortName)
char *PortName;

/*
   Set the command modes and switches on the Smartmodem 1200.
   Test the modem to see if it will accept commands and give proper response.
   Return GOOD if all's well, ERR otherwise.
*/

{
   register int i;
   int c;
   char str[80];

    FlushModemInput (ModemFd);  /* Be sure buffer is empty */

    if (write (ModemFd, SETUP, strlen(SETUP)) == 0)
       {
        WriteLog ("SetHayes: can't write setup string to", PortName, "", "");
        return (ERR);
       }
    FlushModemInput (ModemFd);  /* Be sure buffer is empty */

    if (DebugLevel)
       WriteLog ("SetHayes: wrote SETUP string to", PortName, "", "");

    /* try to talk to the modem */
    for (i=0; i<10; i++)
       {
        write (ModemFd, ATTENTION, strlen(ATTENTION));
        switch (c = Receive ())
           {
            case ERR : WriteLog ("SetHayes: no response to ATTENTION signal",
                                 "", "", "");
                       break;

            case OK : FlushModemInput (ModemFd);
                      if (DebugLevel)
                         WriteLog ("SetHayes: modem responded to ATTENTION",
                                   "signal", "", "");
                      return (GOOD);

            default : sprintf(str, "'%c'", c);
                      WriteLog ("SetHayes: modem answered ATTENTION with",
                                str, "", "");
                      break;
           }
       }

    FlushModemInput (ModemFd);  /* Be sure buffer is empty */
    WriteLog ("SetHayes: Cannot establish rapport with modem on",
              PortName, "", "");
    return (ERR);

}
```

IOCONTROL

This section contains the functions used only by the IOControl program (IOCONTROL).

```
iocontrol : iocontrol.o checksum.o \
 getblock.o getfile.o getpacket.o preemption.o \
 receive.o send.o sendblock.o sendbyte.o sendenq.o \
 sendname.o sendfile.o sendpacket.o waitack.o \
 archive.o waitname.o filenq.o movmem.o \
 waitenq.o dequeue.o writelog.o datetime.o \
 validsite.o myname.o getdir.o lockfile.o readsite.o \
 fileopen.o setport.o remove.o createfile.o \
 sendheader.o getheader.o touchsite.o putsite.o abort.o \
 exit.o
        cc -O -o iocontrol iocontrol.o checksum.o \
 getblock.o getfile.o getpacket.o preemption.o \
 receive.o send.o sendblock.o sendbyte.o sendenq.o \
 sendfile.o sendpacket.o waitack.o \
 archive.o sendname.o waitname.o filenq.o movmem.o \
 waitenq.o dequeue.o writelog.o datetime.o \
 validsite.o myname.o getdir.o lockfile.o readsite.o \
 fileopen.o setport.o remove.o createfile.o \
 sendheader.o getheader.o touchsite.o putsite.o abort.o \
 exit.o
        strip iocontrol

iocontrolh : iocontrolh.o checksum.o \
 getblockh.o getfileh.o getpacket.o preemption.o \
 receiveh.o sendh.o sendblockh.o sendbyteh.o sendenq.o \
 sendfileh.o sendpacket.o waitack.o \
 archive.o sendname.o waitname.o filenq.o movmem.o \
 waitenq.o dequeue.o writelog.o datetime.o \
 validsite.o myname.o getdir.o lockfile.o readsite.o \
 fileopen.o setporth.o remove.o createfile.o \
 sendheader.o getheader.o touchsite.o putsite.o \
 waitbeep.o abort.o exit.o
        cc -O -o iocontrolh iocontrolh.o checksum.o \
 getblockh.o getfileh.o getpacket.o preemption.o \
 receiveh.o sendh.o sendblockh.o sendbyteh.o sendenq.o \
 sendfileh.o sendpacket.o waitack.o \
 archive.o sendname.o waitname.o filenq.o movmem.o \
 waitenq.o dequeue.o writelog.o datetime.o \
 validsite.o myname.o getdir.o lockfile.o readsite.o \
 fileopen.o setporth.o remove.o createfile.o \
 sendheader.o getheader.o touchsite.o putsite.o \
 waitbeep.o abort.o exit.o
        strip iocontrolh

iocontrol.o : iocontrol.c iocontrol.h
        cc -O -c iocontrol.c

iocontrolh.o : iocontrol.c iocontrol.h
        mv iocontrol.o ,iocontrol.o
        cc -O -c -DFORHONEY iocontrol.c
        cp iocontrol.o iocontrolh.o
        mv ,iocontrol.o iocontrol.o

checksum.o : checksum.c
        cc -O -c checksum.c

getblock.o : getblock.c iocontrol.h
        cc -O -c getblock.c

getblockh.o : getblock.c iocontrol.h
```

```
	mv getblock.o ,getblock.o
	cc -O -c -DFORHONEY getblock.c
	cp getblock.o getblockh.o
	mv ,getblock.o getblock.o

getfileh.o : getfile.c iocontrol.h
	mv getfile.o ,getfile.o
	cc -O -c -DFORHONEY getfile.c
	cp getfile.o getfileh.o
	mv ,getfile.o getfile.o

getpacket.o : getpacket.c iocontrol.h
	cc -O -c getpacket.c

preemption.o : preemption.c iocontrol.h
	cc -O -c preemption.c

send.o : send.c iocontrol.h
	cc -O -c send.c

sendh.o : send.c iocontrol.h
	mv send.o ,send.o
	cc -O -c -DFORHONEY send.c
	cp send.o sendh.o
	mv ,send.o send.o

sendblock.o : sendblock.c iocontrol.h
	cc -O -c sendblock.c

sendblockh.o : sendblock.c iocontrol.h
	mv sendblock.o ,sendblock.o
	cc -O -c -DFORHONEY sendblock.c
	cp sendblock.o sendblockh.o
	mv ,sendblock.o sendblock.o

sendbyte.o : sendbyte.c iocontrol.h
	cc -O -c sendbyte.c

sendbyteh.o : sendbyte.c iocontrol.h
	mv sendbyte.o ,sendbyte.o
	cc -O -c -DFORHONEY sendbyte.c
	cp sendbyte.o sendbyteh.o
	mv ,sendbyte.o sendbyte.o

sendenq.o : sendenq.c iocontrol.h
	cc -O -c sendenq.c

sendname.o : sendname.c iocontrol.h
	cc -O -c sendname.c

sendfileh.o : sendfile.c iocontrol.h
	mv sendfile.o ,sendfile.o
	cc -O -c -DFORHONEY sendfile.c
	cp sendfile.o sendfileh.o
	mv ,sendfile.o sendfile.o

sendpacket.o : sendpacket.c iocontrol.h
	cc -O -c sendpacket.c

waitack.o : waitack.c iocontrol.h
	cc -O -c waitack.c
```

```
waitname.o : waitname.c iocontrol.h
	cc -0 -c waitname.c

movmem.o : movmem.c
	cc -0 -c movmem.c

waitenq.o : waitenq.c iocontrol.h
	cc -0 -c waitenq.c

setporth.o: setport.c
	mv setport.o ,setport.o
	cc -c -0 -DFORHONEY setport.c
	cp setport.o setporth.o
	mv ,setport.o setport.o

setport.o: setport.c
	cc -c -0 setport.c

createfile.o: createfile.c
	cc -0 -c createfile.c

sendheader.o: sendheader.c
	cc -0 -c sendheader.c

getheader.o: getheader.c
	cc -0 -c getheader.c

touchsite.o: touchsite.c
	cc -0 -c touchsite.c

waitbeep.o: waitbeep.c
	cc -0 -c waitbeep.c

exit.o: exit.c
	cc -0 -c exit.c

iocontroldb: iocontroldb.c
	cc -0 -o iocontroldb iocontroldb.c

iocontrolhdb: iocontrolhdb.c
	cc -0 -o iocontrolhdb iocontrolhdb.c
```

```
#include <stdio.h>
/*
  CheckSum - compute a 16-bit checksum of 'length' bytes starting at 'data'.
*/

int CheckSum (data, max)

unsigned char *data;
int  max;

{
   int i;
   long Result; /* Storage for temporary result */

   Result = 0;

  for (i = 0; i <= max; i++)
      {
#ifdef FORHONEY
        if (data[i] == (unsigned char) '\\')
           {
            if (data[i+1] == (unsigned char) '\\')
               {
                Result += (int) data[i];
                i++;  /* Skip second backslash in case of double-backslash */
               }
            else
               ;  /* Do not checksum a stand-alone backslash */
           }
        else
#endif
           Result += (int) data[i];
      }

   Result &= 0xffff; /* Restrict to 16-bit quantity */

   return ((int) Result);
}
```

```
#include "net.h"

/*
   CreateFile - build a file name in the directory given by Queue, with
                a '.' in front of the name to make it invisible, using
                the name of the file in FileName. Open the file and
                return a file descriptor.
*/

FILE *CreateFile (FileName, Queue)

char *FileName;      /* pointer to new file name */
char *Queue;         /* directory in which to put the file */

{
   pathname PathName;      /* pointer to the full path name of the file */

   sprintf (PathName, "%s/.%s", Queue, FileName);  /* Create full pathname */

   return (fopen(PathName, "w"));
}
```

```
#include "net.h"
#include "iocontrol.h"

extern pathname QueueName;  /* Name of current queue */

/*
   Exit - process exit condition for IOControl by unlocking the current
          queue and calling the system exit routine (exit(2)) with the
          given exit status code.
*/

void Exit (code)

int code;  /* Exit return code */

{
   UnLock (QueueName);  /* Try to remove the lock file associated with queue */

   exit (code);
}
```

```
#include "net.h"
#include "iocontrol.h"
#include "iocontrol.e"

extern pathname QueueName;  /* Reference semi-global queue name */
extern int CheckBetween;
extern int MaxRetry;

/*
   GetBlock - read the next block from remote system, taking
              care of retransmission, etc.  Either return a pointer
              to good packet data along with a length, or return an error
              code (NULL) with the length set to indicate reason
              for leaving (EOT = end of transmission received,
              ERR = error). Return parameters "length" and "endflag"
              indicate the length of data (in bytes) and the
              value of the end-of-text flag, respectively.
*/

#define ENDFLAG (PackLen - ETXOFFSET)  /* End-of-text pointer is here */

unsigned char *GetBlock (length, endflag)

int *length;  /* Return parameter for length of block */
int *endflag; /* Return parameter for end-of-text vs. end-text-block */

{
   int TempValue;     /* Result to return to caller */
   unsigned char *TempBuf;     /* Pointer to buffer returned from 'getpacket' */
   unsigned char *BlockBuffer; /* Buffer to return to caller with good data */
   int PackLen;       /* Length of incoming packet */
   int HisCheckSum;   /* Checksum sent by remote system */
   int MyCheckSum;    /* Checksum computed here */
   int HisSeqNo;      /* Sequence number decoded from received packet */
   int RetryCount;    /* Retry count before giving up on a packet */
   unsigned char c;
   char HisStr[9];
   char MyStr[9];

StartOver:
   RetryCount = 0;
   *length = ERR;  /* Default return parameter */
   while (RetryCount <= MaxRetry)
      {
       TempBuf = GetPacket (&PackLen);

       if (CheckBetween)
          {
           if (Preemption (QueueName))
              {
               SendByte (EOT);     /* Tell remote we couldn't stay */
               *length = MYEOT;    /* I sent an EOT */
               if (TempBuf != NULL)
                  free (TempBuf);
               return (NULL);
              }

           if (MasterMode && Abort ("."))
              {
               SendByte (CAN);     /* Tell remote we were terminated */
```

```
                *length = MYCAN;     /* Sent a cancel (must delete temp file) */
                if (TempBuf != NULL)
                   free (TempBuf);
                return (NULL);       /* Exit gracefully */
               }
            }

        if (TempBuf == NULL)  /* We have an error */
            {
             sleep (2);       /* Wait 2 seconds - attempted error recovery */
             FlushModemInput (ModemFd);
             RetryCount++;    /* Consider TIMEOUT to be very bad */
             WriteLog("GetBlock: timed out on GetPacket", "", "", "");
#ifndef FORHONEY
             SendByte (ACK); /* try to recover & get in sync */
#endif
             continue;        /* Go back to main loop and try again */
            }

        if (PackLen == 3)
            {
             if (TempBuf [0] == EOT)  /* Empty transmission - switch modes */
                {
                 *length = EOT;  /* Set error condition to "EOT received" */
                 free (TempBuf);
                 return (NULL);  /* Return error condition */
                }
             else if (TempBuf [0] == CAN)  /* Remote had to abort cleanly */
                {
                 *length = CAN;  /* Set error condition to "CAN received" */
                 free (TempBuf);
                 return (NULL);  /* Return error condition */
                }
             else /* Phase error, etc. */
                {
                 FlushModemInput (ModemFd);
                 RetryCount++;    /* Note the failure */
                 SendByte (NAK);  /* Improper control information */
                 WriteLog("GetBlock: improper control info", "-", "sent NAK", "");
                 free (TempBuf);
                }
            }

        else if (PackLen < 3)
            {
             sleep (2);  /* wait while things stabilize - this is adjustable */
             FlushModemInput (ModemFd); /* Phase error requires good cleanup */
             RetryCount++;
             SendByte (NAK);   /* Send negative acknowledgement */
             free (TempBuf);
             WriteLog("GetBlock: phase error - sent NAK", "", "", "");
            }

        else if (PackLen > 3)
            {
             c = TempBuf [3];  /* Save */
             TempBuf [3] = '\0';
             sscanf (TempBuf, "%x", &HisSeqNo);  /* Decode packet number */
             TempBuf [3] = c;  /* Restore */

             if ((HisSeqNo != SeqNo) && (HisSeqNo != SeqNo-1))
```

```
            {
             RetryCount++;     /* Bad data - don't go forever */
             SendByte (NAK);   /* Send negative acknowledgement */
             WriteLog ("GetBlock: Sequence number is bad","-","sent NAK","");
             sprintf (HisStr, "%d", HisSeqNo);
             sprintf (MyStr, "%d", SeqNo);
             WriteLog ("GetBlock: his number is", HisStr, "mine is", MyStr);
             continue;         /* Get back to main loop */
            }

         c = TempBuf [PackLen-2];  /* Save */
         TempBuf [PackLen-2] = '\0'; /* Set up EOS for a second */
         sscanf (&(TempBuf[PackLen - 6]), "%x", &HisCheckSum);
         TempBuf [PackLen-2] = c;  /* Restore EOS - don't worry */
         MyCheckSum = CheckSum (TempBuf+3, PackLen-10); /* Compute CkSum */

         if (MyCheckSum != HisCheckSum)  /* Packet in error */
            {
             FlushModemInput (ModemFd);  /* Make sure buffer is clean */
             SendByte (NAK);  /* Send negative-acknowledgement control */
             WriteLog ("GetBlock: checksum's bad", "-", "sent NAK", "");
             sprintf (HisStr, "%x", HisCheckSum);
             sprintf (MyStr, "%x", MyCheckSum);
             WriteLog ("GetBlock: his checksum is", HisStr, "mine is", MyStr);
            }
         else  /* All is A.O.K. */
            {
             SendByte (ACK);  /* Send positive acknowledgement */
             break;  /* Leave this loop, we have a good packet */
            }
        }

     RetryCount++;  /* See how long it takes */
    }

/* Why did we leave the loop? */

if (RetryCount > MaxRetry)  /* Because of error */
   return (NULL);  /* We had an error */
else
   {
    if (SeqNo == HisSeqNo)
       SeqNo = (SeqNo + 1) % 0x100;   /* Set up for next packet if good */
    else
       goto StartOver; /* Start over if old packet retransmitted */

    /* Set up return information - create block buffer */

    *endflag = TempBuf [ENDFLAG];  /* Set return parameter */

    *length = ENDFLAG - 4;  /* Set block length for caller (no EOS) */
    BlockBuffer = (unsigned char *) malloc (*length + 1);/* Allocate block */
    if (BlockBuffer == NULL)  /* Don't keep going if problem exists */
       {
        WriteLog ("GetBlock:", "OUT OF MEMORY", "allocating block", "");
        Exit (INTERNALERROR);
       }

    movmem (TempBuf + 3, BlockBuffer, *length);  /* Move good data */

    /* Get rid of storage allocated in lower level for packet */
```

```
        free (TempBuf);

        return (BlockBuffer);    /* Return pointer to good block of data */
       }
}
```

```
#include "net.h"
#include "iocontrol.h"
#include "iocontrol.e"

/*
   GetFile - read the next file (if available) from remote system, taking
             care of retransmission, etc.  Either write a new file or
             return an error value.
*/

int GetFile ()

{
   int BlockLen;            /* Length of each new block read */
   int ETXFlag;             /* Indicates EOT received */
   int Result;              /* Result from 'GetHeader' call */
   unsigned char *TempBuf; /* Block pointer for each read-in block */
   int MsgLength = 0;       /* Length of message (index to Message) */
   FILE *FileFp;            /* File pointer for temporary file */
   filename FileName;       /* Name of current incoming message */
   filename TmpFileName;
   int i;

   /* For each file received, get header with message name */

   Result = GetHeader (FileName); /* Get header from remote system & audit */

   if (Result == EOT)
      return (EOT);           /* No new header arrived; end of transmission */

   if (Result == REJECT)      /* The incoming message has been rejected */
      return (NULL);          /* Exit gracefully, as if nothing happened */

   /* Create and receive new message */

#ifdef FORHONEY
   FileName[0] = toupper(FileName[0]);
#endif

   FileFp = CreateFile (FileName, MSGPROCQ);

   if (FileFp == NULL)
      {
       WriteLog ("GetFile:", "can't open file for message:", FileName, "");
       Exit (RECOVERABLE);
      }

   while ((TempBuf = GetBlock (&BlockLen, &ETXFlag)) != NULL) /* Loop */
      {
       for (i = 0; i < BlockLen; i++)
           {
            if (TempBuf [i] == DLE)
               {
                i++;                        /* Skip the first DLE of any pair */
                if (i >= BlockLen)
                   {
                    WriteLog ("GetFile:", "INTERNAL ERROR:", "unmatched DLE",
                              "at end of packet data");
                    Exit (INTERNALERROR);
                   }
```

```
            }

#ifdef FORHONEY
            if (TempBuf [i] == '\r')
               TempBuf [i] = '\n';  /* Convert CR to NL, receiving from Honey */
#endif

            fputc (TempBuf [i], FileFp); /* Write the next character */
           }

       MsgLength += BlockLen;  /* Keep track of length of entire message */

       if (ETXFlag == ETX)
            break;  /* Leave the loop if end-of-text received */

       free (TempBuf);  /* Get rid of storage allocated at top of loop */
       fflush (FileFp);
      }

   fclose (FileFp);  /* Write the complete file */

   /* Find out why we left while loop */

   if (TempBuf != NULL)
      {
       if (ETXFlag == ETX)
          {
           free (TempBuf);  /* Let it go */
           WriteLog ("GetFile:", "file", FileName, "received OK");
           FileNQ (FileName, MSGPROCQ);  /* Give the file to msg-proc */
           return (NULL);
          }
      }
   else if (BlockLen == EOT)  /* An end-of-transmission was received */
      {
       WriteLog ("GetFile:", "EOT received", "but not expected", "");
       WriteLog ("GetFile:", FileName, "not saved", "");
       sprintf (TmpFileName, ".%s", FileName);
       Remove (TmpFileName, MSGPROCQ); /* Get rid of it */
       return (INTERRUPTED);  /* Indicate we were interrupted */
      }
   else if (BlockLen == CAN)  /* The remote system shut down suddenly */
      {
       WriteLog ("GetFile:", "CAN received", "(remote aborted)", "");
       WriteLog ("GetFile:", FileName, "not saved", "");
       sprintf (TmpFileName, ".%s", FileName);
       Remove (TmpFileName, MSGPROCQ); /* Get rid of it */
       return (ABORTED);       /* Indicate remote system aborted */
      }
   else if (BlockLen == MYEOT)  /* We sent an EOT because of preemption */
      {
       WriteLog ("GetFile:", "INTERRUPTED while receiving", FileName, "");
       WriteLog ("GetFile:", FileName, "not saved", "");
       sprintf (TmpFileName, ".%s", FileName);
       Remove (TmpFileName, MSGPROCQ);  /* Get rid of it */
       Exit (INTERRUPTED);
      }
   else if (BlockLen == MYCAN)  /* We sent a CAN because of abort */
      {
       WriteLog ("GetFile:", "ABORTED while receiving", FileName, "");
       WriteLog ("GetFile:", FileName, "not saved", "");
```

```
        sprintf (TmpFileName, ".%s", FileName);
        Remove (TmpFileName, MSGPROCQ);  /* Get rid of it */
        ShutDown ();  /* Shut this thing down */
       }
    else  /* Process errors */
       {
        WriteLog ("GetFile:", "timed out -", "FAILED", "");
        sprintf (TmpFileName, ".%s", FileName);
        Remove (TmpFileName, MSGPROCQ);  /* Throw away incomplete file */
        return (ERR);
       }
}
```

```
#include "net.h"
#include "iocontrol.h"
#include "iocontrol.e"

extern pathname QueueName;  /* Reference semi-global queue name */

/*
   GetHeader - read header for next incoming message.  The message header
               is merely a packet containing the name of the message file.
               Verify that the message is 'new' and transmit ACK if so,
               NAK if message rejected for being a 'repeat'.
               Return EOT if remote has no more files to send.
*/

int GetHeader (messagename)

char *messagename;

{
   unsigned char *ResponseBlock; /* Block from 'getblock', allocated therein */
   int  EndFlag;          /* End of text flag - needed for 'getblock' */
   int  BlockLength;      /* Length of block (message-name) returned */

   if ((ResponseBlock = GetBlock (&BlockLength, &EndFlag)) == NULL)
      if (BlockLength != EOT)
         {
          /* Unexpected condition - no error-handling here */
          WriteLog ("GetHeader:", "message header received improperly", "", "");
          Exit (BADCONNECTION);  /* Synchronization bad between messages */
         }
      else
         return (EOT);     /* Remote had no more files to transmit */

   if (BlockLength > FILENAMELEN)  /* Don't permit overindexing */
      {
       /* Remote message name is not of proper form */
       WriteLog ("GetHeader:", "message name is invalid", "", "");
       free (ResponseBlock);
       Exit (BADCONNECTION);
      }

   /* Check message name against records -- not implemented now */

   strncpy (messagename, ResponseBlock, BlockLength);  /* set return text */
   messagename [BlockLength] = '\0';

   /* Check for need to abort (either from preemption or shutdown) */

   if (Preemption (QueueName))
      {
       WriteLog ("GetHeader:", "INTERRUPTED while receiving", messagename, "");
       SendByte (EOT);  /* Remote will understand */
       Exit (INTERRUPTED);
      }

   if (MasterMode && Abort ("."))
      {
       WriteLog ("GetHeader:", "ABORTED while receiving", messagename, "");
       SendByte (CAN);  /* Remote will know we are aborting */
       ShutDown ();     /* Don't do anything else */
```

```
    }
  SendByte (ACK);  /* Transmit file acceptance ACKnowledge */
  FlushModemInput (ModemFd);
  free (ResponseBlock); /* release storage */
  return (NULL);   /* Good result */
}
```

```
#include "net.h"
#include "iocontrol.h"
#include "iocontrol.e"

/*
   GetPacket - retrieve a packet (control information or data) from the
               remote system, allocating storage for it, and return a
               pointer to the caller, who will release storage when done.
*/

unsigned char *GetPacket (length)

int *length;  /* Return parameter is length of packet in bytes */

{
   unsigned char TempPacket [MAXPACKET];  /* Allocate static storage for now */
   unsigned char *Ptr;       /* Pointer for allocated storage when ready */
   unsigned char c;          /* Character received from modem */
   int RcvByte;
   int Remember = FALSE;     /* Flag to say if last character was "DLE" */
#ifdef DEBUG
   FILE *tmp=fopen ("/usr/taccnet/log/proto.log", "a");
#endif

#ifdef DEBUG
   fprintf(tmp, "Response:\t\n-----------------------------------------------\n\t");
   fflush(tmp);
#endif

   *length = 0;
   while ((RcvByte = Receive ()) != ERR)
      {
       c = (unsigned char) (RcvByte & 0xff);
#ifdef DEBUG
   fprintf(tmp," %.2x", c);
   fflush(tmp);
#endif

       if ((Remember) && (c == EM))  /* This is the end of the message */
          {
           TempPacket [*length] = c;
           Receive ();               /* Read the CR */
           break;                    /* Get out of this infernal read loop */
          }

       if ((*length) + 1 >= MAXPACKET)  /* There will be no room for EM */
          {
           WriteLog ("GetPacket:", "packet received is too big for buffer;",
                     "discarding", "");
           sleep (2);    /* Make sure no more garbage comes */
           FlushModemInput (ModemFd);  /* Get rid of old */
           SendByte (NAK);
           *length = 0;
           continue;     /* Skip to the next iteration */
          }

       TempPacket [(*length)++] = c; /* Put this character in the block */

       Remember = ((c == DLE) && (!Remember)); /* set or reset as needed */
      }
```

```
#ifdef DEBUG
   fprintf(tmp, "\t\n-----------------------------------------\n");
   fflush(tmp);
#endif

   if (RcvByte == ERR)  /* This was an error - return the NULL pointer */
      return (NULL);

   (*length)++;                          /* Offset length to start at 1 */

   Ptr = (unsigned char *) malloc (*length + 1);/* Allocate necessary storage */
   if (Ptr == NULL)
       {
        WriteLog ("GetPacket:", "OUT OF MEMORY", "allocating for packet", "");
        Exit (INTERNALERROR);
       }

   movmem (TempPacket, Ptr, *length);  /* Move data to new buffer */

#ifdef DEBUG
   fclose (tmp);
#endif
   return (Ptr);                         /* Back to caller with good pointer */
}
```

```
#include "net.h"
#include "iocontrol.h"
#include <signal.h>

/*
   IOControl - perform both SEND and RECEIVE functions for bottom layer of
               network system.  Use character-oriented protocol with check-
               sums and stop-and-wait retransmission scheme. (With a
               single-packet transmission window.)  If invoked
               by local system, assume role of SENDER at start,
               otherwise become a RECEIVER for the first transactions.
*/

/* Global variables for I/O Control system */

int ModemFd;            /* Modem file descriptor (for read(2)/write(2)) */
int SeqNo = 0;          /* Current packet-sequence number */
pathname LogFile;       /* Global LogFile for IOControl routines */
pathname QueueName;     /* Name of queue from which we're reading */
int DebugLevel = 0;     /* Runtime debug level (0 = normal) */
int MasterMode;         /* Flag indicating if IOCONTROL was loaded as master */
int CheckBetween = 0;   /* Preemption flag indicating to check 'tween packets */
int TimeOut = 10;       /* Timeout (in seconds) for reading a character */
int HoneyTime = 30;     /* Timeout for Honeywell versions */
int BlockLength = BLOCKLENGTH;  /* Size of a block of text in a packet */
int HoneyBlock = BLOCKLENGTH;   /* Size of a GCOS block -- cannot exceed ~64 */
int MaxRetry = MAXRETRY;        /* Maximum number of retransmissions, etc. */
int Archiving = TRUE;  /* Set automatic archiving of all messages */

/* */

main (argc, argv)

int   argc;
char *argv[];

{
   int mode;          /* Current operating mode (0,1=init; 2=send; 3=receive) */
   int memory = FALSE;      /* Used to determine hangup status */
   pathname NextPath;       /* Complete path of any given file */
   char *NextName;          /* File name for each file read from queue */
   sitename RemoteName;     /* Name of calling remote site */
   int Result;
   FILE *ParamFileFp;       /* Parameter file pointer */
   char Key[20];
   int  Value;
   int  FirstTime = TRUE;  /* Set until first file is successfully sent */

   void ShutDown ();        /* Shutdown routine */

   /* Parse arguments to see if this is invoked on local system */

   umask (UMASK);

   sprintf (LogFile, "log/%s.log", IOCONTROL);

   /* validate and parse arguments */
   if ((argc < 1) || (argc > 4))
         usage (argv[0]);
```

```
    --argc;
    if ((argv[argc][0] == '-') && (argv[argc][1] == 'd'))
        sscanf (argv[argc]+2, "%d", &DebugLevel);

    if (argc >= 2)      /* master mode */
       {
        mode = MASTERINIT;              /* Enter INITIALIZATION as MASTER */
        strcpy (QueueName, argv[1]);    /* Place queue name in local var */
        strcpy (RemoteName, argv[1]);   /* Remember name of remote */
        ModemFd = SetPort (argv [2]);   /* Open modem as instructed */
        if (ModemFd == NULL)  /* Could not do it */
           {
            WriteLog ("IOControl:", "FAILED opening", argv[2], "(modem)");
            Exit (FATAL);  /* The modem should have been available; abort */
           }

        if ((ParamFileFp = fopen (PARAMFILE, "r")) != NULL)
           {
            fscanf (ParamFileFp, "%s %d\n", Key, &Value);
            while ((!feof (ParamFileFp)) && (!ferror (ParamFileFp)))
               {
                if (EQUALS (Key, "preemption"))
                   CheckBetween = Value;
                else if (EQUALS (Key, "timeout"))
                   TimeOut = Value;
                else if (EQUALS (Key, "timeout(gcos)"))
                   HoneyTime = Value;
                else if (EQUALS (Key, "blocklength"))
                   BlockLength = Value;
                else if (EQUALS (Key, "blocklength(gcos)"))
                   HoneyBlock = Value;
                else if (EQUALS (Key, "maxretransmit"))
                   MaxRetry = Value;
                else if (EQUALS (Key, "archiving"))
                   Archiving = Value;

                fscanf (ParamFileFp, "%s %d\n", Key, &Value);
               }

            fclose (ParamFileFp);
           }

        MasterMode = TRUE;              /* This is the master */
       }
    else                /* slave mode */
       {
        ModemFd = SetPort ("/dev/tty"); /* Treat terminal as remote */
        mode = SLAVEINIT;               /* Enter INITIALIZATION as SLAVE */
        MasterMode = FALSE;             /* This is the slave */
       }

    signal (SIGTERM, ShutDown);  /* Set interrupt handler routine */


    /* Main loop - perform tasks depending on current mode */

    while (mode != HANGUP)
       {
        switch (mode) {
           case MASTERINIT :
                    sprintf (LogFile, "log/%s.log", RemoteName);
```

```
                if (DebugLevel)
                   WriteLog ("IOControl: assuming MASTER mode.", "", "", "");
                WaitEnq ();             /* Wait for enquire, send ACK */
                if (DebugLevel)
                   WriteLog ("IOControl:", "enquire received and",
                             "acknowledged", "");
                SendName ();            /* Send system name, wait for ACK */
                WriteLog ("IOControl:", "(MASTER) connection established (",
                          RemoteName, ")");
                mode = SENDMODE;        /* Enter send mode */
                break;

        case SLAVEINIT :
                if (DebugLevel)
                   WriteLog ("IOControl: assuming SLAVE mode.", "", "", "");
#ifdef FORHONEY
                sleep (15);
#endif
                SendEnq ();              /* Send ENQ signal, wait for ACK */

                if (DebugLevel)
                   WriteLog ("IOControl:", "enquire sent and",
                             "acknowledged", "");

                WaitName (RemoteName); /* Wait for remote id, send ACK */

                if (DebugLevel)
                   WriteLog ("IOControl:", "ID packet received from",
                             RemoteName, "");

                sprintf (LogFile, "log/%s.log", RemoteName);
                WriteLog ("IOControl:", "(SLAVE) connection established (",
                          RemoteName, ")");

                TouchSite (RemoteName);  /* note that site called in OK */
                sprintf (QueueName, "%s/%s", MASTERQ, RemoteName);

                if (Lock (QueueName) == ERR) /* Try to lock current queue */
                   {
                    WriteLog ("IOControl:", "cannot lock", QueueName,
                              "- goodbye!");
                    exit (FATAL);
                   }

                mode = RECEIVEMODE;       /* Enter receive mode */
                break;

        case SENDMODE : /* Send mode */
                /* Check to see if line needed by higher priority task */
                /* ...or if we're simply finished.  Hang up if so.     */

                if ((memory) || Preemption (QueueName))
                   {
                    mode = HANGUP;
#ifdef FORHONEY
                    sleep (3);
#endif
                    SendByte (EOT);      /* Send signal to remote */
                    break;
                   }
```

```
                    NextName = DeQueue (QueueName); /* Read sysX queue */
                    if (NextName != NULL)
                        {
                         sprintf (NextPath, "%s/%s", QueueName, NextName);
                         if (SendFile (NextPath, NextName) != ERR)  /* Send it! */
                            {
                             Remove (NextName, QueueName);   /* Delete file */
                             if (FirstTime)
                                {
                                 TouchSite (QueueName);      /* Site must be up */
                                 FirstTime = FALSE;
                                }
                            }
                         else  /* Something went wrong during transmission */
                            {
                             WriteLog ("IOControl:", QueueName, "FAILED", "");
                             Exit (LOSTCONTACT);  /* Hang up and all that */
                            }
                        }
                    else
                        {
#ifdef FORHONEY
                         sleep (3);
#endif
                         SendByte (EOT);  /* Send an end-of-transmission */
                         memory = TRUE;        /* Remember that we sent EOT */
                         mode = RECEIVEMODE;  /* Enter receive mode */
                        }
                    break;

            case RECEIVEMODE : /* Receive mode */
                    Result = GetFile (); /* Get next receive file */
                    if (Result == EOT)
                         {
                          mode = SENDMODE;
                         }
                    else if ((Result == INTERRUPTED) || (Result == ABORTED))
                         {
                          WriteLog ("IOControl:", RemoteName, "aborted", "");
                          WriteLog ("IOControl:", "relinquishing line", "", "");
                          mode = HANGUP;  /* Force a graceful hangup */
                         }
                    else if (Result == ERR)
                         {
                          WriteLog ("IOControl:", RemoteName, "FAILED", "");
                          Exit (LOSTCONTACT);
                         }
                    else
                         memory = FALSE;       /* Forget about hanging up */
                    break;

            default : /* Illegal (undefined) mode */
                    WriteLog ("IOControl:", "internal error:", "bad mode", "");
                    Exit (INTERNALERROR); /* Ungraceful exit to O/S */
            }
        }  /* End of WHILE */

    WriteLog ("IOControl:", "conversation COMPLETE (", RemoteName, ")");

    if (Preemption (QueueName))  /* Give different return code if preempted */
        Exit (INTERRUPTED);
```

```
   Exit (GOOD);  /* Exit with 'normal completion' code, 0 */
}



int usage (name)
char *name;
{
   fprintf (stderr, "usage: %s [ sysname portname ][-dn]\n", name);
   exit (FATAL);
}



void ShutDown ()

{
   WriteLog ("ShutDown:", "IOControl aborted by operator request", "", "");
   Exit (ABORTED);
}
```

```
#include "sysdef.h"
main()
{
#ifdef ONYX
execl ("/v/airmics/bin/iocontrol", "/v/airmics/bin/iocontrol", "-dl", 0);
#else
execl ("/usr/taccnet/bin/iocontrol", "/usr/taccnet/bin/iocontrol", "-dl", 0);
#endif
}
```

```
#include "sysdef.h"
main()
{
#ifdef ONYX
execl ("/v/airmics/bin/iocontrolh", "/v/airmics/bin/iocontrolh", "-dl", 0);
#else
execl ("/usr/taccnet/bin/iocontrolh", "/usr/taccnet/bin/iocontrolh", "-dl", 0);
#endif
}
```

```
#include <stdio.h>
/*
   movmem - copy memory contents from one location to another.  Locations
            are passed in as string pointers indicating 'from' and 'to'
            addresses, respectively. Simply copy bytes one-by-one
            until all bytes copied.  Length is the third parameter.
*/

movmem (from, to, length)

unsigned char from[];  /* Memory address to copy from */
unsigned char to[];    /* Address to copy to */
int  length;  /* Length (in bytes) to copy */

{
    register int i=0;

    while (i < length)
        {
         to[i] = from[i];
         i++;
        }
}
```

```
#include "net.h"
#include "iocontrol.h"

/*
   Preemption - return boolean value indicating whether or not the current
                queue has been interrupted so that a higher-priority message
                can be sent to another system via the line in use.  Check
                for the file INTFILE in the given queue (directory).
*/

int Preemption (queuename)

pathname *queuename;  /* Name of directory to check for INTFILE file */

{
   FILE *fp;  /* Temporary file pointer to be released if successful */
   pathname IntFileName;  /* Storage for complete file path */

   sprintf (IntFileName, "%s/%s", queuename, INTFILE);  /* Construct path */

   if ((fp = fopen (IntFileName, "r")) == NULL)
      {
       /* The file was not found, or else we have a bad problem. Say 'no' */
       return (FALSE);  /* Indicate no need to relinquish line */
      }

   fclose (fp);  /* Make sure we close the file */

   return (TRUE);  /* Indicate that we have agreed to terminate */
}
```

```
#include "net.h"
#include "iocontrol.h"
#include "iocontrol.e"

/*
   Send - output a 'packet' to the output device (ModemFd).  Do not make
          any assumptions about format (permit binary if needed).
          Place an EM (End-of-Message) followed by an ASCII carriage-
          return (CR) after the packet as part of the low-level protocol.
*/

int Send (string, length)

unsigned char *string; /* String of characters to send (may be binary) */
int  length;  /* Length of string to send in bytes */

{
   unsigned char Tail [3];

#ifdef FORHONEY
   register int i;
   register int j=0;
   unsigned char *newstr;

   newstr = (unsigned char *) malloc (length*2);

   for (i=0; i < length; i++)
       {
        if ((string[i] == (unsigned char) '\\') ||
            (string[i] == (unsigned char) '\r'))
           newstr[j++] = (unsigned char) '\\';

        newstr[j++] = string[i];
       }

   /* Check to see if Honeywell is ready for entire packet by sending
      just the first character and checking for BEEP */

   FlushModemInput (ModemFd);   /* Clear input buffer before sending */

   write (ModemFd, newstr, 1);  /* Output the first character to Honeywell */

   while (WaitBeep (1))         /* Wait one second for a beep */
      {
       /* Remote system was not ready -- keep trying */
       sleep (1);               /* Always wait one second before retrying */
       FlushModemInput (ModemFd);
       write (ModemFd, newstr, 1);
      }

   write (ModemFd, &(newstr[1]), j-1);  /* Output remainder of string */
   free (newstr);

#else

   write (ModemFd, string, length);  /* Output the string */

#endif

   Tail [0] = DLE;  Tail [1] = EM;  Tail [2] = CRET;
```

```
    write (ModemFd, Tail, 3);  /* Always terminate with a carriage-return */
}
```

```
#include "net.h"
#include "iocontrol.h"
#include "iocontrol.e"

extern int MaxRetry;

/*
   SendBlock - send a block to the remote system until received correctly
               or until retry limit is reached.
*/

int SendBlock (data, length, endflag)

unsigned char *data;    /* Data to be transmitted, null-terminated */
int  length;   /* Length of block in bytes (can't use string fns) */
int  endflag;  /* Flag indicating this is the last block of a message */

{
   int RetryCount = 0;  /* Count of number of negative acknowledgements */
   int Result;          /* Result code from WaitAck */

   while (RetryCount <= MaxRetry)  /* Do not try forever */
      {
       FlushModemInput (ModemFd);   /* Dump the garbage */

       SendPacket (data, length, endflag);  /* Try to send the packet */

       if ((Result = WaitAck ()) == (int) NULL)
          {
           SeqNo = (SeqNo + 1) % 0x100; /* Give us next 2-byte sequence no. */
           break;                       /* Exit loop acknowledge received */
          }

#ifdef FORHONEY
       if (Result == NAK)
          {
           sleep (2);       /* Honeywell requires special error recovery */
           SendByte (ACK);  /* Acknowledge the NAK to flush Honey buffer */
           sleep (2);
          }
#endif

       RetryCount++;

       sleep (1);     /* Hold off on retransmit for a second in all cases */
      }

   return (RetryCount <= MaxRetry);  /* Return error condition */
}
```

```
#include "net.h"
#include "iocontrol.h"
#include "iocontrol.e"

/*
   SendByte - send a single control byte followed by End-of-Message
              (EM) and Carriage-Return (CR) as dictated by the low-
              level protocol.
*/

int SendByte (byte)

unsigned char byte;

{
   unsigned char Tail [3];

#ifdef FORHONEY

   /* Check to see if Honeywell is ready for entire packet by sending
      just the first character and checking for BEEP */

   FlushModemInput (ModemFd);   /* Clear input buffer before sending */

   write (ModemFd, &byte, 1);   /* Output the first character to Honeywell */

   while (WaitBeep (1))         /* Wait one second for a beep */
      {
       /* Remote system was not ready -- keep trying */
       sleep (1);               /* Always wait one second before retrying */
       FlushModemInput (ModemFd);
       write (ModemFd, &byte, 1);
      }

#else

   write (ModemFd, &byte, 1);  /* See send.c */

#endif

   Tail [0] = DLE;
   Tail [1] = EM;
   Tail [2] = CRET;
   write (ModemFd, Tail, 3);
}
```

```
#include "net.h"
#include "iocontrol.h"
#include "iocontrol.e"

extern int MaxRetry;

/*
   SendEnq - send an enquire signal (ENQ), and await response with an
             acknowledge signal from master.  Time out after specified
             retry limit is exceeded.
*/

int SendEnq ()

{
   unsigned char *Buffer;  /* Response buffer allocated by GetPacket */
   int RetryCount = 0;     /* Counter for number of retries */
   int Length;             /* Response buffer length (not used) */

   FlushModemInput (ModemFd);  /* Clear input buffer before sending ENQ */

   while (RetryCount++ < MaxRetry)
      {
       SendByte (ENQ);  /* Transmit enquire signal */
       if ((Buffer = GetPacket (&Length)) == NULL)
         {
          if (DebugLevel)
             WriteLog ("SendEnq:", "Timed out waiting for ACK", "", "");
         }
       else
          if (Buffer [0] == ACK)
             break;  /* Response was good, systems are in sync */
          else
             free (Buffer); /* release storage */
      }

   FlushModemInput (ModemFd);  /* Get rid of possible extra characters */

   if (RetryCount >= MaxRetry)
      {
       WriteLog ("SendEnq:", "timed out awaiting", "contact with master.", "");
       Exit (LOSTCONTACT);
      }

   free (Buffer); /* release storage */
}
```

```
#include "net.h"
#include "iocontrol.h"
#include "iocontrol.e"

extern pathname QueueName;  /* Reference semi-global variable */
extern int CheckBetween;
extern int BlockLength;     /* Block size in a packet in bytes */
extern int HoneyBlock;

/*
   SendFile - send a file to the remote system, breaking it into certain
              sized 'blocks' for better error detection.  Also preserve
              original file name.
*/

int SendFile (filepathname, messagename)

pathname *filepathname;
char *messagename;   /* Message file name - unique throughout network */

{
   FILE *FileFd;
   int BlockSize;  /* Size of each block transmitted */
   unsigned char *BlockPtr;
   int EndOfText;
   unsigned int c;
   int MaxBlock;   /* Depends on 'FORHONEY' */

#ifdef FORHONEY
   MaxBlock = HoneyBlock;
#else
   MaxBlock = BlockLength;     /* Maximum length of data in a packet */
#endif

   if ((FileFd = fopen (filepathname, "r")) == NULL)
      {
       WriteLog ("SendFile:", "cannot open message file:", filepathname, "");
       return (ERR);
      }

   /* Begin message introduction session */

   if (SendHeader (messagename) != ACK)  /* Send header for message name */
      {
       WriteLog ("SendFile:", filepathname, "rejected by remote", "");
       return (NULL);          /* Normal return - just don't send file */
      }

   /* Remote will accept message; begin transmission */

   if (Preemption (QueueName))
      {
       SendByte (EOT);  /* Other system will know */
       WriteLog ("SendFile:", "INTERRUPTED while sending", filepathname, "");
       Exit (INTERRUPTED);
      }
   else if (MasterMode && Abort ("."))
      {
       SendByte (CAN);  /* CANCEL connection */
       WriteLog ("SendFile:", "ABORTED while sending", filepathname, "");
```

```
        ShutDown ();      /* Definitely abort */
       }

   BlockPtr = (unsigned char *) malloc (MaxBlock + 1); /* Allocate storage */

   EndOfText = FALSE;   /* Set to transmit ETB after each block */

   while (!EndOfText)   /* Loop until all blocks are sent */
      {
       if (CheckBetween)
          {
           /* First of all, check to see if a higher entity requests the line */
           if (Preemption (QueueName))
              {
               /* We must relinquish this line */
               SendByte (EOT);  /* Other system will understand */
               WriteLog ("SendFile:", "INTERRUPTED while sending", filepathname,
                         "");
               Exit (INTERRUPTED);
              }
           else if (MasterMode && Abort ("."))
              {
               /* We have been asked to shut down */
               SendByte (CAN);  /* Other system will know why we stopped */
               WriteLog ("SendFile:", "ABORTED while sending",
                         filepathname, "");
               ShutDown ();
              }
          }

       BlockSize = 0;
       while (BlockSize < MaxBlock)  /* Blocks may overrun by at most 1 */
          {
           if ((c = getc (FileFd)) == EOF)
              break;       /* Exit the loop if end of file within a block */

           c = c & (unsigned int) 0xff;       /* Just keep the bottom 8 bits */

           if (c == DLE)  /* Check for data-link escape in file */
              BlockPtr [BlockSize++] = c;     /* Go ahead */

#ifdef FORHONEY
           if (c == '\n')
              c = '\r';  /* Convert NL to CR for Honeywell */
#endif

           BlockPtr [BlockSize++] = c;        /* Place character in block */
          }

       /* We have a block, now transmit it */

       EndOfText = (c == EOF);  /* We read an end-of-file, end with ETX */

       if (!SendBlock (BlockPtr, BlockSize, EndOfText)) /* Send & Wait */
          {
           WriteLog ("SendFile:", "Timeout awaiting acknowledgement.", "", "");
           WriteLog ("SendFile:", "Could not transmit", filepathname, "");
           return (ERR);  /* Probably lost contact */
          }
      }
```

```
    WriteLog ("SendFile:", "message", filepathname, "sent OK.");
    fclose (FileFd);
    free (BlockPtr);  /* Free storage now */
    return (NULL);
}
```

```
#include "net.h"
#include "iocontrol.h"
#include "iocontrol.e"

/*
   SendHeader - send current message identification header and await
                remote acceptance.  (Remote may deny a message by header
                alone, indicating it has already received the message.)
                Remote must respond with two ACK's.  The first indicates
                that the message i.d. header packet was received correctly,
                and the second is the remote's acceptance of the message.
*/

SendHeader (messagename)

char *messagename;          /* Name of message being sent */

{
   int NameLength;          /* Length of message name to transmit */
   int GoodResult;          /* Result from 'sendblock' call */
   int EndFlag = TRUE;      /* End of text flag for 'sendblock' */
   int Length;              /* Length of response buffer - space holder */
   int Result;              /* Result from 'waitack' */

   NameLength = strlen (messagename);  /* Compute length of string */

   FlushModemInput (ModemFd);  /* Clear input buffer before sending header */

   /* Send the block and wait for an acknowledgement */

   GoodResult = SendBlock (messagename, NameLength, EndFlag);

   if (!GoodResult)
      {
       WriteLog ("SendHeader:", "can't send message header:", messagename, "");
       Exit (LOSTCONTACT);
      }

   /* Now acknowledge message acceptance - must get double acknowledge */

   Result = WaitAck ();  /* See if remote sends an acknowledgement */
   if (Result == ERR)
      {
       WriteLog ("SendHeader:", "TIMED OUT waiting for acceptance:",
                 messagename, "");
       Exit (LOSTCONTACT);
      }

   FlushModemInput (ModemFd);  /* Make sure no garbage remains */

   return (Result == NULL ? ACK : NAK); /* Return acceptance flag */
}
```

```
#include "net.h"
#include "iocontrol.h"
#include "iocontrol.e"

/*
   SendName - send this system's node name to remote system for secondary
              login and identification.  (Remote system needs to know who
              called so it can check for outgoing messages for that system.)
*/

SendName ()

{
   char *ThisSiteName;      /* Name of current site */
   int SiteNameLength;      /* Length of site name to transmit */
   int GoodResult;          /* Result from 'sendblock' call */
   int EndFlag = TRUE;      /* End of text flag for 'sendblock' */
   int Length;              /* Length of response buffer - not used */
   int Result;              /* Result from 'waitack' */

   ThisSiteName = MyName ();  /* Get the name of this node */

   SiteNameLength = strlen (ThisSiteName);

   FlushModemInput (ModemFd);  /* Clear input buffer before sending name */

   /* Send the block and wait for an acknowledge */

   GoodResult = SendBlock (ThisSiteName, SiteNameLength, EndFlag);

   free (ThisSiteName);

   FlushModemInput (ModemFd);  /* Get rid of possible extra information */

   if (!GoodResult)
      {
       WriteLog ("SendName:", "could not achieve","validation from slave", "");
       Exit (LOSTCONTACT);
      }

   /* Now acknowledge login - must get double acknowledge */

   Result = WaitAck ();  /* See if remote sends an acknowledgement */
   if (Result == NAK)
      {
       WriteLog ("SendName:", "BAD SITE NAME", "(remote does not know me)", "");
       Exit (BADCONNECTION);
      }
   else if (Result == ERR)
      {
       WriteLog ("SendName:", "TIMED OUT waiting for second ACK", "", "");
       Exit (LOSTCONTACT);
      }

   /* Simply returning indicates success.  Errors are fatal, causing exit. */
}
```

```
#include "net.h"
#include "iocontrol.h"
#include "iocontrol.e"

/*
   SendPacket - construct a 'packet' with checksum
                using DLE as a data-link escape symbol (precedes ETX/ETB).
                The parameter "end" indicates if this is the end of a message
                or if there is more to come.
*/

int SendPacket (data, length, end)

unsigned char *data;   /* Pointer to character data (binary permitted) */
int  length;  /* Length of packet to transmit */
int  end;     /* End of text flag */

{
   unsigned char *Packet;  /* Must allocate storage in this routine */
   int  Ptr;       /* Pointer into packet */
   int  Code;      /* Checksum code returned */
   int  i;
#ifdef DEBUG
   FILE *tmp=fopen("/usr/taccnet/log/proto.log", "a");
#endif

   Packet = (unsigned char *) malloc (length + PKTOVERHEAD);  /* Reserve data */

   if (Packet == NULL)
      {
       WriteLog ("SendPacket:", "OUT OF MEMORY", "allocating for packet", "");
       Exit (INTERNALERROR);
      }

   sprintf (Packet, "%.2X", SeqNo);  /* Encode packet number - ASCII hex */

   Packet [2] = STX;

   for (i = 0; i < length; i++)    /* Copy data section into packet */
        Packet [i+3] = data [i];  /* (Transparency already taken care of) */

   Ptr = i + 3;

   if (Ptr > (length + PKTOVERHEAD))
      {
       WriteLog ("SendPacket:", "memory violation", "building packet", "");
       Exit (INTERNALERROR);
      }

   Packet [Ptr++] = DLE;  /* Place Data-link Escape before end-text marker */
   Packet [Ptr++] = (end ? ETX : ETB);  /* End-of-text or -text-block */

   Code = CheckSum (Packet+3, Ptr-4);   /* Compute checksum on data & ETX */

   sprintf (Packet + Ptr, "%.4X", Code);

   /* Now that the packet is constructed, send it to low-level I/O */

#ifdef DEBUG
   fprintf(tmp,"----------------------------------------\n");
```

```
    for (i=0;i<Ptr+4;i++) fprintf(tmp," %.2x",Packet[i]);
    fprintf(tmp,"\n---------------------------------------\n");
    fclose(tmp);
#endif

    Send (Packet, Ptr + 4);  /* Sends a sequence of n characters */

    free (Packet);  /* Free storage */
}
```

```
#include "net.h"

extern int ModemFd;
struct termio TTYSet;

int SetPort (PortName)

char *PortName;

/*
Open the named port for use by IOControl.  Make sure to set the line
parameters correctly: no buffering, 1200bps, ignore input parity, 7 bits out,
even parity out, no echo, etc. Returns ERR if the modem cannot be contacted
or ModemFd if all is well. Set global variable ModemFd.
*/

{
   register int OFlag = O_RDWR;

   if ((ModemFd = open (PortName, OFlag)) == NULL)
      {
       WriteLog ("SetPort: Can't open", PortName, "", "");
       return (ERR);
      }

    if (ioctl (ModemFd, TCGETA, &TTYSet) == ERR) /* get old modem settings */
       {
        WriteLog ("SetPort: Can't get old modem settings from", PortName,
                  "", "");
        return (ERR);
       }

    TTYSet.c_iflag &= ~INPCK;          /* Don't check input parity    */
    TTYSet.c_iflag &= ~ICRNL;          /* Don't convert CR to NL      */
    TTYSet.c_iflag &= ~IXON;
    TTYSet.c_iflag &= ~IXOFF;

#ifdef FORHONEY
    TTYSet.c_iflag |= ISTRIP;
#else
    TTYSet.c_iflag &= ~ISTRIP;
#endif

#ifdef FORHONEY
    TTYSet.c_cflag |= PARENB;          /* enable parity               */
    TTYSet.c_cflag &= ~PARODD;         /* even parity                 */
    TTYSet.c_cflag &= ~CSIZE;          /* clear character size field  */
    TTYSet.c_cflag |= CS7;             /* Set 7 bits data, 1 parity   */
#else
    TTYSet.c_cflag &= ~PARENB;         /* disable parity              */
    TTYSet.c_cflag &= ~CSIZE;          /* clear character size field  */
    TTYSet.c_cflag |= CS8;             /* Set 8 data bits, no parity  */
#endif

    TTYSet.c_cflag &= ~CBAUD;          /* Clear old baud-rate bits    */
    TTYSet.c_cflag |= B1200;           /* Set baud to 1200            */

    TTYSet.c_cflag |= CLOCAL;          /* assert DTR and RTS to modem */
    TTYSet.c_cflag |= HUPCL;           /* hang up on last close       */
```

```
    TTYSet.c_lflag &= ~ISIG;            /* Ignore QUIT and INTR sigs   */
    TTYSet.c_lflag &= ~ICANON;          /* Don't want canonical input  */
    TTYSet.c_lflag &= ~ECHO;            /* No echo                     */

    TTYSet.c_oflag &= ~OPOST;           /* Don't post-process output   */

    TTYSet.c_cc [VMIN] = 1;             /* MIN = 1 char (no buffering) */
    TTYSet.c_cc [VTIME] = 0;            /* Expect data after 0 ms      */

    if (ioctl (ModemFd, TCSETA, &TTYSet) == ERR) /* set new modem attributes */
       {
        WriteLog ("SetPort: Can't set new modem attributes on", PortName,
                  "", "");
        return (ERR);
       }

    return (ModemFd);
}
```

```
#include "net.h"

extern int DebugLevel;

int TouchSite (SysName)
char *SysName;  /* name of the system to update */

/* reset site table entry to indicate successful contact */

{
   site *Site;  /* allocated by ValidSite() */

   if ((Site = ValidSite (SysName)) != NULL)  /* Site is valid */
      {
       Site->Status = UP;       /* declare site up    */
       Site->TimeToCall = NOW; /* reset time to call */
       Site->NumCalls = 0;      /* reset retry count  */
       PutSite (Site);          /* save entry         */
      }
   else  /* ValidSite returned NULL */
      if (DebugLevel)
         WriteLog ("TouchSite: No active entry for", SysName,
                   "in", SITETABLE);
}
```

```
#include "net.h"
#include "iocontrol.h"
#include "iocontrol.e"

/*
   WaitAck - wait for an acknowledgement signal from remote system and
             return error status if "NAK" received or if timeout occurs.
*/

int WaitAck ()

{
   unsigned char *ResponsePacket;  /* From "getpacket", allocated therein */
   int  ResultStatus;      /* Status to pass to caller */
   int  PacketLength;      /* Needed for call to "getpacket" (unused) */

   if ((ResponsePacket = GetPacket (&PacketLength)) == NULL)
      ResultStatus = ERR;  /* Timeout before packet was received */
   else
      if (*ResponsePacket == ACK)
         ResultStatus = NULL;
      else if (*ResponsePacket == EOT)
         {
          WriteLog ("WaitAck:", "remote system was preempted;",
                    "connection terminated", "");
          Exit (GOOD);
         }
      else if (*ResponsePacket == CAN)
         {
          WriteLog ("WaitAck:", "remote system shut down suddenly;",
                    "connection terminated", "");
          Exit (GOOD);
         }
      else
         ResultStatus = NAK; /* Differentiate from ERROR and error */

   if (ResponsePacket != NULL)
      free (ResponsePacket); /* Get rid of unneeded storage */

   return (ResultStatus);
}
```

```
#include "net.h"
#include <signal.h>
#include "iocontrol.h"
#include "iocontrol.e"

/*
   WaitBeep - check the modem for a BELL character (ctrl-G) from the
              Honeywell indicating that the buffer on the Honeywell
              is not ready for more input.  Wait the specified number
              of seconds for the bell, then return TRUE if the bell
              occurred, FALSE otherwise.
*/

int WaitBeep (sec)

int sec;  /* Number of seconds to wait for beep */

{
   char c;

   int stopread ();  /* Interrupt handler to abort read after sec seconds */

   alarm (0);                    /* Clear pending interrupt */
   signal (SIGALRM, stopread);  /* Set interrupt handler */

   alarm (sec);      /* Wait sec seconds before sending SIGALRM signal */

   if (read (ModemFd, &c, 1) > 0)  /* Read a character */
      {
       /* A beep indeed occurred */
       alarm (0);
       FlushModemInput (ModemFd);
       return (c == '\007');
      }
   else
      return (FALSE);              /* A beep was not received */
}




int stopread ()
{
   alarm (0);
   return (-1);  /* Cause read to abort with -1 result code */
}
```

```
#include "net.h"
#include "iocontrol.h"
#include "iocontrol.e"

extern int MaxRetry;

/*
   WaitEnq - wait for an enquire signal from remote system and
             return error status if timeout occurs.  Send acknowledgement
             signal if enquiry received.  This establishes synchronization
             with remote after local system is activated.
*/

int WaitEnq ()

{
   unsigned char *ResponsePacket; /* From "getpacket", allocated therein */
   int  PacketLength;             /* Needed for call to "getpacket" (unused) */
   register int Count = 0;        /* retry counter */

   while (Count++ < MaxRetry)
      if ((ResponsePacket = GetPacket (&PacketLength)) == NULL)
         {
          if (DebugLevel)
             WriteLog ("WaitEnq:", "Timed out waiting for ENQ", "", "");
          SendByte (NAK);
         }
      else
         if (*ResponsePacket == ENQ)
            break;
         else
            free (ResponsePacket); /* release storage */

   FlushModemInput (ModemFd);

   if (Count >= MaxRetry)
      {
       WriteLog ("WaitEnq:", "Did not receive ENQ packet", "", "");
       SendByte (NAK);  /* Make sure remote understands */
       WriteLog ("WaitEnq:", "Bad connection - Goodbye!", "", "");
       Exit (BADCONNECTION);  /* Could not synchronize */
      }

   free (ResponsePacket); /* release storage */

   SendByte (ACK);  /* Transmit acknowledge signal */
}
```

```
#include "net.h"
#include "iocontrol.h"
#include "iocontrol.e"

/*
   WaitName - wait for the remote system to identify itself and
              return error status if timeout occurs.  Send acknowledgement
              signal if valid name received.  This ensures security
              and facilitates bidirectionality of system.
*/

int WaitName (RemoteName)

sitename RemoteName;  /* Name of remote system; storage must be pre-allocated */

{
   unsigned char *ResponseBlock; /* Block from "getblock", allocated therein */
   int  EndFlag;          /* End of text flag - needed for 'getblock' */
   int  BlockLength;      /* Length of block (remote-name) returned */

   if ((ResponseBlock = GetBlock (&BlockLength, &EndFlag)) == NULL)
      {
       /* Unexpected condition - no error-handling here */
       WriteLog ("WaitName:", "did not receive remote name properly.", "", "");
       SendByte (NAK);  /* Make sure remote knows there's a problem */
       Exit (BADCONNECTION);  /* Synchronization bad at initial connection */
      }

   if (BlockLength > SITENAMELEN)  /* Don't permit overindexing */
      {
       /* Remote system name is not of proper form */
       WriteLog ("WaitName:", "remote name is not of proper form.", "", "");
       SendByte (NAK);  /* Tell remote there's a problem */
       free (ResponseBlock);
       Exit (BADCONNECTION);
      }

   /* Move site name to caller's storage;  NOTE: may still be bad data */

   strncpy (RemoteName, ResponseBlock, BlockLength);

   /* We allow unknown sites to log in, since sites may come & go using */
   /* administrative messages to  change our tables. They still have to */
   /* get in through password security at the login level.              */

   /* Validate the site before going on */
   if ((DebugLevel) && (ValidSite (RemoteName) == NULL))
       WriteLog ("WaitName:", "contact from unknown or inactive site",
                  RemoteName, ")");

   FlushModemInput (ModemFd);

   free (ResponseBlock); /* Get rid of unneeded storage (smart) */

sleep (3);  /* HONEYWELL KLUDGE */
   SendByte (ACK);  /* Transmit acknowledge signal */
}
```

MSGPROC

This section contains the functions used only by the Message Processor program (MSGPROC).

```
msgproc: filenq.o fileopen.o forward.o getaltsites.o lockfile.o\
  getmsghead.o makecc.o newfile.o remove.o reroute.o process.o\
  msgproc.o dequeue.o getdir.o myname.o handlerror.o writelog.o\
  datetime.o stripme.o mpmail.o administer.o readsite.o activatesite.o\
  changesite.o deletesite.o examinesite.o validsite.o hold.o\
  validuser.o getline.o high.o frename.o save.o transfer.o\
  archive.o recover.o hashtime.o abort.o
        cc -O -o msgproc filenq.o fileopen.o forward.o getaltsites.o\
  getmsghead.o makecc.o newfile.o remove.o reroute.o process.o hold.o\
  msgproc.o dequeue.o getdir.o myname.o handlerror.o writelog.o\
  datetime.o stripme.o mpmail.o administer.o lockfile.o validsite.o\
  readsite.o activatesite.o changesite.o deletesite.o examinesite.o\
  validuser.o getline.o high.o frename.o save.o transfer.o\
  archive.o recover.o hashtime.o abort.o
        strip msgproc

msgproc.o: net.h  msgproc.c
        cc -O -c msgproc.c

getline.o: net.h getline.c
        cc -c -O getline.c

transfer.o: net.h transfer.c
        cc -O -c transfer.c

save.o: net.h save.c
        cc -O -c save.c

activatesite.o: net.h activatesite.c
        cc -O -c activatesite.c

changesite.o: net.h changesite.c
        cc -O -c changesite.c

deletesite.o: net.h deletesite.c
        cc -O -c deletesite.c

examinesite.o: net.h examinesite.c
        cc -O -c examinesite.c

hold.o: net.h hold.c
        cc -O -c hold.c

reroute.o: net.h  reroute.c
        cc -O -c reroute.c

process.o: net.h  process.c
        cc -O -c process.c

mpmail.o: net.h mpmail.c
        cc -O -c mpmail.c

administer.o: net.h administer.c
        cc -O -c administer.c

forward.o: net.h  forward.c
        cc -O -c forward.c

getaltsites.o: net.h  getaltsites.c
        cc -O -c getaltsites.c
```

```
getmsghead.o: net.h  getmsghead.c
        cc -0 -c getmsghead.c

handlerror.o: net.h  handlerror.c
        cc -0 -c handlerror.c

makecc.o: net.h  makecc.c
        cc -0 -c makecc.c

validuser.o: net.h validuser.c
        cc -0 -c validuser.c

high.o: net.h high.c
        cc -c -0 high.c

recover.o: net.h recover.c
        cc -c -0 recover.c

hashtime.o: net.h hashtime.c
        cc -c -0 hashtime.c

rmhdr: net.h rmhdr.c
        cc -o rmhdr rmhdr.c
        strip rmhdr
```

```
/* Global variable descriptions */

extern int DebugLevel;  /* Runtime debug level (0 = normal) */
```

```
#include "net.h"

extern int DebugLevel;  /* Runtime debug level (0 = normal) */

int ActivateSite (Fd)

FILE *Fd;

/*
      ActivateSite - Read a site name from the stream Fd and look it up
                     in the site table.
                     If it is defined and active, do nothing.
                     If it is not defined, add it to the table.
                     If it is defined but deactivated, reactivate the site by
                     changing the site delimiter DELCHAR to a FIELDMARK
                     so that ValidSite will see it.
*/

{
   FILE     *SiteTableFd;
   register int Result;
   register int c, n=0;
   long     StartOfLine;
   sitename Site;
   sitename TempBuf;
   int      Status;

   SkipEOL (Fd);   /* skip to beginning of site entry */

   fscanf (Fd, "%c%s", &Status, Site);  /* read the site name from the file */

   if (DebugLevel)
      WriteLog ("ActivateSite: activate site", Site, "", "");

   if ((Lock (SITETABLE)) == ERR)
      {
       WriteLog ("DeleteSite: Can't lock", SITETABLE, "", "");
       exit (1);
      }

   if ((SiteTableFd = fopen (SITETABLE, "r+")) == NULL)
      {
       WriteLog ("DeleteSite: Can't open", SITETABLE, "", "");
       UnLock (SITETABLE);
       return (ERR); /*indicate failure */
      }

   /* find the desired site and reactivate it */

   while ((c = getc (SiteTableFd)) != EOF)
      {
       StartOfLine = ftell (SiteTableFd) - (long)1;
       fscanf (SiteTableFd, "%s", TempBuf);  /* get a site name */
       if (EQUALS (TempBuf, Site))  /* it is the one we seek */
         {
          if (DebugLevel)
             WriteLog ("ActivateSite: site", Site, "found in", SITETABLE);

          switch (c)
             {
```

```
            case DELCHAR :
              {
               fseek (SiteTableFd, StartOfLine, 0); /* reset */
               fprintf (SiteTableFd, "%c", FIELDMARK); /* make site active */
               WriteLog ("ActivateSite:", Site, "has been activated", "");
               Result = GOOD;
               break;
              }
            case FIELDMARK :
              {
               WriteLog ("ActivateSite: site", Site, "already active.", "");
               Result = GOOD;
               break;
              }
            default :
              {
               WriteLog ("ActivateSite: Invalid site entry for",Site,"","");
               Result = ERR;
               break;
              }
           }
         fclose (SiteTableFd);
         UnLock (SITETABLE);
         return (Result);
        }
     else
        SkipEOL (SiteTableFd);
    }

  /* arrive here only if site was not defined in site table */
  if (DebugLevel)
    WriteLog ("ActivateSite: site", Site, "not found in", SITETABLE);


  /* file is in update mode - we must close and reopen in append mode */
  if ((SiteTableFd = freopen (SITETABLE, "a", SiteTableFd)) == NULL)
     {
      WriteLog ("ActivateSite: Can't open ", SITETABLE, "in append mode", "");
      UnLock (SITETABLE);
      return (ERR);
     }

  if (DebugLevel)
    WriteLog ("ActivateSite: adding site", Site, "to", SITETABLE);

  /* write the site name, then copy the rest of the site entry */
  fprintf (SiteTableFd, "%c%s", Status, Site);
  while ((c = getc (Fd)) != EOF)
     putc (c, SiteTableFd);
  fclose (SiteTableFd);

  WriteLog ("ActivateSite: ", Site, " has been added", "");

  UnLock (SITETABLE);
  return (GOOD);

}
```

```
#include "net.h"

extern int DebugLevel;

int Administer (MsgFileName)

char *MsgFileName;   /* Message file name to process */

/*
   Administer - interpret and execute network administrative command
                contained in file MsgFileName in MSGPROCQ
*/

{
   int Result;
   FILE *MsgFileFd;
   int c;

   /* read and parse the command */

   if (( MsgFileFd = FileOpen (MsgFileName, MSGPROCQ, "r")) == NULL)
      {
       WriteLog ("Administer: can't open", MsgFileName, "", "");
       return (ERR);
      }

   while ((c = getc (MsgFileFd)) == HEADERLINE)  /* skip to command line */
      SkipEOL (MsgFileFd);

   if (DebugLevel)
      {
       char com[2];
       sprintf (com, "%c", c);
       WriteLog ("Administer: file =", MsgFileName, "command =", com);
      }

   switch (c)
     {
      case ADD      : Result = ActivateSite (MsgFileFd); break;
      case CHANGE   : Result = ChangeSite (MsgFileFd);   break;
      case DELETE   : Result = DeleteSite (MsgFileFd);   break;
      case EXAMINE  : Result = ExamineSite (MsgFileFd);  break;
      case RECOVER  : Result = Recover (MsgFileFd);      break;
      case SAVE     : Result = Save (MsgFileFd);         break;
      case TRANSFER : Result = Transfer (MsgFileFd);     break;

      default       : WriteLog ("Administer: bad command in",
                                MsgFileName, "", "");
                      Result = ERR;
                      break;
     }

   fclose (MsgFileFd);
   return (Result);    /* errors are handled by msgproc */
```

```
#include "net.h"

int ChangeSite (MsgFd)

FILE *MsgFd;

{
   FILE *SiteTableFd;
   FILE *NewSitesFd;
   sitename SiteName;
   sitename TargetName;
   register int c;

   if (Lock (SITETABLE) != GOOD)
      {
       WriteLog ("ChangeSite:", "can't lock", SITETABLE, "");
       return (ERR);
      }

   if ((NewSitesFd = fopen (NEWSITETABLE, "w")) == NULL)
      {
       WriteLog ("ChangeSite:", "can't create", NEWSITETABLE, "");
       return (ERR);
      }

   if ((SiteTableFd = fopen (SITETABLE, "r")) == NULL)
      {
       WriteLog ("ChangeSite:", "can't open", SITETABLE, "");
       fclose (NewSitesFd);
       return (ERR);
      }

   SkipEOL (MsgFd);  /* skip to line containing site entry */
   fscanf (MsgFd, "%*c%s", TargetName); /* read in the name of the site */

   /* copy current table to new table file, replacing designated site data */
   c = getc(SiteTableFd);
   while (c != EOF)
      {
       putc (c, NewSitesFd);
       if (c == FIELDMARK)     /* then we are at a site entry line */
          {
           fscanf (SiteTableFd, "%s", SiteName);  /* get the site name */
           fprintf (NewSitesFd, "%s", SiteName); /* go ahead and copy it */
           if (EQUALS (SiteName, TargetName)) /* then we replace the old data */
              {
               while ((c=getc(MsgFd)) != EOF)  /* copy new site data */
                  putc (c, NewSitesFd);
               while ((c=getc(SiteTableFd)) != FIELDMARK) /* skip old data */
                  SkipEOL (SiteTableFd);
               ungetc (c, SiteTableFd);  /* reset input stream */
               WriteLog ("ChangeSite:", SiteName, "has been changed", "");
              }
          }
       c = getc(SiteTableFd);
      }

   fclose (NewSitesFd);
   fclose (SiteTableFd);
```

```
    /* save current site table as sites.old */
    unlink (OLDSITETABLE);
    FRename (SITETABLE, OLDSITETABLE);

    /* replace current table with new table */
    FRename (NEWSITETABLE, SITETABLE);

    UnLock (SITETABLE);

    return (GOOD);
}
```

```
#include "net.h"

int DeleteSite (Fd)

FILE *Fd;

/*
        DeleteSite - Read a site name from the stream Fd and look it up
                     in the site table. If found, deactivate the site by
                     changing the site delimiter FIELDMARK to a DELCHAR
                     so that ValidSite won't see it.
*/

{
   FILE     *SiteTableFd;
   register int c, n=0;
   long     StartOfLine;
   sitename Site;
   sitename TempBuf;


   SkipEOL (Fd); /* skip down to site entry */

   fscanf (Fd, "%*c%s", Site);  /* read the site name from the file */

   if ((Lock (SITETABLE)) == ERR)
      {
       WriteLog ("DeleteSite: Can't lock", SITETABLE, "", "");
       exit (1);
      }

   if ((SiteTableFd = fopen (SITETABLE, "r+")) == NULL)
      {
       WriteLog ("DeleteSite: Can't open", SITETABLE, "", "");
       UnLock (SITETABLE);
       return (ERR); /*indicate failure */
      }

   /* find the desired site and deactivate it */

   while ((c = getc (SiteTableFd)) != EOF)
      if (c == FIELDMARK)
         {
          StartOfLine = ftell (SiteTableFd) - (long)1;
          fscanf (SiteTableFd, "%s", TempBuf);  /* get a site name */
          if (EQUALS (TempBuf, Site))  /* it is the one we seek */
             {
              fseek (SiteTableFd, StartOfLine, 0); /* reset */
              fprintf (SiteTableFd, "%c", DELCHAR); /* make site inactive */
              fclose (SiteTableFd);
              UnLock (SITETABLE);
              WriteLog ("DeleteSite:", Site, "has been deactivated", "");
              return (GOOD);  /* all is well */
             }
         }
      else
         SkipEOL (SiteTableFd);

   WriteLog ("DeleteSite:", Site, "was not found in site table", "");
   fclose (SiteTableFd);
```

```
    UnLock (SITETABLE);
    return (ERR);  /* indicate site not found */

}
```

```
#include "net.h"

/*
   Command line format: e[xamine] <site>

   Get a copy of the site table entry for the site specified on the command
   line and send it in a message to the requesting site.
*/

int ExamineSite (MsgFd)

FILE *MsgFd;

{
   register int Found = FALSE;
   register int c;
   sitename Priority, Destination, Target, SiteName;
   pathname Command;
   FILE *SiteFd;
   FILE *PipeFd;
   char *ThisName;  /* Pointer to this site's name */

   ThisName = MyName();  /* Get the name of this system */

   rewind (MsgFd);

   /* get priority and originator address */
   fscanf (MsgFd, "%*c%s", Priority);
   SkipEOL (MsgFd);                              /* next line */
   SkipEOL (MsgFd);                              /* skip the To: line */
   fscanf (MsgFd, "%*c%s", Destination);         /* read the From: line */
   SkipEOL (MsgFd);                              /* next line */

   /* skip past any old headers to command line */
   while ((c = getc(MsgFd)) == HEADERLINE)
      SkipEOL(MsgFd);
   ungetc (c, MsgFd);

   /* get the name of the site to be examined - second word on command line */
   fscanf (MsgFd, "%*s %s", Target);

   if (strlen(Target) == 0)
      {
       WriteLog ("ExamineSite: invalid command line", "", "", "");
       return (ERR);
      }

   if (Lock(SITETABLE) != GOOD)
      {
       WriteLog ("ExamineSite:", "can't lock", SITETABLE, "");
       return (ERR);
      }

   /* build command string for popen to genmsg program */
   sprintf (Command, "%s/bin/%s %s %s", MASTERQ, GENMSG, Priority, Destination);

   /* open pipe to genmsg, write site entry data */
   if ((PipeFd=popen(Command, "w")) == NULL)
      {
       WriteLog ("ExamineSite:", "can't open pipe to", Command, "");
```

```
        UnLock (SITETABLE);
        return (ERR);
      }

   if ((SiteFd = fopen (SITETABLE, "r")) == NULL)
      {
       WriteLog ("ExamineSite:", "can't open", SITETABLE, "");
       pclose (PipeFd);
       UnLock (SITETABLE);
       return (ERR);
      }

   /* search the site table and copy the site entry to the pipe */
   for (c = getc(SiteFd); (c != EOF) && (!Found); c = getc(SiteFd))
      {
       fscanf (SiteFd, "%s", SiteName);
       if (EQUALS(SiteName, Target))
          {
           Found = TRUE;  /* this will get us out of the loop */
           fprintf (PipeFd, "%c%s", c, SiteName); /* copy first fields */
           while ((c=getc(SiteFd)) != FIELDMARK) /* copy the rest of the data */
              putc (c, PipeFd);
          }
       else
          while (((c=getc(SiteFd)) != FIELDMARK) && c != DELCHAR)
              SkipEOL (SiteFd);   /* skip to the next site entry */
      }

   if (!Found) /* tell requestor that the site is not defined on this node */
       fprintf (PipeFd, "Site %s is not defined at node %s\n",
                Target, ThisName);

   free (ThisName);

   fclose (SiteFd);
   UnLock (SITETABLE);

   if (pclose (PipeFd) != 0)
      {
       WriteLog ("ExamineSite:", "error executing", Command, "");
       return (ERR);
      }

   if (Found)
      WriteLog ("ExamineSite: a copy of site", Target,
                "has been sent to", Destination);
   else
      WriteLog ("ExamineSite:", Target,
                "not defined, negative response sent to", Destination);

   return (GOOD);
}
```

```
#include "net.h"

/*
   Forward -- Put a new header on the current message directing
   it to the chosen alternate site.  Leave the message in MSGPROCQ
   for disposition.
*/

int Forward (MsgFileFd, NewPath, MsgPriority)

FILE     *MsgFileFd;
sitename NewPath;
int      MsgPriority;

{
   FILE *NewFileFd;       /* New message file with extra header */
   filename NewFileName; /* New message file name               */
   char MsgLine [LINELEN+1];   /* Storage for copying                */
   char *ThisName;        /* Pointer to this system's name      */

   ThisName = MyName ();     /* Get this site's name */
   /* Create new file for forwarded message */
   if ((NewFileFd = NewFile (NewFileName, MESSAGETYPE, MSGPROCQ)) == NULL)
      {
       WriteLog ("Forward: can't create file in", MSGPROCQ, "queue", "");
       return (ERR);  /* Try to recover */
      }

   /* Build a new header, using the old priority */

   /* Original msg priority */
   fprintf (NewFileFd, "%c%d\n", HEADERLINE, MsgPriority);

   /* Path to next site */
   fprintf (NewFileFd, "%c%s\n", HEADERLINE, NewPath);

   /* Indicate this site's id */
   fprintf (NewFileFd, "%c%s\n", HEADERLINE, ThisName);
   free (ThisName);

   /* Then copy the old message after the new header */

   rewind (MsgFileFd);   /* Make sure it's at the top */

   while (fgets (MsgLine, LINELEN, MsgFileFd) != NULL)  /* Loop to EOF */
         {
          fputs (MsgLine, NewFileFd);  /* Perform the copy */
         }

   fclose (NewFileFd);   /* Close the file now that it's ready */

   FileNQ (NewFileName+1, MSGPROCQ);   /* enqueue it */

   return (GOOD);  /* No errors */
}
```

```
#include "net.h"

extern DebugLevel;

/*
   GetAltSites -- Build a list of alternate sites for a downed site
                  so that a message may be rerouted to one of them.
                  These sites are read from a Backup Site Database
                  which lists them in the order in which rerouting
                  should be performed.
*/

int GetAltSites (MsgPath, AltSites)

char *MsgPath;      /* network path from failed message */
char *AltSites[];   /* List of alternate sites to be returned */

{
   sitename SiteName; /* Name of original site that was apparently down */
   sitename TempSiteName;
   sitename SiteList[MAXDOWNSITES+1]; /* array of sitenames */
   FILE     *SiteFileFd;   /* Pointer to alternate sites file */
   int      Done = FALSE;
   int      j = 0;
   int      i = 0;
   char     *Path;    /* another pointer into the message path */

   if ((SiteFileFd = fopen (ALTSITES, "r")) == NULL)
      {
       WriteLog ("GetAltSites: Can't open alternate sites file",
                 ALTSITES,"","");
       return (ERR);
      }

   Path = MsgPath;
   while (((Path = StripMe (Path, SiteName)) != NULL) && (j < MAXDOWNSITES-1))
      strcpy (SiteList [j++], SiteName);
   strcpy (SiteList [j], SiteName);   /* last one in path is original dest */

   /* for each site in list, look for altsites in table */
   for ( ; j >= 0 ; j--)  /* search backward from destination site */
      {
       fseek (SiteFileFd, (long)0, 0);  /* rewind to beginning of file */
       Done = FALSE;
       while ((!Done) && (fgetc (SiteFileFd) != EOF))
          {
           fscanf (SiteFileFd, "%[^:]", TempSiteName);
           if (EQUALS (SiteList[j], TempSiteName))  /* found a site entry */
              {
               if (DebugLevel)
                  WriteLog ("GetAltSites: found", TempSiteName, "", "");
               Done = TRUE;  /* force loop to end & try next site */
               while ((fgetc (SiteFileFd) != NL) && (i <= MAXALTSITES))
                  {
                   /* copy each altsite for this entry */
                   fscanf (SiteFileFd, "%s", AltSites [i]);
                   i++;  /* will be equal to number of altsites found */
                  }
              }
           else
```

```
                SkipEOL (SiteFileFd);
            }
        }

    if (i == 0) /* then we didn't find a single alternate site */
        {
         WriteLog ("GetAltSites:", MsgPath, "has no alternate sites", "");
         fclose (SiteFileFd); /* Clean up after yourself */
         return (ERR);  /* Try to recover */
        }

    if (i <= MAXALTSITES)
       AltSites [i] = NULL;    /* Terminate the list */

    fclose (SiteFileFd);

    return (GOOD);  /* Return no error condition */
}
```

```
#include "net.h"

char *GetLine (Line, Fd)

register char *Line;
register FILE *Fd;

/*
   Read a line of input from the stream Fd into the string Line, removing the
   newline character from the end if it is present.
*/

{
register int i;
char *ptr;

   if ((ptr=fgets (Line, LINELEN, Fd)) != NULL)
      for (i=strlen(Line) ; i>=0; i--)
         if (Line[i] == NL)
           {
            Line[i] = '\0';
            break;
           }

   return (ptr);
}
```

```
#include "net.h"

extern DebugLevel;

/*
   GetMsgHead - Reads and decodes the header from a message file and
                returns a pointer to a structure containing the
                message priority, courtesy-copy status, original destination,
                and a list of all previous sites used for rerouting of
                the message.

*/

header *GetMsgHead (MsgFileFd)

FILE *MsgFileFd;            /* File containing the message itself */

{
   /* Header information structure to be returned */
   static header HeaderStruct;

   sitename Site;
   pathname Path;
   int      i = 0;
   int      c;
   char     Line[LINELEN+1];
   header *HeaderInfo = &HeaderStruct;

   /* Make sure file is positioned at top */

   rewind (MsgFileFd);

   /* Read first header to get priority and CC flag */

   /* read in the priority string */
   GetLine (Line, MsgFileFd);

   /* file is now positioned at the start of the destpath line */

   if (Line[0] != HEADERLINE)
      {
       WriteLog ("GetMsgHead: Message header is invalid.", "", "", "");
       return ((header *)NULL);  /* Try to recover */
      }

   HeaderInfo -> Priority = atoi (Line+1);  /* convert priority to int */

   if ((strlen(Line) > CCPOS) && (Line[CCPOS] == CCTYPE))
           HeaderInfo -> CCFlag = TRUE;  /* This is a courtesy copy */
   else
           HeaderInfo -> CCFlag = FALSE;

   /* Read all the previously tried sites */

   fscanf (MsgFileFd, "%*c%s\n", Path);     /* read the path       */
   StripMe (Path, Site);                    /* get the first site  */
   if (HeaderInfo -> DownSites [i] != NULL)
      {
       free (HeaderInfo -> DownSites [i]);
       HeaderInfo -> DownSites [i] = NULL;
```

```
      }
   HeaderInfo -> DownSites [i] = malloc (strlen (Site) + 1);
   strcpy (HeaderInfo -> DownSites [i++], Site);  /* copy the site */
   SkipEOL (MsgFileFd);  /* Ignore the trace-back (full) path */

   while ((fgetc (MsgFileFd) == HEADERLINE) && (i < MAXDOWNSITES))
       {
        SkipEOL (MsgFileFd);  /* ignore the priority line */
        fscanf (MsgFileFd, "%*c%s\n", Path);     /* read the path       */
        StripMe (Path, Site);                    /* get the first site  */
        if (HeaderInfo -> DownSites [i] != NULL)
          {
           free (HeaderInfo -> DownSites [i]);
           HeaderInfo -> DownSites [i] = NULL;
          }
        HeaderInfo -> DownSites [i] = malloc (strlen (Site) + 1);
        strcpy (HeaderInfo -> DownSites [i++], Site);  /* copy the site */
        SkipEOL (MsgFileFd);  /* Ignore the trace-back (full) path */
       }

   while ((HeaderInfo->DownSites [i] != NULL) && (i <= MAXDOWNSITES))
          {
           free (HeaderInfo -> DownSites [i]);
           HeaderInfo -> DownSites [i++] = NULL;
          }

   /* Save the last site-name as the original destination */

   HeaderInfo -> DestSite = malloc (strlen (Path) + 1);
   strcpy (HeaderInfo -> DestSite, Path);

   if (DebugLevel)
      WriteLog ("GetMsgHead: DestPath was", HeaderInfo->DestSite, "", "");

   /* Return with the whole structure */

   return (HeaderInfo);
}
```

```
#include "net.h"

/*
   HandleError - Save the original message in an error queue for messages
                 that were improperly formatted or which were for some
                 other reason not completely processed.
*/

int HandleError (MsgFileName)

char *MsgFileName;   /* Name of message file in MSGPROCQ */

{
   pathname MsgPathName;      /* Complete pathname of the original message */
   pathname ErrPathName;      /* Complete pathname of the error queue entry */
   int      Result;           /* Result code from link & unlink */

   sprintf (MsgPathName, "%s/%s", MSGPROCQ, MsgFileName);
   sprintf (ErrPathName, "%s/%s", ERRORQ, MsgFileName);

   Result = link (MsgPathName, ErrPathName);  /* Move the file */
   if (Result == ERR)         /* Uh-oh.. It didn't work. */
      {
       WriteLog ("HandleError: Cannot move ", MsgPathName, " to ",
                 ErrPathName);
       return (ERR);  /* Try to recover, even now */
      }

   Result = unlink (MsgPathName);  /* Remove old pointer */
   if (Result == ERR)         /* Couldn't remove old file. */
      {
       WriteLog ( "HandleError: Cannot remove ", MsgPathName, "", "");
      }

   WriteLog ("HandleError: Message placed in ", ErrPathName, "", "");

   return (GOOD);  /* Good result */
}
```

```
#include "net.h"

#define YEARSECS 31557600;
#define DAYSECS  86400;
#define HOURSECS 3600;
#define MINSECS  60;

/*
   HashTime() takes the given date string in the form  "MM/DD/YY hh:mm:ss"
   and returns an integer value representing the number of seconds
   elapsed since the beginning of the "epoch". The "epoch" is defined on
   most UNIX systems to be 00:00:00 1 January 1970 GMT.
*/

/*  days per month  J   F   M   A   M   J   J   A   S   O   N   D      */
int Marray[12] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };




long HashTime (DateStr)

{
   int i;
   int Month, Day, Year, Hour, Minute, Second;
   long Time;

   i = sscanf (DateStr, "%d/%d/%d %d:%d:%d",
           &Month, &Day, &Year, &Hour, &Minute, &Second);

   /* make sure date is valid */
   if (i != 6)
      return ((long) ERR);
   if (Year < 0)
      return ((long) ERR);
   if ((Month > 12) || (Month < 1))
      return ((long) ERR);
   if ((Day > 31) || (Day < 1))
      return ((long) ERR);
   if ((Hour < 0) || (Hour > 23))
      return ((long) ERR);
   if ((Minute < 0) || (Minute > 59))
      return ((long) ERR);
   if ((Second < 0) || (Second > 59))
      return ((long) ERR);

   --Day;       /* account for non-zero offset */
   --Month;     /* account for non-zero offset */

   /* account for seconds since beginning of month */
   Time =  (long)Second;
   Time += (long)Minute * MINSECS;
   Time += (long)Hour * HOURSECS;
   Time += (long)Day * DAYSECS;
```

```
    /* account for preceeding months of this year */
    for (i = 0; i < Month ; i++)
        Time += (long)(Marray[i]) * DAYSECS;  /* days/month * seconds/day */

    if ((Month > 2) && (Year % 4 == 0))  /* if after Feb. in a leap year */
        Time += DAYSECS;                 /* add the extra day */

    Time += (long)(Year - 70) * YEARSECS;

    return (Time);
}
```

```
#include "net.h"

#define PRIORPOS 1
#define PRIORCUT 2

int High (Priority)

char *Priority;

{
   return ( atoi(Priority+PRIORPOS) < PRIORCUT);
}
```

```
#include "net.h"

/*
        Hold - See if message MsgName was destined for an inactive
               site. If so, put it in a holding queue. If not, return ERR.
*/

int Hold (MsgName)

char *MsgName;

{
   FILE *SiteTableFd;
   FILE *Fd;
   header *MsgHeader;
   header *GetMsgHead();
   register int n=0;
   register int Found = FALSE;
   register int c;
   pathname Command;
   sitename Site;
   sitename TempBuf;

   /* open the message file */

   if ((Fd = FileOpen(MsgName, MSGPROCQ, "r")) == NULL)
      {
       WriteLog ("Hold: can't open", MsgName, "for processing", "");
       return (ERR);
      }

   /* read in the message header */

   /* Get message's header info. */
   if ((MsgHeader = GetMsgHead (Fd)) == (header *)NULL)
      {
       WriteLog ("Hold: can't read header in", MsgName, "", "");
       fclose (Fd);
       return (ERR);  /* Try to recover */
      }

   fclose (Fd);

   /* get most recently tried site */
   strcpy (Site, MsgHeader->DownSites[0]);

   /* see if the destination site was inactive */

   if ((Lock (SITETABLE)) == ERR)
      {
       WriteLog ("Hold: Can't lock", SITETABLE, "-", "goodbye!");
       UnLock (MSGPROCQ); /* Don't leave my lock file sitting around */
       exit (1);
      }

   if ((SiteTableFd = fopen (SITETABLE, "r")) == NULL)
      {
       WriteLog ("Hold: Can't open", SITETABLE, "", "");
       UnLock (SITETABLE);
       return (ERR); /*indicate failure */
```

```
    }

  /* check all site entries starting with a DELCHAR */
  while (((c = getc (SiteTableFd)) != EOF) && (!Found))
    if (c == DELCHAR)  /* it is a deactivated site entry */
      {
       fscanf (SiteTableFd, "%s", TempBuf);  /* get a site name */
       if (EQUALS (TempBuf, Site))  /* it is the one we seek */
       Found = TRUE;
      }
    else
      SkipEOL (SiteTableFd);   /* next line */

  fclose (SiteTableFd);
  UnLock (SITETABLE);

  if (!Found)
    {
     WriteLog ("Hold:", Site, "was not found in site table", "");
     return (ERR);
    }

  /* if the queue does not exist, create holding queue */

  sprintf (Command, "mkdir %s 2>/dev/null", Site);
  if (system (Command) == 0)  /* has no effect if queue already exists */
    WriteLog ("Hold: created holding queue for", Site, "", "");

  /* move the message to the holding queue */

  sprintf (Command, "mv %s/%s %s 2>/dev/null", MSGPROCQ, MsgName, Site);
  if (system (Command) != 0) /* then the command failed */
    {
     WriteLog ("Hold: can't move", MsgName, "to holding queue", Site);
     return (ERR);
    }

  WriteLog ("Hold:", MsgName, "saved in holding queue for", Site);

  return (GOOD); /* all is well */

}
```

```
#include "net.h"

/*
  Create a "courtesy-copy" of the message file pointed to by
  MsgFileFd by adding the courtesy-copy flag to the file
  header and requeueing for later transmission to the downed site.
  Leave the file in MSGPROCQ for disposition.
*/

int MakeCC (MsgFileFd, NewSiteName, MsgPriority)

FILE *MsgFileFd;     /* Message file descriptor */
char *NewSiteName;   /* Name of site where message will now be processed */
int  MsgPriority;    /* Priority of message for new header */

{
   FILE     *CCMsgFd;        /* New file descriptor */
   filename CCMsgName;       /* New file name */
   char     MsgLine [LINELEN+1];  /* Storage for copying */

   /* Create a new message */
   if ((CCMsgFd = NewFile (CCMsgName, CCTYPE, MSGPROCQ)) == NULL)
      {
       fprintf (stderr, "MakeCC: Cannot create a new feeder queue entry.\n");
       return (ERR);  /* Try to recover */
      }

   rewind (MsgFileFd);   /* Reposition the original file at beginning */

   SkipEOL (MsgFileFd);  /* Skip the first line since we're rewriting it */

   fprintf (CCMsgFd, ">%d C %s\n", MsgPriority, NewSiteName);  /* Make C.C. */

   /* Copy the remainder of the file, word-for-word. */

   while ((fgets (MsgLine, LINELEN, MsgFileFd)) != NULL)   /* Loop to EOF */
      fputs (MsgLine, CCMsgFd);  /* Copy the line */

   fclose (CCMsgFd);        /* Close the file now that it's ready */

   /* Rename the message file so that it appears in the queue */

   FileNQ (CCMsgName+1, MSGPROCQ);   /* put message in xcvr input queue */

   rewind (MsgFileFd);      /* Put it back at the beginning */

   return (GOOD);           /* No error */
}
```

```
#include "net.h"

extern DebugLevel;

int MPMail (MsgName)

char *MsgName;          /* message file name - assume message in MSGPROCQ */

{

   FILE     *MsgFd;     /* message file descriptor */
   char     *UserName;  /* user name to receive mail message */
   pathname Path;       /* message destination path */
   pathname Command;    /* buffer to build UNIX command into */

   if ((MsgFd = FileOpen (MsgName, MSGPROCQ, "r")) == NULL)
      {
       WriteLog ("MPMail: can't open", MsgName, "", "");
       return (ERR);
      }

   SkipEOL (MsgFd); /* skip the priority line */

   fscanf (MsgFd, "%*c%s", Path);          /* read in the message path */
   UserName = StripMe (Path, Command);     /* point to user name */
   if (UserName == NULL)                   /* path just has a username */
      UserName = Path;

   fclose (MsgFd); /* be sure to close the file */

   if (DebugLevel)
      WriteLog ("MPMail: user name is", UserName, "", "");

   if ( ! ValidUser(UserName) )    /* if user is not registered in /etc/passwd */
      {
       WriteLog ("MPMail: unregistered user", UserName, "", "");
       return (ERR);
      }

   /* build UNIX command to mail message to user */
   sprintf (Command, "mail %s <%s/%s 2>/dev/null",
            UserName, MSGPROCQ, MsgName);

   if (system(Command) != 0)    /* zero return code means it worked */
      {
       WriteLog ("MPMail: can't invoke UNIX mail to", UserName,
                 "for", MsgName);
       /* later we may want to have the dead.letter file removed here */
       return (ERR);
      }

   WriteLog ("MPMail:", MsgName, "mailed to", UserName);

   return (GOOD);
```

```
#include "net.h"
#include <signal.h>

#define ABORTFILE ".abort"

pathname LogFile;       /* global LogFile for msgproc routines */
int DebugLevel = 0;     /* runtime debug level (0 = normal) */
int Archiving = TRUE;   /* Flag indicating that archiving is to take place */

main (argc, argv)         /* Message Processor main program */

int argc;
char ** argv;

/*
   This program monitors the MSGPROCQ directory and processes any files
   found there according to the file type. The file type is given by the
   first letter of the filename. All incoming messages, whether generated
   locally or recieved from remote systems, are placed in MSGPROCQ.

   The program is invoked with the name of the main directory (MasterQueue)
   and will run until there are no more files in the MSGPROCQ in that
   directory. If it is invoked with the "-" option, it will run in the
   background indefinitely, checking MSGPROCQ every 2 seconds.
*/

{
pathname MasterQueue;
pathname NewName, OldName;
char *NextMsgName;    /* Next message in Message Processor input queue */
int  Result;          /* Result from lower-level routines */
int  Forever=FALSE;   /* if the "-" argument is given, loop forever */
char Level[3];        /* string version of debug level */
FILE *ParamFileFp;    /* Parameter file pointer */
char Key[20];
int  Value;
int  PollDelay = 30; /* interval (in seconds) between scans */

void ShutDown ();     /* Routine to abort system */

signal (SIGTERM, ShutDown); /* Set signal handler for TERM signal */

umask (UMASK);

sprintf (LogFile, "log/%s.log", MSGPROC); /* set the global LogFile */

/* validate and parse arguments */
if ((argc < 2) || (argc > 4))
      usage (argv[0]);

while (--argc > 0)
   switch (argv[argc][0])
      {
       case '-': {
                  if (strlen(argv[argc]) == 1)
                     Forever = TRUE; /* continuous operation */
                  else
                     if (argv[argc][1] == 'd')
                        {
                         sscanf (argv[argc]+2, "%d", &DebugLevel);
```

```
                        sprintf (Level,"%d", DebugLevel);
                       }
                    else
                       {
                        fprintf (stderr, "illegal option: %s\n", argv[argc]);
                        usage (argv[0]);
                       }
                 break;
                }

       default : strcpy(MasterQueue,argv[argc]); /* get working directory*/
      }

/* set working directory */
if ((chdir (MasterQueue)) != 0)
   {
    fprintf (stderr, "invalid directory\n");
    usage (argv[0]);
   }

if (Lock(MSGPROCQ) == ERR)
   {
    WriteLog ("MsgProc: can't lock", MSGPROCQ, "-", "Goodbye!");
    exit (ERR);
   }

unlink (ABORTFILE);  /* Remove ".abort" before running */

if (DebugLevel)
   WriteLog ("MsgProc:", "system debug level is", Level, "");

   if ((ParamFileFp = fopen (PARAMFILE, "r")) != NULL)
      {
       fscanf (ParamFileFp, "%s %d\n", Key, &Value);
       while ((!feof (ParamFileFp)) && (!ferror (ParamFileFp)))
          {
           if (EQUALS (Key, "msgprocpoll"))
              PollDelay = Value;
           if (EQUALS (Key, "archiving"))
              Archiving = Value;

           fscanf (ParamFileFp, "%s %d\n", Key, &Value);
          }

       fclose (ParamFileFp);
      }

if (Forever)
   WriteLog ("MsgProc: activated in Scanner mode in", MasterQueue, "", "");
else
   WriteLog ("MsgProc: activated in One-Pass mode in", MasterQueue, "", "");

do    /* non-terminating process loops to scan input queue */
  {
   if (Abort (MasterQueue))  /* Check for need to abort */
      ShutDown ();

   while ((NextMsgName = DeQueue (MSGPROCQ)) != NULL)
      {
       if (DebugLevel)
          WriteLog ("MsgProc: Next message is ", NextMsgName, "", "");
```

```
        switch (NextMsgName[0])  /* process messages by type indicator */
           {
            case ADMINTYPE :
                       /* Process administrative messages */
                       Result = Administer (NextMsgName);
                       break;

            case SERVERTYPE :
                       /* give to JINTACCS message server */
                       sprintf (NewName, "%s/%s", SERVERQ, NextMsgName);
                       sprintf (OldName, "%s/%s", MSGPROCQ, NextMsgName);
                       Result = FRename (OldName, NewName);
                       break;

            case REROUTETYPE :
                       /* Perform high-level rerouting of the message */
                       Result = ReRoute (NextMsgName);
                       break;

            case CCTYPE :
            case ROUTINETYPE :
            case PRIORTYPE :
            case MESSAGETYPE :
                       /* Process normal messages */
                       Result = Process (NextMsgName);
                       break;

            case USERTYPE :
                       /* mail message to local user */
                       Result = MPMail (NextMsgName);
                       break;

            case NOPATHTYPE :
                       /* message destination not found by ValidSite */
                       Result = Hold (NextMsgName);
                       break;

            case NOHEADTYPE :
                       /* message header was invalid */
                       Result = ERR;
                       break;

            case NAKTYPE :
                       /* message received with NAK */
                       Result = ERR;
                       break;

            case ERRORTYPE :
                       /* message is mangled */
                       Result = ERR;
                       break;

            default  : /*  does not match naming conventions */
                       WriteLog ("MsgProc: unknown message type: ",
                                 NextMsgName, "", "");
                       Result = ERR;
                       break;

          }  /* end of switch cases */
```

```
        switch (Result)
           {
            case GOOD: Archive (NextMsgName, MSGPROCQ);
                       Remove (NextMsgName, MSGPROCQ);
                       break;
            case ERR :
            default  : HandleError (NextMsgName);
           }

        if (Abort (MasterQueue))  /* Check for need to abort */
           ShutDown ();

       } /* got on to next message to be processed */

    if (Forever)    /* sleep between scans */
       sleep (PollDelay);

  } while (Forever);    /* Continue scanning queue for more arrivals */

UnLock (MSGPROCQ);

WriteLog ("MsgProc: normal termination.", "", "", "");

exit (GOOD);

}
```

```
int usage (Name)
char *Name;
{
   fprintf (stderr, "usage: %s directory [-]\n", Name);
   exit (ERR);
}
```

```
void ShutDown ()

{
   WriteLog ("ShutDown:", "operator requested system shutdown", "", "");
   UnLock (MSGPROCQ);
   exit (0);
}
```

```
#include "net.h"

extern int DebugLevel;   /* Runtime debug level (0 = normal) */
extern pathname LogFile; /* Name of system log */

int Process (MsgFile)

char *MsgFile;

{
   FILE *Fd;
   char Priority [LINELEN+1];
   char Path [LINELEN+1];
   char FullPath [LINELEN+1];
   sitename FirstSite;
   sitename NextSite;
   char *NewPath;
   pathname FileName;
   pathname Queue;
   int  FixPath;
   int  Type = MESSAGETYPE;
   FILE *MsgFileFd;
   register int c;
   char *ThisName;  /* Pointer to storage for name of this system */
   site *SiteEntry;

   ThisName = MyName ();  /* Get site's name */

   /* open the message file for reading */
   if ((Fd = FileOpen(MsgFile, MSGPROCQ, "r")) == NULL)
      {
       WriteLog ("Process: can't open", MsgFile, "", "");
       return (ERR);
      }

   if (DebugLevel)
      WriteLog ("Process: processing file", MsgFile, "", "");

   /* get message header components for processing */
   GetLine (Priority, Fd);
   GetLine (Path, Fd);
   GetLine (FullPath, Fd);

   if (DebugLevel)
      WriteLog ("Process: path is:", Path, "", "");

   /* get first site from path and set new path to remainder */
   NewPath = StripMe (Path+1, FirstSite);

   /* assume the message is for me or a user at my site */
   FixPath = FALSE;                /* don't remove myname from path */
   strcpy (NextSite, FirstSite);   /* assume message stays here     */

   /*
     If it is a message for me to relay to another site, then we must
     remove myname from the path and get the name of the next site for
     validation and message disposition.

     If FirstSite != MyName then the message isn't for me, and we
     simply forward it to the correct site. The variables NextSite, FixPath,
```

```
    and FirstSite are already set correctly by the code above.
  */

  if ((NewPath != NULL) && (EQUALS(FirstSite, ThisName)))
        {
         FixPath = TRUE;              /* remove myname from the path       */
         StripMe (NewPath, NextSite); /* set NextSite to next site in path */
        }

  free (ThisName);

  strcpy (Queue, NextSite);
  if (MsgFile[0] == CCTYPE)  /* leave CCs alone, otherwise set priority */
     Type = CCTYPE;
  else
     if (High(Priority))  /* priority messages get special treatment */
        {
         Type = PRIORTYPE;
         strcpy (Queue, PRIORQ);
        }
     else
         Type = ROUTINETYPE;

  /*
    Now we validate the destination site and place the message in its
    proper queue.
  */

  if ((SiteEntry = ValidSite(NextSite)) == NULL)  /* For user or program? */
     {
      FixPath = FALSE;
      strcpy (Queue, MSGPROCQ); /* give it to MP to process */
      if (ValidUser(NextSite))   /* it is for a known user */
         {
          if (EQUALS(NextSite, NETADMIN))
             Type = ADMINTYPE;   /* network administrative message */
          else if (EQUALS(NextSite, SERVER))
             Type = SERVERTYPE;
          else
             Type = USERTYPE;   /* user mail message */
         }
      else    /* it is for an unknown site or user */
         {
          Type = NOPATHTYPE;
          WriteLog ("Process: unknown site or user:", NextSite, "", "");
         }
     }
  else   /* May be for second (emulated) system on local machine */
     if (SiteEntry->SysType == EMULATED)
        sprintf (Queue, "%s/%s", SiteEntry->PhoneNum[0], MSGPROCQ);

  /* build filename of correct type */
  sprintf (FileName, ".%c%s", Type, MsgFile+1);

  if (DebugLevel)
     WriteLog ("Process: creating file", FileName+1, "in", Queue);

  /* copy message to that file in proper queue */
  if ((MsgFileFd = FileOpen(FileName, Queue, "w")) == NULL)
     {
      WriteLog ("Process: can't create", FileName, "in", Queue);
```

```
        fclose (Fd);
        return (ERR);
       }

   /* copy the message header into new file */
   fprintf (MsgFileFd, "%s\n", Priority);
   if (FixPath)
     {
      /* move myname from start of Path to end of FullPath */
      fprintf (MsgFileFd, "%c%s\n", HEADERLINE, NewPath);
      fprintf (MsgFileFd, "%s%c%s\n", FullPath, SEPCHAR, FirstSite);
     }
   else
     {
      /* copy Path and FullPath verbatim */
      fprintf (MsgFileFd, "%s\n", Path);
      fprintf (MsgFileFd, "%s\n", FullPath);
     }

   /* copy the rest of the message */
   while (( c = getc (Fd)) != EOF)
       putc (c, MsgFileFd);

   fclose (MsgFileFd);
   fclose (Fd);

   FileNQ (FileName+1, Queue); /* make the file visible in the target queue */

   if (DebugLevel)
      WriteLog ("Process:", MsgFile, "has been processed", "");

   if (SiteEntry->SysType == EMULATED)  /* Take care of logging in this case */
      {
       sprintf (LogFile, "log/%s.log", SiteEntry->SiteName);
       WriteLog ("Process:", FileName+1, "sent to emulated system", "OK");
       sprintf (LogFile, "log/%s.log", MSGPROC);
      }

   return (GOOD);
}
```

```
#include "net.h"

extern int DebugLevel;

int PutMsg (MsgName)

char *MsgName;

{
   FILE     *Fd;
   pathname Path;
   pathname Priority;
   pathname OldName;
   pathname NewName;
   sitename Site;

   /* open the message file and read the destination line */
   sprintf (OldName, "%s/%s", MSGPROCQ, MsgName);
   if ((Fd = fopen(OldName, "r")) == NULL)
      {
       WriteLog ("PutMsg: can't open", MsgName, "for processing", "");
       return (ERR);
      }

   GetLine (Priority, Fd);
   GetLine (Path, Fd);

   fclose (Fd);

   /* get the first site on the destination path */
   StripMe (Path+1, Site);

   /* move the message to that site's queue */
   sprintf (NewName, "%s/%s", Site, MsgName);
   if (FRename (OldName, NewName) == ERR)
      {
       WriteLog ("PutMsg: can't move", OldName, "to", NewName);
       return (ERR);
      }

   if (DebugLevel)
      WriteLog ("PutMsg: moved", OldName, "to", NewName);
   return (GOOD);

}
```

```
#include "net.h"
#include <sys/stat.h>
#include <sys/dir.h>

#define NFILES 96   /* Max files per directory */

extern int DebugLevel;

/*
   Recover() is an administrative routine used to scan the message archive
   directory for any messages sent to Site after the given Date. Site is a
   TACCNET site name. Date is an ascii string in the form "MM/DD/YY hh:mm:ss".
   Any messages found by Recover() will be linked into the system queue for
   Site so that they will be picked up by QMS and retransmitted.
*/

int Recover (MsgFileFd)
FILE *MsgFileFd;

{
   long HashTime(); /* convert string date to long int */
   long lseek();
   FILE *Fp;
   char Date[40];
   struct stat StatBuf;
   struct direct DirEntry;
   sitename Site;
   pathname QueuePath;
   pathname FilePath;
   pathname DestPath;
   pathname DestSite;
   filename *Entries;
   long BackUpDate;
   int DirFd;
   int i = 0;
   int Num = 0;
   int errcount = 0;

   while (getc(MsgFileFd) != ' ');    /* skip to start of parameter list */

   if (fscanf (MsgFileFd, "%s %[^\n]", Site, Date) != 2)
      {
       WriteLog ("Recover: error in command line, site =",
                 Site, "date =", Date);
       return (ERR);
      }

   WriteLog ("Recover: messages for", Site, "since", Date);

   /* convert date string to integer */
   if ((BackUpDate = HashTime (Date)) == ERR)
      {
       WriteLog ("Recover: backup date format error", "'", Date, "'");
       return (ERR);
      }

   if ((DirFd = open(ARCHIVEQ, O_RDONLY)) == ERR)   /* open directory */
      {
       WriteLog ("Recover:", "unable to open", ARCHIVEQ, "directory");
       return (ERR);
```

```
    }

if (lseek (DirFd, 32L, 0) == (long) ERR)   /* Skip . and .. entries */
    {
     close (DirFd);
     return (ERR);
    }

Entries = (filename *) malloc (NFILES * sizeof(filename));

/* read all the file names from the directory into an array */
while (read (DirFd, &DirEntry, sizeof (struct direct)) != 0)
    if ((DirEntry.d_ino != (ino_t) 0) && (DirEntry.d_name [0] != DOT))
        strcpy (Entries [Num++], DirEntry.d_name);   /* copy file name */

*Entries [Num] = '\0';   /* mark end of list with null string */
close (DirFd);

/* read directory listing from the archive and check all files on list */
for (i = 0; (*Entries[i] != '\0') && (i < Num); i++)
   {
    if (DebugLevel >2)
       WriteLog ("Recover: checking", Entries[i], "in", ARCHIVEQ);

    /* build the full pathname of the file */
    sprintf (FilePath, "%s/%s", ARCHIVEQ, Entries[i]);

    /* get creation date of file */
    if (stat (FilePath, &StatBuf) != GOOD)
       {
        WriteLog ("Recover: cannot stat", FilePath, "", "");
        if (++errcount > 10)
           {
            free (Entries);
            return (ERR);
           }
        continue;
       }

    /* if msg date is later than backup date, check destination */
    if (StatBuf.st_mtime >= BackUpDate)
       {
        if ((Fp = fopen(FilePath, "r")) == NULL)
           {
            WriteLog ("Recover: can't open", FilePath, "", "");
            if (++errcount > 10)
               {
                free (Entries);
                return (ERR);
               }
            continue; /* next iteration of while loop */
           }

        SkipEOL (Fp);  /* skip the priority line */

        fscanf (Fp, "%*c%s", DestPath); /* read the destination path */
        fclose (Fp);

        if (DebugLevel > 2)
           WriteLog ("Recover: destination path is", DestPath, "", "");
```

```
            StripMe (DestPath, DestSite);   /* get the name of the next site */

            if (EQUALS (DestSite, Site))    /* if msg is for the given site */
               {
                sprintf (QueuePath, "%s/%s", MSGPROCQ, Entries[i]);
                if (link (FilePath, QueuePath) == GOOD)
                  WriteLog ("Recover: queued", FilePath, "for", Site);
                else
                  WriteLog ("Recover: can't link", FilePath, "to", QueuePath);
               }
          }
      }

   free (Entries);
   return (GOOD);
}
```

```
#include "net.h"

extern int DebugLevel;

/*
   ReRoute - Perform the high-level rerouting of messages in the case
             that a site is down and a message needs to be sent to an
             alternate site.
*/

int ReRoute (MsgFileName)

char *MsgFileName;

{
   header *MsgHeader;       /* Message header information */
   header *GetMsgHead();  /*  function to get message header info */
   sitename AltSites [MAXALTSITES+1]; /* List of alt. sites for down site */
   sitename NewSite;                  /* Site chosen for rerouting */
   char   **BadSites;                 /* List of bad sites */
   FILE   *MsgFileFd;                 /* Pointer to original message file */
   int    Result;                     /* Result code from lower-lever routines */
   register int Done = FALSE;
   register int Found = FALSE;
   register int i = 0;
   register int j = 0;
   register int x=0;

   if ((MsgFileFd = FileOpen (MsgFileName, MSGPROCQ, "r")) == NULL)
    {
     WriteLog ("ReRoute: Can't open message file ", MsgFileName,"","");
     return (ERR);  /* Try to recover */
    }

/* Get the header information, returning to higher level if an error occurred */

   if ((MsgHeader = GetMsgHead (MsgFileFd)) == (header *)NULL)
      return (ERR);  /* Try to recover */

/* Get alternate sites and verify that no error occurred reading the file */

   Result = GetAltSites (MsgHeader->DestSite, AltSites);
   if (Result == ERR)
      return (ERR);  /* Try to recover */

   /* Scan list of alternate sites for a new destination, taking into
      consideration those sites already attempted in the rerouting.   */

   /* Point to the previously-attempted (assumedly down) sites */

   BadSites = MsgHeader -> DownSites;

   if (DebugLevel > 2)
      {
       for (x=0; AltSites[x] != NULL; x++)
           fprintf (stderr, "A%.ld: %s\n", x, AltSites[x]);
       for (x=0; BadSites[x] != NULL; x++)
           fprintf (stderr, "B%.ld: %s\n", x, BadSites[x]);
      }
```

```
   while ((AltSites [i] != NULL) && (!Done))  /* Prevent message looping */
      {
       while ((BadSites [j] != NULL) && (!Found))
          {
           if (DebugLevel > 2)
              WriteLog ("GetAltSites: comparing",
                        AltSites[i], "with", BadSites[j]);
           if (EQUALS (AltSites[i], BadSites[j]))
               Found = TRUE;
           else
               j++;
          }
       if (!Found)    /* We didn't find the site in the list of down sites */
          {
           Done = TRUE;
           WriteLog ("ReRoute: will forward to", AltSites[i], "", "");
          }
       else           /* The site was down, try another alternate site */
          {
           if (DebugLevel)
              WriteLog ("ReRoute:", AltSites[i], "was down - try again", "");
           Found = FALSE;
           j = 0;
           i++;
          }
      }

   if (!Done)    /* We didn't find a good alternate site */
      {
       WriteLog ("ReRoute: Could not find a working alternate site.",
                 "","","");
       return (ERR);
      }

   /* Otherwise, we found a good alternate site, so forge ahead. */

   strcpy (NewSite, AltSites [i]);

   /* Forward the message to NewSite */

   Result = Forward (MsgFileFd, NewSite, MsgHeader -> Priority);
   if (Result == ERR)
      {
       WriteLog ("ReRoute: Could not construct message to ", NewSite,
                 "for", MsgHeader -> DestSite);
       return (ERR);  /* Recover by ignoring but saving this message. */
      }

   /* Send a Courtesy Copy for the downed site */

   Result = MakeCC (MsgFileFd, NewSite, MsgHeader -> Priority);
   if (Result == ERR)
      {
       WriteLog ("ReRoute: Could not construct courtesy copy for ",
                 NewSite, "", "");
       return (ERR);  /* Recover by ignoring but saving this message. */
      }

   fclose (MsgFileFd);

   return (GOOD);  /* No errors */
```

```
}
#include "net.h"

main ()

{
   register int c;

   while ((c = getc(stdin)) == HEADERLINE)
      SkipEOL (stdin);

   putc (c, stdout);

   while ((c = getc(stdin)) != EOF)
      putc (c, stdout);
}
```

```
#include "net.h"

/*
   Command line format: save <filename>

   Read a filename from the given message file stream and save the body of
   the message in that file.
*/

int Save (Fd)

FILE *Fd;

{
   FILE *NewFd;
   pathname FileName;
   sitename Destination;
   register int c;

   /* skip to start of parameter */
   while (getc(Fd) != ' ');

   /* read the desired file name from the command line */
   fscanf (Fd, "%s", FileName);

   if ((NewFd = fopen(FileName, "w")) == NULL)    /* create the file */
      {
       WriteLog ("Save:", "cannot create", FileName, "");
       return (ERR);
      }

   SkipEOL (Fd);     /* skip to first line of text after command line */

   while ((c = getc(Fd)) != EOF)     /* copy the body of the message */
      putc (c, NewFd);

   fclose (NewFd);

   WriteLog ("Save:", FileName, "has been created", "");

   return (GOOD);
}
```

```
#include <stdio.h>
#include <sys/types.h>
#include <time.h>

long HashTime();

main (argc, argv)
int argc;
char **argv;

{
   long MyTime;
   long UTime;
   struct tm *TBuf;
   char TStr[80];
   struct tm *gmtime();
   char *ctime();
   long time();

   time (&UTime);
   printf ("Unix time: %ld, %s", UTime, ctime(&UTime));

   TBuf = gmtime(&UTime);
   sprintf (TStr, "%d/%d/%d %d:%d:%d", TBuf->tm_mon+1, TBuf->tm_mday,
            TBuf->tm_year, TBuf->tm_hour, TBuf->tm_min, TBuf->tm_sec);

   MyTime = HashTime(TStr);
   printf ("My time:   %ld, %s", MyTime, ctime(&MyTime));
}
```

```
#include "net.h"

/*
   Command line format: transfer <filename> <taccnet path>!<filename>

   Request a copy of the named file be built into a message and sent to
   a taccnet site via the given path, there to be saved under the given name.
   The desired file will be built into a "save" administrative message and
   sent to the given taccnet site for processing.  Use the priority of the
   request message.  If the file cannot be sent, send an error message.
   If no path is given, sent the file to the request origin site. If no
   target file name is given, use the source file name. Message will be
   addressed to Network Administrator process at destination site.
*/

int Transfer (Fd)

FILE *Fd;

{
   int Priority;
   register int c;
   FILE *PipeFd;
   FILE *FileFd;
   pathname Command;
   pathname FileName;       /* name of source file */
   pathname DestPath;       /* taccnet pathname of target file */
   pathname Origin;         /* request message origin site */
   pathname Address;        /* complete taccnet address for response message */
   char *SaveName;          /* pointer to name of target file */
   char *Destination;       /* pointer to destination site taccnet path */
   char *ThisName;          /* pointer to name of this site */

   ThisName = MyName ();    /* Get the name of this computer */

   /* skip to start of parameter list */
   while (getc(Fd) != ' ');

   /* read the desired file name and destination from the command line */
   DestPath[0] = '\0';  /* in case destination is omitted */
   fscanf (Fd, "%s %s", FileName, DestPath);

   /* rewind the file and read the message priority and origin site */
   rewind (Fd);
   fscanf (Fd, "%*c%d", &Priority);
   SkipEOL (Fd);    /* skip to destination line */
   SkipEOL (Fd);    /* skip to origin line      */
   fscanf (Fd, "%*c%s", Origin);

   /* set destination path and target file name */
   if (strlen(DestPath) == 0)  /* then destination was omitted */
      {
       SaveName = FileName;     /* default target name = source name */
       Destination = Origin;    /* default destination is request origin */
      }
   else      /* set target file name and destination path */
      if ((SaveName = strrchr(DestPath, SEPCHAR)) != NULL)
         {
          *SaveName = '\0'; /* separate destination path and target filename */
          ++SaveName;       /* point to start of file name */
```

```
         Destination = DestPath;   /* set pointer to destination path */
        }
      else    /* only a target filename was supplied */
        {
         Destination = Origin;    /* default destination is request origin */
         SaveName = DestPath;     /* use target name supplied */
        }

   sprintf (Address, "%s%c%s", Destination, SEPCHAR, NETADMIN);
   sprintf (Command, "%s/bin/%s %d %s 1>/dev/null 2>&1",
            MASTERQ, GENMSG, Priority, Address);
   if ((PipeFd = popen (Command, "w")) == NULL)
      {
       WriteLog ("Transfer:", "cannot open pipe to", Command, "");
       return (ERR);
      }

   /* make sure the desired file exists and is readable */
   if ((FileFd = fopen(FileName, "r")) == NULL)    /* open the file */
      {
       WriteLog ("Transfer:", "cannot access", FileName, "");
       fprintf (PipeFd, "*** could not access %s at site %s\n",
                FileName, ThisName);
       free (ThisName);
       if (pclose (PipeFd) != GOOD)
          {
           WriteLog ("Transfer:", "error executing", Command, "");
           return (ERR);
          }
       WriteLog ("Transfer:", "requesting site", Destination,
                 "has been notified");
       return (GOOD);
      }

   fprintf (PipeFd, "save %s\n", SaveName);
   while ((c = getc(FileFd)) != EOF)     /* copy the body of the message */
      putc (c, PipeFd);

   fclose (FileFd);

   if (pclose (PipeFd) != GOOD)
      {
       WriteLog ("Transfer:", "error executing", Command, "");
       return (ERR);
      }

   WriteLog ("Transfer:", FileName, "will be sent to", Destination);

   return (GOOD);
}
```

```
#include "net.h"

/*      ValidUser - returns FALSE if UserName is not defined in PASSWDFILE
                    and !FALSE ( not necessarily = TRUE) if it is defined.
*/

int ValidUser (UserName)

char *UserName;

{

   char Name [10];    /* assumes unix usernames are 9 characters or less */
   char Command [80];
   FILE *Pipe;

   sprintf (Command, "fgrep %s %s", UserName, PASSWDFILE);

   if ((Pipe = popen (Command, "r")) == NULL)
      {
       WriteLog ("ValidUser: can't run fgrep on", PASSWDFILE, "", "");
       return (FALSE);
      }

   fscanf (Pipe, "%[^:]%*s", Name);    /* read the username field only */

   pclose (Pipe);

   return (EQUALS(UserName, Name));

}
```

GENMSG

This section contains the functions used only by the Message Generator program (GENMSG).

```
genmsg: genmsg.o validsite.o validpath.o myname.o filenq.o\
      writelog.o datetime.o newfile.o fileopen.o lockfile.o\
      readsite.o stripme.o
        cc -O -o genmsg genmsg.o validpath.o myname.o writelog.o\
      datetime.o newfile.o filenq.o validsite.o fileopen.o lockfile.o\
      readsite.o stripme.o
        strip genmsg

genmsg.o: net.h genmsg.c
        cc -c -O genmsg.c

validpath.o: net.h validpath.c
        cc -c -O validpath.c
```

```
#include "net.h"

#define  TEMPFILEHEAD ",gen"
#define  MAXSITES 20

pathname LogFile; /* global LogFile for genmsg routines */

/* GenMsg is a program to generate messages for transmission by qms.
   It expects a priority as parameter 1 followed by a list of destinations.
   There must be at least 1 destination given.

   It will read the message body from stdin and build a message file
   or files which will be placed in the message processor queue.
   Stdin may be redirected to read from file or pipe. Input may be
   text or binary data.

   Multiple destination paths or sites may be specified on the command
   line. They should be separated by one or more spaces. All messages will
   be given the same priority.

   The destination may be given as a path alias, an absolute path, or a
   path alias with an absolute path appended. The last token in the path
   may be a user id at the target site. If so, the message will be mailed
   to that user upon arrival at the site.

   destination ::= <alias>[!<path>] | <site>[!<path>]
   path ::= <site>[!<path>] | <site>!<user>
   site ::= a network node name defined in the site table
   user ::= a valid user id on the target site
*/

main (argc, argv)
int argc;
char **argv;

{
   FILE     *TermFd;
   FILE     *TmpFileFd;
   filename TmpFileName;
   pathname Path;
   char     *SiteList [MAXSITES+1];
   char     *PathList [MAXSITES+1];
   char     Priority [2+1];
   register int Ch;
   register int j,k;
   int      MakeMessage();
   char     *BuildPath ();

   umask (UMASK);

   /* set log file for WriteLog() */
   sprintf (LogFile, "log/%s.log", GENMSG);

   /* validate argument count */
   if (--argc < 2)     /* there are at least two arguments */
      usage (argv[0]);

   /* get and validate message priority */
   strcpy (Priority, argv[1]);
   if ((strlen(Priority) != 1) || (atoi(Priority) < 0) || (atoi(Priority) > 9))
```

```
      {
       fprintf (stderr, "invalid priority - must be in range 0-9\n");
       usage (argv[0]);
      }

   /* get list of destination sites */
   for (j=2 ; j <= argc ; j++)
      {
       SiteList[j-2] = malloc ( strlen(argv[j])+1 );
       strcpy (SiteList[j-2], argv[j]);  /* copy the site into the list */
      }
   SiteList[j-2] = NULL;  /* mark end of list */

   for (k = 0, j = 0; SiteList[j] != NULL; j++)
       PathList[k++] = BuildPath (SiteList[j]);

   PathList[k] = NULL; /* mark end of list */

   if (PathList[0] == NULL)
      {
       fprintf (stderr, "no valid paths specified\n");
       usage (argv[0]);
      }

   /* read the message into a temporary file */

   sprintf (TmpFileName, "%s%d", TEMPFILEHEAD, NOW);
   if ((TmpFileFd = fopen(TmpFileName, "w+")) != NULL)
      {
       if ((Ch = getc (stdin)) == EOF)  /* empty message not allowed */
          {
           fprintf (stderr, "Sorry, empty messages are not allowed.\n");
           fclose (TmpFileFd);
           if (unlink (TmpFileName) == ERR)  /* delete the temporary file */
              WriteLog ("GenMsg: can't unlink", TmpFileName, "", "");
           exit (ERR);
          }
       while (Ch != EOF)  /* copy text till end of file */
          {
           putc (Ch, TmpFileFd);
           Ch = getc (stdin);
          }
      }
   else
      {
       fprintf (stderr, "Can't allocate tmpfile for message input.");
       exit (ERR);
      }

   /* for each path in PathList, generate a message and enqueue it */

   for ( j = 0; PathList[j] != NULL; j++)
       if (!MakeMessage (Priority, PathList[j], TmpFileFd))
          fprintf (stderr, "can't generate message\n");

   fclose (TmpFileFd);

   if (unlink (TmpFileName) == ERR)  /* delete the temporary file */
      {
       WriteLog ("GenMsg: can't unlink", TmpFileName, "", "");
       exit(ERR);
```

```
      }

   exit(GOOD);

}
```

```
/* This function will generate a message for the site given in SiteName
   with the body of text contained in the stream Fd.
   Return value is TRUE if operation succeeds, FALSE otherwise.
*/

int MakeMessage (Priority, Path, Fd)

char *Priority;
char *Path;
FILE *Fd;

{
   FILE     *MsgFileFd;
   filename MsgFileName;
   register int Ch;
   char     *ThisName;  /* Pointer to name of this system */

   ThisName = MyName ();  /* Get this site's name */

   /* create a message file to hold the message */

   if ((MsgFileFd = NewFile (MsgFileName, MESSAGETYPE, MSGPROCQ)) == NULL)
      {
       fprintf (stderr, "GenMsg: Can't create %s\n", MsgFileName);
       return (FALSE);
      }

   /* write the message header into the message file */

   fprintf (MsgFileFd, ">%s\n>%s\n>%s\n", Priority, Path, ThisName);

   free (ThisName);

   rewind (Fd);                          /* rewind the message text */

   while ((Ch = getc (Fd)) != EOF)       /* copy the message text */
        putc (Ch, MsgFileFd);

   fclose (MsgFileFd);

   FileNQ (MsgFileName+1, MSGPROCQ); /* put message in proper system queue */

   return (TRUE);
}
```

```
/* This function will build a network path to the site given in SiteName.
   The SiteName may be a path alias, an absolute path, or an alias
   with an absolute path appended.
   Return value will be a pointer to the constructed path, or NULL if the
   path was invalid.
*/

char *BuildPath (SiteName)

char *SiteName;

{
   char     *RetVal;
   pathname DestPath;
   pathname Path;
   sitename FirstSite;


   /* validate or expand the first site on the path if possible */

    sscanf (SiteName, "%[^!]", FirstSite);  /* strip first site */

    if (ValidSite(FirstSite) == NULL)       /* undefined site */
      if  (ValidPath(FirstSite,Path))       /* expand path alias */
         sprintf (DestPath, "%s%s", Path, SiteName+(strlen(FirstSite)));
      else  /* FirstSite is not defines in Path or Site table */
         {
          /* if site and path are undefined, message may be for a user or */
          /* a deactivated site, so pass it on to msgproc for disposition */
         }
    else /* site is defined in Site table */
      strcpy (DestPath, SiteName);  /* first site is defined */

   RetVal = malloc ( strlen(DestPath) + 1 );
   strcpy (RetVal, DestPath);
   return (RetVal);
}



int usage (name)
char *name;
{
   fprintf (stderr, "usage: %s priority dest [dest ...]\n", name);
   exit (ERR);
```

```
#include "net.h"

int ValidPath (Site, Path)

char *Site;
char *Path;

{
   FILE *PathTableFd;
   char SomeSite [128];
   register int c;

   if ((PathTableFd = fopen (PATHTABLE, "r")) == NULL)
      {
       fprintf (stderr, "ValidPath: Can't open %s\n", PATHTABLE);
       exit (1);
      }

   getc (PathTableFd);                              /* skip first colon */

   do {
       fscanf (PathTableFd, "%s", SomeSite);        /* get a site */
       if (EQUALS (SomeSite, Site))
          {
           getc (PathTableFd);                      /* read NL   */
           fscanf (PathTableFd, "%s", Path);
           fclose (PathTableFd);
           return (TRUE);
          }
       else
          {
           c = getc (PathTableFd);
           while ((c != FIELDMARK) && (c != EOF))
               c = getc (PathTableFd);
          }

      } while (c != EOF);

   fclose (PathTableFd);
   return (FALSE);
}
```

SERVER

This section contains the functions used only by the Database Server program (SERVER).

```
build: build.o request.o startmsg.o datetime.o writelog.o\
          makepol.o makecas.o makesup.o realtime.o unitid.o\
          blsup.o putdtgm.o
       cc -O -o build build.o request.o startmsg.o datetime.o\
          writelog.o makepol.o makecas.o makesup.o realtime.o\
          unitid.o blsup.o putdtgm.o
       strip build

server: server.o request.o post.o remove.o dequeue.o getdir.o\
          writelog.o datetime.o lockfile.o fileopen.o frename.o\
          makepol.o makecas.o makesup.o postpol.o postcas.o\
          postsup.o putdtgm.o startmsg.o realtime.o unitid.o\
          blsup.o abort.o
       cc -O -o server server.o request.o post.o remove.o\
          dequeue.o getdir.o writelog.o datetime.o lockfile.o\
          makepol.o makecas.o makesup.o postpol.o postcas.o\
          postsup.o fileopen.o frename.o putdtgm.o startmsg.o\
          realtime.o unitid.o blsup.o abort.o
       strip server

build.o: server.h build.c
       cc -c -O build.c

server.o: server.c net.h
       cc -c -O server.c

post.o: post.c net.h
       cc -c -O post.c

request.o: request.c net.h
       cc -c -O request.c

makepol.o: makepol.c net.h
       cc -c -O makepol.c

makecas.o: makecas.c net.h
       cc -c -O makecas.c

makesup.o: makesup.c net.h
       cc -c -O makesup.c

postpol.o: postpol.c net.h
       cc -c -O postpol.c

postcas.o: postcas.c net.h
       cc -c -O postcas.c

postsup.o: postsup.c net.h
       cc -c -O postsup.c

putdtgm.o: server.h putdtgm.c
       cc -c -O putdtgm.c

startmsg.o: server.h startmsg.c
       cc -c -O startmsg.c

realtime.o: realtime.c
       cc -c -O realtime.c

unitid.o: unitid.c net.h server.h
```

```
	cc -c -O unitid.c

blsup.o: blsup.c
	cc -c -O blsup.c
```

```
/* program constants for Server system */

#define SERVER  "server"
#define AMS     "ams"
#define MSG     "msg"
#define REQ     "REQ"
#define MSGID   "MSGID"
#define EXER    "EXER"
#define OPER    "OPER"
#define S006    "S006"
#define S026    "S026"
#define S034    "S034"
#define SQL     "/usr/bin/sql"
#define DBLOAD  "/usr/bin/dbload"
#define POLLOC  "POLLOC"
#define CASSTATS "CASSTATS"
#define SHORTSUP "SHORTSUP"

/* work files for Server system */

#define DATAFILE ".data"
#define WORKFILE ".work"
#define UNITID   "unitid"

/* user functions for Server system */

int Request();      /* handle JINTACCS request messages - generate message    */
int Post();         /* hanlde JINTACCS data messages - post to C2 database    */
int PostPOL();      /* post POLLOC message S026 to C2 database                */
int PostCAS();      /* post CASSTATS message S006 to C2 database              */
int PostSUP();      /* post SHORTSUP message S034 to C2 database              */
int MakePOL();      /* generate POLLOC message S026 from C2 database          */
int MakeCAS();      /* generate CASSTATS message S006 from C2 database        */
int MakeSUP();      /* generate SHORTSUP message S034 from C2 database        */
int PutDTGM();      /* write DTGM date/time group set for JINTACCS message    */
int RealTime();     /* return current system date and time as integer values  */
FILE *StartMsg();   /* create file & write JINTACCS message initial main text */
char *UnitID();     /* return name of this army unit (C2 database id)         */
char *blsup();      /* suppress trailing blanks                               */
```

```
char *blsup (txt)
char *txt;

/* suppress trailing blanks in string pointed to by txt */
/* blank string becomes null string                     */

{
   register int i;

   /* point to last non-blank character */
   for (i = strlen(txt)-1; ( (i >= 0) && (txt[i] == ' ') ); i--);

   /* set following character to indicate end of string */
   txt[i+1] = '\0';

   /* return pointer to blank-suppressed string */
   return (txt);
}
```

```
#include "net.h"
#include "server.h"

pathname LogFile; /* global LogFile for server routines */

main (argc, argv)        /* Message Builder main program */

int argc;
char ** argv;

{
   char MsgType[10];
   char UnitId[21];
   char Priority[10];
   char DestPath[80];
   char Imt[2];

if (argc != 5)
   {
    fprintf (stderr, "usage: %s MsgType UnitId Priority DestPath\n", argv[0]);
    exit (ERR);
   }

strcpy (MsgType, argv[1]);  /* get message name */
strcpy (UnitId, argv[2]);  /* get unit id */
strcpy (Priority, argv[3]); /* get priority */
strcpy (DestPath, argv[4]); /* get destpath */

strcpy (Imt, "");  /* leave EXER|OPER set blank */

sprintf (LogFile, "../log/%s.log", SERVER);  /* use same log file as server */

/* request the message and exit - this unit is the originator */
exit (Request (MsgType, UnitId, Imt, DestPath, Priority));

}
```

```
#include "net.h"
#include "server.h"

int MakeCAS (DataFp, UnitId)

FILE *DataFp;
char *UnitId;

{
   WriteLog ("MakeCAS: don't know how to make", CASSTATS, "for", UnitId);
   return (ERR);
}
```

```
#define ENDQUERY '/'
#include "net.h"
#include "server.h"

/* MakePOL() - build an input file for AMS to create a POLLOC message */

int MakePOL (DataFp)

FILE *DataFp;

{
   FILE *WorkFp;
   char Command [80];
   int  Index;
   int  LastIndex = 0;
   char *Ptloc [20];    /* assume maximum of 20 POL locations */
   char Name [27];
   char Location [21];
   char FOL;
   long int  Quantity;
   char Unit [4];
   char Type [7];
   char QtyUnitType [20];  /* Combined quantity, unit, and type */
   char *Fuel [20];     /* assume at most 20 types of fuel */
   char *Oil [20];      /* 20 types of oil */
   char *Lube [20];     /* 20 types of lubricant */
   char **Ptr;          /* Temporary pointer */
   char Dash[2];        /* Filler character for empty slots */
   int  MaxLines = 0;   /* Total number of lines used for current index */
   int  c;
   int  i;

   /* write DTGM set to work file for AMS */
   PutDTGM (DataFp);

   /* execute database query for POLLOC, put results in file for processing */

   sprintf (Command, "%s %s > %s 2> /dev/null", SQL, POLLOC, WORKFILE);
   system (Command);

      /* {
       WriteLog ("MakePOL: database query failed", "", "", "");
       return (ERR);
      } */

   if ((WorkFp = fopen (WORKFILE, "r")) == NULL)
      {
       WriteLog ("MakePOL: can't open work file", "", "", "");
       return (ERR);
      }


   /* write 3KPOLLOC set to work file for AMS */


   for (i=0, c=getc(WorkFp); c != ENDQUERY; i++, c=getc(WorkFp))
      {
       Name[0] = (char) c;
       fscanf (WorkFp, "%[^,]%*c%s", Name+1, Location);
       blsup (Name);
```

```
        Ptloc[i] = malloc (strlen(Name)+1);
        strcpy (Ptloc[i], Name); /* save POL names for comparison below */
        /* write the Data Entry number, POL Name, and POL Location */
        fprintf (DataFp, "%.2d\n%s\n%s\n", i+1, Name, Location);
        SkipEOL (WorkFp);
       }

    Ptloc[i] = NULL;
    SkipEOL (WorkFp);  /* Skip to the next line */
    fprintf (DataFp, "//\n");  /* End this section in the data file */

    /* write 3KCLTHRE set to work file for AMS */

    strcpy (Dash, "-");
    for (i = 0; i < 20; i++)  /* Initialize slots to "-" (empty) */
        Fuel[i] = Oil[i] = Lube[i] = Dash;

    for (c = getc (WorkFp); c != EOF; c = getc (WorkFp))
       {
        Name[0] = (char) c;
        fscanf (WorkFp, "%[^,]%*c%[^,]%*c%c%*c%ld%*c%s",
                        Name+1, Type, &FOL, &Quantity, Unit);  /* Get data */

        sprintf (QtyUnitType, "%ld%s%s", Quantity, blsup (Unit), Type);

        SkipEOL (WorkFp);  /* Skip to the next line */

        blsup (Name);        /* suppress trailing blanks */
        for (Index=0, i=0; Ptloc[i] != NULL; i++) /* match the ptloc entry */
            if (EQUALS(Name, Ptloc[i]))
               Index = i+1; /* set the Data Entry value for the POL item */

        if (Index == 0)  /* We did not find that site name */
           {
            WriteLog ("MakePOL:", "invalid site name (", Name, ")");
            return (ERR);
           }

        if ((Index != LastIndex) && (LastIndex > 0))  /* See if flush needed */
           {
            for (i = 0; i < MaxLines+1; i++)  /* Flush previous buffers */
                {
                 fprintf (DataFp, "%.2d\n%s\n%s\n%s\n-\n",
                          LastIndex, Fuel[i], Oil[i], Lube[i]);  /* Write data */

                 if (Fuel[i] != Dash) free (Fuel[i]);  /* Free data only */
                 if (Oil[i]  != Dash) free (Oil[i]);   /*  if allocated  */
                 if (Lube[i] != Dash) free (Lube[i]);  /*   during loop  */

                 Fuel[i] = Oil[i] = Lube[i] = Dash;  /* Clear slots */
                }
            MaxLines = 0;  /* Reset counter for maximum lines */
           }

        LastIndex = Index;   /* Remember this index to note change */

        switch (FOL)  /* Determine next column based on fuel/oil/lube */
           {
            case 'F' : Ptr = (char **) Fuel;  /* Point to Fuel slot */
                       break;
            case 'O' : Ptr = (char **) Oil;   /* Point to Oil slot */
```

```
                    break;
          case 'L' : Ptr = (char **) Lube;  /* Point to Lubricant slot */
         }

      for (i = 0; (i < 20) && (Ptr[i] != Dash); i++)
          ;               /* Scan for first empty position in this column */

      Ptr[i] = malloc (strlen (QtyUnitType) + 1);  /* Make room for data */
      strcpy (Ptr[i], QtyUnitType);  /* Copy data into slot */
      if (i > MaxLines)
         MaxLines = i;    /* Keep track of maximum depth */
    }  /* End of 'for' loop - exit at EOF */

  for (i = 0; i < MaxLines+1; i++)  /* Flush final buffers */
      {
       fprintf (DataFp, "%.2d\n%s\n%s\n%s\n-\n",
                LastIndex, Fuel[i], Oil[i], Lube[i]);  /* Write data */

       if (Fuel[i] != Dash) free (Fuel[i]);  /* Free data only */
       if (Oil[i]  != Dash) free (Oil[i]);   /*  if allocated  */
       if (Lube[i] != Dash) free (Lube[i]);  /*      above     */
      }

  fprintf (DataFp, "//\n");

  /* write remaining input to tidy up */

  fprintf (DataFp, "//\n//\n-\n"); /* no AMPN, no RMKS, no DWNGRADE */

  fclose (WorkFp);

  return (GOOD);
}
```

```
#include "net.h"
#include "server.h"

/* MakeSUP() - build an input file for AMS to create a SHORTSUP message */

int MakeSUP (DataFp, ReqUnitId)

FILE *DataFp;
char *ReqUnitId;

{
   FILE *WorkFp;
   char Command [80];
   int  c;
   int  Count;
   char UnitId[81];
   char Item[21];
   char Model[21];
   char Quantity[11];
   char Unit [4];
   char OnReq;
   char ReqNum [21];

   /* write DTGM set to work file for AMS */
   PutDTGM (DataFp);

   /* execute database query for SHORTSUP, put results in file for processing */

   sprintf (Command, "%s %s > %s 2> /dev/null", SQL, SHORTSUP, WORKFILE);
   system (Command);

   if ((WorkFp = fopen (WORKFILE, "r")) == NULL)
      {
       WriteLog ("MakeSUP: can't open work file", "", "", "");
       return (ERR);
      }

   /* write UNITIDM set to work file for AMS */
   fprintf (DataFp, "%s\n", ReqUnitId);
   fprintf (DataFp, "//\n");  /* mark end of UNITIDTM set */

   /* write 6KSHTSUP set to work file for AMS */
   for (Count=0, c = getc (WorkFp); c != EOF; c = getc (WorkFp), Count++)
      {
       UnitId[0] = (char) c;
       fscanf (WorkFp,
               "%[^,]%*c%[^,]%*c%[^,]%*c%[^,]%*c%[^,]%*c%c%*c%s",
               UnitId+1, Item, Model, Quantity, Unit, &OnReq, ReqNum);

       blsup (UnitId);        /* suppress trailing blanks */

       if (EQUALS(UnitId, ReqUnitId))
          {
           fprintf (DataFp, "%s %s %s %s\n",
                    blsup(Quantity), blsup(Unit), blsup(Item), blsup(Model));
           if (OnReq == 'Y')
              fprintf (DataFp, "A1 %s\n", ReqNum);
           else
              fprintf (DataFp, "A2\n");
          }
```

```
        SkipEOL (WorkFp);  /* Skip to the next line */

       }  /* End of 'for' loop - exit at EOF */

    if (!Count)
       {
        WriteLog ("MakeSUP: database query failed,", "workfile empty", "", "");
        return (ERR);
       }

    fprintf (DataFp, "//\n");  /* mark end of columnar set */

    /* write key to comments field in AMPN set */

    fprintf (DataFp, "A1 DENOTES ITEM ON REQUISITION, NUMBER FOLLOWING\n");
    fprintf (DataFp, "A1 IS REQUISITION NUMBER.  A2 DENOTES ITEM NOT ON\n");
    fprintf (DataFp, "REQUISITION//\n");

    fprintf (DataFp, "//\n-\n"); /* no RMKS, no DWNGRADE */

    fclose (WorkFp);

    return (GOOD);
}
```

```
#include "net.h"
#include "server.h"

int Post (MsgType, Originator, MsgFd)

char *MsgType;
char *Originator;
FILE *MsgFd;

{
   register int Result = ERR;

   switch (MsgType[0])
      {
       case 'c' :
       case 'C' : /* CASSTATS - S006 */
                  Result = PostCAS (MsgFd);
                  break;

       case 'p' :
       case 'P' : /* POLLOC - S026 */
                  Result = PostPOL (MsgFd);
                  break;

       case 's' :
       case 'S' : /* SHORTSUP - S034 */
                  Result = PostSUP (MsgFd);
                  break;

       default  : WriteLog ("Post: don't know how to post", MsgType,
                            "message", "");
      }

   if (Result != GOOD)
      {
       WriteLog ("Post: can't post", MsgType, "for", Originator);
       return (ERR);
      }

   WriteLog ("Post: posted", MsgType, "from", Originator);

   return (GOOD);
}
```

```
#include "net.h"
#include "server.h"

int PostCAS (MsgFd)

FILE *MsgFd;

{
   char Command [80];       /* Command for running DBLOAD */
   char Name [25];          /* Name of unit being reported */
   char TempBuf [25];       /* Temporary name storage */
   int  LastLine = FALSE;   /* Set after last line read */
   char SSI_MOS [10];       /* military specialty code  */
   char Count  [4][10];     /* array for data on number of KIA, MIA, WIA, NBC */
   char CTPers  [10];       /* trailer field - // means ond of data */
   int  Result;
   int  Found = FALSE;
   int  Len, i;
   FILE *DataFp = fopen (DATAFILE, "w");  /* Open output data file */

   Result = fscanf (MsgFd, "%[^/]", TempBuf); /* skip down to KUNITCAS set */
   while ((!Found) && (Result != EOF))
      if (EQUALS (TempBuf, "KUNITCAS"))
         Found = TRUE;
      else
         {
          SkipEOL (MsgFd);                              /* skip to next set */
          Result = fscanf (MsgFd, "%[^/]", TempBuf);    /* keep looking */
         }

   if (!Found)  /* couldn't find the start of the data */
      {
       WriteLog ("PostCAS: can't find KUNITCAS set", "", "", "");
       fclose (DataFp);
       return (ERR);
      }

   fscanf (MsgFd, "/%[^/]", Name);          /* read Unit ID field */
   SkipEOL (MsgFd);                         /* Skip trailing fields */

   fscanf (MsgFd, "%[^/]", TempBuf);            /* skip down to 5KMOCAS set */
   while (!EQUALS (TempBuf, "5KMOSCAS"))
      {
       SkipEOL (MsgFd);
       if (fscanf (MsgFd, "%s", TempBuf) == EOF)  /* Check for error */
          {
           WriteLog ("PostCAS:", "UNITCAS message missing '5KMOCAS' set",
                     "", "");
           fclose (DataFp);
           return (ERR);
          }
      }
   SkipEOL (MsgFd);  /* skip end of line */
   SkipEOL (MsgFd);  /* skip column headings */

   while (!LastLine)   /* Process all lines in columnar set 5KMOSCAS */
       {
        Result = fscanf (MsgFd, "/%s%*[ ]%s%*[ ]%s%*[ ]%s%*[ ]%s%*[ ]%s",
                SSI_MOS, Count[0], Count[1], Count[2], Count[3], CTPers);
```

```
        if (Result == EOF)
           {
            WriteLog ("PostCAS:", "UNITCAS message incomplete", "", "");
            fclose (DataFp);
            return (ERR);
           }

        SkipEOL (MsgFd);

        for (i=0; i<4; i++)          /* replace dashes with zeros */
           if (Count[i][0] == '-')
              strcpy (Count[i], "0");

        fprintf (DataFp, "%s¦%s¦%s¦%s¦%s¦%s\n", Name, SSI_MOS, Count[0],
                 Count[1], Count[2], Count[3]);

        Len = strlen (CTPers);
        if ((Len > 1) && (CTPers [Len-1] == '/') && (CTPers [Len-2] == '/'))
            LastLine = TRUE;  /* This is the last line */
      }

   fclose (DataFp);  /* Close output data file */

   /* Run DBLOAD to update values from message */

   sprintf (Command, "%s file.db cas %s cas.sp > /dev/null 2>> /dev/null",
                     DBLOAD, DATAFILE);

   system (Command);

   WriteLog ("PostCAS: posted CASSTATS message for", Name, "", "");
   return (GOOD);
}
```

```
#include "net.h"
#include "server.h"

char *FOL = "FOL";

int PostPOL (MsgFd)

FILE *MsgFd;

{
   int  LastLine = FALSE;  /* Set after last line read */
   int  Index;
   char Command [80];
   char SetName [20];
   char Name [80];
   char Location [80];
   char Remainder [80];
   int  RemLen;  /* Length of remaining fields */
   char *PtName [20];  /* Allow maximum of 20 different location names */
   char QtyType [3][20];
   long Quantity;
   char Type [20];
   char Unit [4];
   FILE *DataFp = fopen (DATAFILE, "w");  /* Open output data file */
   int  Result;
   int  Found = FALSE;
   int  i;

   Result = fscanf (MsgFd, "%s", SetName);  /* Read '3KPOLLOC' header */
   while ((!Found) && (Result != EOF))
      if (EQUALS (SetName, "3KPOLLOC"))
         Found = TRUE;
      else
         {
          SkipEOL (MsgFd);                              /* skip to next set */
          Result = fscanf (MsgFd, "%s", SetName);  /* Read '3KPOLLOC' header */
         }

   if (!Found)   /* couldn't find the start of the data */
      {
       WriteLog ("PostPOL:", "cannot find '3KPOLLOC' set", "", "");
       fclose (DataFp);
       return (ERR);
      }

   SkipEOL (MsgFd);  /* Skip to next line */
   SkipEOL (MsgFd);  /* Skip description line for columnar set */

   for (i = 0; i < 20; i++)
       PtName [i] = NULL;  /* Initialize pointers to NULL */

   while (!LastLine)  /* Process first columnar set (POLLOC) */
       {
        Result = fscanf (MsgFd, "%*c%2d%*[ ]%[^\n]", &Index, Remainder);
fprintf (stderr, "%d %s\n", Index, Remainder);

        if (Result == EOF)
           {
            WriteLog ("PostPOL:", "POLLOC message incomplete", "", "");
            fclose (DataFp);
```

```
            return (ERR);
           }

        SkipEOL (MsgFd);

        RemLen = strlen (Remainder);  /* Compute length of string */
        if ((Remainder [RemLen-1] == '/') && (Remainder [RemLen-2] == '/'))
           {
            LastLine = TRUE;  /* This is the last line */
            Remainder [RemLen-2] = '\0';  /* Remove those unsightly slashes */
            RemLen -= 2;  /* Compensate */
           }

        for (i = RemLen; (i >= 0) && (Remainder[i] != ' '); i--)
            ;   /* Skip to beginning of last field */

        if (i < 0)
           {
            WriteLog ("PostPOL:", "syntax error decoding '3KPOLLOC' set",
                      "", "");
            fclose (DataFp);
            return (ERR);
           }

        strcpy (Location, Remainder+i+1);  /* Copy last field into Location */
        Remainder[i] = '\0';  /* Stuff end-of-string before last field */
        strcpy (Name, Remainder);          /* Move name to permanent home */
        blsup (Name);  /* Suppress trailing blanks */

        /* Place names into table for later cross-reference */
        if (Index > 19)
           {
            WriteLog ("PostPOL:", "PTNAME index too large",
                      "(max = 20)", "");
            fclose (DataFp);
            return (ERR);
           }

        PtName[Index-1] = malloc (strlen (Name) + 1);  /* Allocate storage */
        strcpy (PtName[Index-1], Name);    /* Copy name to table */

        fprintf (DataFp, "%s|%s\n", Name, Location);  /* Write temporary data */
       }

    fclose (DataFp);  /* Close output data file */
    LastLine = FALSE; /* Reset flag */

    /* Run DBLOAD command to update database before beginning next section */

    sprintf (Command, "%s file.db ptloc %s ptloc.sp > /dev/null 2>> /dev/null",
                      DBLOAD, DATAFILE);

    system (Command);  /* Cannot check error conditions simply */
     /*
       {
        WriteLog ("PostPOL:", "'", Command, "' FAILED");
        WriteLog ("PostPOL:", "unable to update ptloc data", "", "");
        return (ERR);
       }
     */
```

```
   /* Process 'CLTHRE' set */

   if (fscanf (MsgFd, "%s", SetName) == EOF)
      {
       WriteLog ("PostPOL:", "POLLOC message missing '3KCLTHRE' set", "", "");
       return (ERR);
      }

   SkipEOL (MsgFd);
   if (!EQUALS (SetName, "3KCLTHRE"))
      {
       WriteLog ("PostPOL:", "cannot find '3KCLTHRE' set", "", "");
       return (ERR);
      }

   SkipEOL (MsgFd);  /* Skip header line */

   DataFp = fopen (WORKFILE, "w");  /* Create a new data file */

   while (!LastLine)   /* Process all lines in second columnar set (CLTHRE) */
       {
        Result = fscanf (MsgFd, "%*c%2d%*[ ]%s%s%s%s", &Index, QtyType[0],
                         QtyType[1], QtyType[2], Remainder);

        if (Result == EOF)
           {
            WriteLog ("PostPOL:", "POLLOC message incomplete", "", "");
            fclose (DataFp);
            return (ERR);
           }

        SkipEOL (MsgFd);

        RemLen = strlen (Remainder);  /* Compute length of string */
        if ((Remainder [RemLen-1] == '/') && (Remainder [RemLen-2] == '/'))
            LastLine = TRUE;  /* This is the last line */

        if (PtName[Index-1] == NULL)  /* Check for bad index */
           {
            WriteLog ("PostPOL:", "descriptor matches no location",
                      "(in '3KCLTHRE' set)", "");
            fclose (DataFp);
            return (ERR);
           }

        for (i = 0; i < 3; i++)  /* Process each of {fuel, oil, lube} */
            if (!EQUALS (QtyType[i], "-"))
               {
                sscanf (QtyType[i], "%ld%3s%s", &Quantity, Unit, Type);
                fprintf (DataFp, "%s¦%s¦%c¦%ld¦%s\n", PtName[Index-1], Type,
                         FOL[i], Quantity, Unit);  /* Write data record */
               }
       }

   fclose (DataFp);  /* Close output data file */

   /* Run DBLOAD command to update 'pol' records */

   sprintf (Command, "%s file.db pol %s pol.sp > /dev/null 2>> /dev/null",
                     DBLOAD, WORKFILE);
```

```
    system (Command);  /* Cannot check error conditions simply */
     /*
        {
         WriteLog ("PostPOL:", "'", Command, "' FAILED");
         WriteLog ("PostPOL:", "unable to update pol data", "", "");
         return (ERR);
        }
     */

    return (GOOD);
}
```

```
#include "net.h"
#include "server.h"

int PostSUP (MsgFd)

FILE *MsgFd;

{
   char *Ptr;
   char Name [80];
   int  LastLine = FALSE;  /* Set after last line read */
   char Command [80];
   char Quantity [10];
   char Unit [10];
   char Item [20];
   char Model [20];
   char Comments [20];
   char Req;
   char ReqNum [10];
   int  Result;
   int Found = FALSE;
   int  ComLen;
   FILE *DataFp = fopen (DATAFILE, "w");  /* Open output data file */

   Result = fscanf (MsgFd, "%[^/]%*c", Name); /* skip down to UNITIDM set */
   while ((!Found) && (Result != EOF))
      if (EQUALS (Name, "UNITIDM"))
         Found = TRUE;
      else
         {
          SkipEOL (MsgFd);                          /* skip to next set */
          Result = fscanf (MsgFd, "%[^/]%*c", Name);   /* keep looking */
         }

   if (!Found)   /* couldn't find the start of the data */
      {
       WriteLog ("PostSUP:", "cannot find UNITIDM field in message", "", "");
       fclose (DataFp);
       return (ERR);
      }

   fscanf (MsgFd, "%[^/]", Name);  /* read the unit ID */
   if ((Ptr = strrchr(Name, FIELDMARK)) != NULL) /* if the field is labeled */
      Ptr++;  /* skip over the label to the actual data */
   else
      Ptr = Name;

   SkipEOL (MsgFd);  /* Skip trailing fields */

   SkipEOL (MsgFd);  /* Skip 6KSHTSUP introduction */
   SkipEOL (MsgFd);  /* skip the column headers    */

   while (!LastLine)   /* Process all lines in columnar set */
       {
        Result = fscanf  (MsgFd, "/%s%s%s%s%*[ ]%[^\n]", Quantity, Unit,
                          Item, Model, Comments);

        if (Result == EOF)
           {
            WriteLog ("PostSUP:", "SHTSUP message incomplete", "", "");
```

```
            fclose (DataFp);
            return (ERR);
           }

        SkipEOL (MsgFd);

        ComLen = strlen (Comments);
        if ((Comments [ComLen-1] == '/') && (Comments [ComLen-2] == '/'))
           {
            LastLine = TRUE;  /* This is the last line */
            Comments [ComLen-2] = '\0';  /* Remove those unsightly slashes */
           }

        if ((Comments [0] == 'A') && (Comments [1] == '1'))
           {
            Req = 'Y';  /* Indicate that item is on request (A1 = requested) */
            sprintf (ReqNum, "%s", Comments+2);  /* Extract request number */
           }
        else
           {
            Req = 'N';  /* Indicate that item is not on request */
            strcpy (ReqNum, "none");  /* Fill empty field */
           }

        fprintf (DataFp, "%s¦%s¦%s¦%s¦%s¦%c¦%s\n", Ptr, Item,
                 Model, Quantity, Unit, Req, ReqNum);
       }

    fclose (DataFp);  /* Close output data file */

    /* Run DBLOAD to update values from message */

    sprintf (Command, "%s file.db lsi %s lsi.sp > /dev/null 2>> /dev/null",
                      DBLOAD, DATAFILE);

    system (Command);  /* No way to check results right now */

    return (GOOD);
}
```

```
#include "net.h"
#include "server.h"

int PutDTGM (MsgFd)

FILE *MsgFd;

{
   int Year, Month, Day, Hour, Minute, Second;

   RealTime (&Year, &Month, &Day, &Hour, &Minute, &Second);
   fprintf (MsgFd, "%.2d%.2d%.2d\n", Day, Hour, Minute);
}
```

```
#include <stdio.h>

/* return current system date and time in integer values */

int RealTime (Year, Month, Day, Hour, Minute, Second)

int *Year;
int *Month;
int *Day;
int *Hour;
int *Minute;
int *Second;

{
   int retcode = -1;  /* default state is error */
   FILE *Fp;
   FILE *popen();

   if ((Fp = popen ("date '+%y %m %d %H %M %S'", "r")) == NULL)
      return (retcode);

   if (fscanf (Fp,"%d %d %d %d %d %d",Year,Month,Day,Hour,Minute,Second) == 6)
      retcode = 0; /* all is well */

   pclose (Fp);

   return (retcode);
}
```

```
#include "net.h"
#include "server.h"

/* Request() - build an input file for AMS to create a message */

int Request (MsgType, UnitId, Imt, DestPath, Priority)

char *MsgType;
char *UnitId;
char *Imt;
char *DestPath;
char *Priority;

{
   filename DataFile;
   FILE *DataFp;
   register int Result = ERR;
   char Command[80];

   if ((DataFp = StartMsg (MsgType, UnitId, Imt)) == NULL)
      {
       WriteLog ("Request: can't start data file", "", "", "");
       return (ERR);
      }

   /* get data from C2 database and prepare it for AMS */
   switch (MsgType[0])
      {
       case 'c' :
       case 'C' : /* CASSTATS - S006 */
                  Result = MakeCAS (DataFp, UnitId);
                  break;

       case 'p' :
       case 'P' : /* POLLOC - S026 */
                  Result = MakePOL (DataFp);
                  break;

       case 's' :
       case 'S' : /* SHORTSUP - S034 */
                  Result = MakeSUP (DataFp, UnitId);
                  break;

       default  : WriteLog ("Request: don't know how to make", MsgType,
                            "message for", UnitId);
      }

   if (Result != GOOD)
      {
       WriteLog ("Request: could not build datafile for", MsgType, "message",
                 "");
       return (ERR);
      }

   fclose (DataFp);

   /* generate a JINTACCS message from the data in DataFile */
   sprintf (Command, "%s < %s | %s %s %s", AMS, DATAFILE, MSG,
           Priority, DestPath);
   if (system (Command) != GOOD)
```

```
        {
         WriteLog ("Request:", Command, "FAILED", "");
         return (ERR);
        }

    WriteLog ("Request: requested", MsgType, "for", UnitId);

    return (GOOD);
}
```

```
#include "net.h"
#include "server.h"
#include <signal.h>

pathname LogFile; /* global LogFile for server routines */

main (argc, argv)        /* Message Server main program */

int argc;
char ** argv;

{
   pathname ServerQueue;
   pathname NewName, OldName;
   char *NextMsgName;  /* Next message in system input queue */
   int Forever=FALSE;  /* if the "-" argument is given, loop forever */
   FILE *MsgFileFd;
   int c,i;
   int Found;
   char SetID[10];
   char MsgID[6][20];
   char tempbuf[80];
   char Imt[80];
   char ReturnPath[80];
   char Priority[2];
   pathname SiteName;
   FILE *ParamFileFp;
   char Key[20];
   int  Value;
   int  PollDelay = 30;  /* Number of seconds to sleep between scans */

   void ShutDown ();

umask (UMASK);

/* validate and parse arguments */
if ((argc < 3) || (argc > 4))
   usage (argv[0]);

strcpy (ServerQueue, argv[1]); /* get working directory*/
strcpy (SiteName, argv[2]);    /* set input queue */
if ((argc == 4) && (argv[3][0] == '-'))
   Forever = TRUE;           /* continuous operation */

/* set working directory */
if ((chdir (ServerQueue)) != 0)
   {
    fprintf (stderr, "invalid directory\n");
    usage (argv[0]);
   }

unlink ("../.abort");  /* Make sure abort file gets blown away */

sprintf (LogFile, "../log/%s.log", SERVER); /* set the global LogFile */

if (Lock(SERVER) == ERR)
   {
    WriteLog ("Server: can't lock",  ServerQueue, "-", "Goodbye!");
    exit (ERR);
   }
```

```
signal (SIGTERM, ShutDown);  /* Point to shutdown routine on signal 15 */

if (Forever)
   {
    WriteLog ("Server: Scanner mode in", ServerQueue, "for", SiteName);
    if ((ParamFileFp = fopen (PARAMFILE, "r")) != NULL)
       {
        fscanf (ParamFileFp, "%s %d\n", Key, &Value);
        while ((!feof (ParamFileFp)) && (!ferror (ParamFileFp)))
           {
            if (EQUALS(Key, "serverpoll"))
               PollDelay = Value;

            fscanf (ParamFileFp, "%s %d\n", Key, &Value);
           }
       }
   }
else
   WriteLog ("Server: One-Pass mode in", ServerQueue, "for", SiteName);

do    /* non-terminating process loops to scan input queue */
  {
   if (Abort (".."))
      ShutDown ();  /* Shut down if requested */

   while ((NextMsgName = DeQueue (SiteName)) != NULL)
      {
       if (Abort (".."))    /* Check inside loop as well */
          ShutDown ();  /* Shut down if requested */

       WriteLog ("Server: Next message is", NextMsgName, "", "");

       if (( MsgFileFd = FileOpen (NextMsgName, SiteName, "r")) == NULL)
          {
           WriteLog ("Server: can't open", NextMsgName, "", "");
           continue; /* on to next message */
          }

       /* read in the message priority and return address for use below */
       while ((c = getc (MsgFileFd)) != HEADERLINE)  /* skip down to priority */
          SkipEOL (MsgFileFd);
       fscanf (MsgFileFd, "%s", Priority);   /* get the message priority      */
       SkipEOL (MsgFileFd);                  /* skip to the next line         */
       SkipEOL (MsgFileFd);                  /* skip the destination line     */
       getc (MsgFileFd);                     /* skip the headerline character */
       fscanf (MsgFileFd, "%s", ReturnPath); /* get the return address        */
       SkipEOL (MsgFileFd);                  /* skip to the next line         */

       /* this code will skip over any following header lines */
       while ((c=getc(MsgFileFd)) == HEADERLINE)
           SkipEOL (MsgFileFd);

       if (feof(MsgFileFd))
          {
           WriteLog ("Server: empty message file", NextMsgName, "", "");
           sprintf (NewName, "../%s/%s", ERRORQ, NextMsgName);
           sprintf (OldName, "%s/%s", SiteName, NextMsgName);
           if (FRename (OldName, NewName) == ERR)
              {
               WriteLog ("Server: can't move", NextMsgName, "to", ERRORQ);
```

```
            WriteLog ("Server: processing terminated. Goodbye!", "", "", "");
            UnLock (SERVER);
            exit(-1);
           }
        continue; /* go on to next message */
       }

    ungetc (c, MsgFileFd);

    /* read in the EXER/OPER line (if present) and locate the MSGID set */
    strcpy (lmt, "");
    Found = FALSE;
    while (!Found && !feof(MsgFileFd))
       {
        fscanf(MsgFileFd, "%[^/]%*c", SetID);
        if ( ! ( Found = EQUALS(MSGID, SetID) ) )
           {
            if (EQUALS(EXER, SetID) || EQUALS(OPER, SetID))
               {
                fscanf (MsgFileFd, "%[^\n]", tempbuf);
                sprintf (lmt, "%s/%s", SetID, tempbuf);
               }
            SkipEOL (MsgFileFd);
           }
       }

    if (!Found)  /* all messages must have the MSGID set */
       {
        WriteLog ("Server: MSGID not found - invalid message format in",
                  NextMsgName, "", "");
        sprintf (NewName, "../%s/%s", ERRORQ, NextMsgName);
        sprintf (OldName, "%s/%s", SiteName, NextMsgName);
        if (FRename (OldName, NewName) == ERR)
           {
            WriteLog ("Server: can't move", NextMsgName, "to", ERRORQ);
            WriteLog ("Server: processing terminated. Goodbye!", "", "", "");
            UnLock (SERVER);
            exit(-1);
           }

        continue; /* go on to next message */
       }

    /* read in the rest of the MSGID set - end marked by // */
    for (i=0; i<6; i++) /* there may be up to 6 data items */
       {
        fscanf (MsgFileFd, "%[^/\n]%*c", MsgID[i]); /* read data */
        if (strlen(MsgID[i]) == 0)   /* got the // marking end of set */
           break;
       }

    SkipEOL (MsgFileFd);  /* position file pointer at start of next set */

    if ( (i>4) && (EQUALS(MsgID[4],REQ)) ) /* the message is a request */
       Request (MsgID[0], MsgID[1], lmt, ReturnPath, Priority);
    else  /* the message is an update */
       Post (MsgID[0], MsgID[1], MsgFileFd); /* post it to the C2 database */

    WriteLog ("Server:", NextMsgName, "has been processed", "");
    Remove (NextMsgName, SiteName);
```

```
      } /* got on to next message to be processed */

   if (Forever)     /* sleep between scans */
      sleep (PollDelay);

  } while (Forever);     /* Continue scanning queue for more arrivals */

UnLock (SERVER);

WriteLog ("Server: normal termination.", "", "", "");

exit (GOOD);

}
```

```
int usage (Name)
char *Name;
{
   fprintf (stderr, "usage: %s directory sitename [-]\n", Name);
   exit (ERR);
}
```

```
void ShutDown ()

{
   WriteLog ("ShutDown:", "operator requested system shutdown", "", "");
   UnLock (SERVER);
   exit (0);
}
```

```
#include "net.h"
#include "server.h"

/* create the datafile for AMS and write the initial main text of the message */

FILE *StartMsg (MsgName, UnitId, Imt)

char *MsgName;
char *UnitId;
char *Imt;

{
   FILE *Fp;
   char Text[80];
   register int NumFields;

   if ((Fp = fopen(DATAFILE, "w+")) == NULL)
      {
       WriteLog ("StartMsg: can't create", DATAFILE, "", "");
       return (NULL);
      }

   /* write data for EXER or OPER set if present */
   if (strlen(Imt) > 0)
      {
       sscanf (Imt, "%[^/]", Text);   /* get first token */
       Imt = Imt+strlen(Text)+1;      /* advance string pointer */
       fprintf (Fp, "%s\n", Text);

       if (EQUALS(Text, EXER)) /* set loop variable to # of fields in set */
          NumFields = 2;
       else  /* OPER set */
          NumFields = 4;

       while ((NumFields-- > 0) && (sscanf (Imt, "%[^/]", Text) == 1))
          {
           Imt = Imt+strlen(Text)+1;    /* advance string pointer  */
           fprintf (Fp, "%s\n", Text);  /* write token to datafile */
          }

       while (NumFields-- > 0)
          fprintf (Fp, "-\n");  /* dash means empty field */
      }
   else
      fprintf (Fp, "-\n");  /* dash means empty set */

   /* write data  for MSGID set */
   fprintf (Fp, "%s\n%s\n-\n-\n-\n-\n", MsgName, UnitId);

   /* write data for REF, AMPN, and NARR sets - all empty */
   fprintf (Fp, "//\n");  /* empty REF set */
   fprintf (Fp, "//\n");  /* empty AMPN set */
   fprintf (Fp, "//\n");  /* empty MARR set */

   fflush (Fp);

   return (Fp);
```

```
#include "net.h"
#include "server.h"

/*
   UnitID - Return the name of this unit, found in the file UNITID.
*/

char *UnitID ()

{
   sitename TempSiteName;    /* Temporary storage for site name */
   FILE     *UnitIDFd;       /* UNITID file descriptor */
   char     *RetPtr;         /* Pointer to return to caller */

   if ((UnitIDFd = fopen (UNITID, "r")) == NULL)
      {
       fprintf (stderr, "UnitID: I don't know my own name.\n");
       return ((char *) NULL);  /* Try to recover */
      }

   fscanf (UnitIDFd, "%s", TempSiteName);  /* Get this site's name */

   fclose (UnitIDFd);

   RetPtr = malloc (strlen (TempSiteName) + 1);   /* Get perm. storage */
   strcpy (RetPtr, TempSiteName);

   return (RetPtr);
}
```

JMS

This section contains the functions used only by the screen oriented JINTACCS message preparation program (JMS).

```
PATH=/bin:/usr/bin:/usr/unify/bin
clean:
        rm -f jms *.o

jms: jms.h jms.o dsp.o form.o\
     list.o scr.o util.o
        uld jms jms.o dsp.o form.o\
           list.o scr.o util.o  -lcurses -ltermcap
         ln jms ../../bin

jms.o:  jms.c jms.h
        ucc -c -Mm jms.c

dsp.o: dsp.c jms.h
        ucc -c -Mm dsp.c

form.o: form.c jms.h
        cc -c -Mm form.c

list.o:  list.c jms.h
        cc -c -Mm list.c

scr.o: scr.c jms.h
        cc -c -Mm scr.c

util.o: util.c jms.h
        cc -c -Mm util.c
```

```
/* ------------------------------------------------------------------ */
/*   file name: ams.e                                                 */
/*                                                                    */
/*   This file contains extern declarations for global variables      */
/* ------------------------------------------------------------------ */

extern WINDOW *main_box_win;
extern WINDOW *main_win;
extern WINDOW *command_win;
extern WINDOW *msg_win;
extern WINDOW *status_win;
extern WINDOW *view_win;
extern WINDOW *view_box_win;
extern WINDOW *help_win;
extern WINDOW *helpcom_win;

extern int    x, y;          /*  screen position: (y,x) coordinate */
extern int    col;           /*  number of columns for indentation */

extern int    SWITCH;        /*  Switch Indicator for commander    */
extern int    EMPTY_TEXT;    /*  Indicator for empty text          */
extern int    SET_STOPPER;   /*  Indicator for leave-the-set       */
extern int    VIEWED;        /*  view_win Indicator                */

extern char   ALIAS[];
extern char   MNO[];
extern char   TITLE[];
extern char   FNAME[];       /*  data field identifier       */
extern char   FDESC[];
extern char   SCOL_J[];      /*  start column                */
extern char   CHDR[];

extern list0    *head;
extern list0    *ptr0;
extern list1    *ptr1;

extern lsnode   *taills;
extern csfnode  *tailcsf;
extern csdnode  *tailcsd;
extern tsnode   *tailts;
```

```
/* ------------------------------------------------------------------- */
/*   file name: ams.h                                                  */
/*                                                                     */
/*   This file contains definitions and declarations globally used     */
/* ------------------------------------------------------------------- */

/* constants and definitions for message entry system */

#include  <stdio.h>
#include  <curses.h>
#include  <signal.h>

#define   BS          0x8
#define   BEEP        0x7
#define   DEL         0x7f
#define   ESC         0x1B
#define   EQUALS      !strcmp
#define   MAXLINES    20
#define   MWLINES     18          /* Number of lines of main_win */
#define   lprf        "/dev/lpr"

struct lst0_str
          {
          char   mtag1[16],MsgNo[5];
          char   mtag2[7],mid[11];
          struct lst0_str  *Prev, *Next;
          struct lst1_str  *schild;
          } ;




struct lst1_str
          {
          char   settyp;
          char   setcat[4];
          char   SetID[9];
          struct lst0_str  *parent;
          struct lst1_str  *Prev, *Next;
          union {
                  struct lsnode_str  *lschild;
                  struct csnode_str  *cschild;
                  struct tsnode_str  *tschild;
                } uval;
          } ;
```

```
struct lsnode_str
        {
        int     fdno;
        char    fdcat[4];
        char    fdname[41];
        char    fdesc[9];
        char    fval[25];
        char    j[2];           /* Left/right justification      */
        char    notype[11];
        char    dtype[5];       /* Combination of A, N, B and S  */
        int     dmin;           /* Lower bound for the data value */
        int     dmax;           /* Upper bound for the data value */
        int     x;              /* screen position: x-coordinate  */
        int     y;              /* screen position: y-coordinate  */
        struct lstl_str  *parent;
        struct lsnode_str  *Prev, *Next, *fchild;
        } ;



struct csnode_str
        {
        char    fdcat[2];
        char    fdname[41];
        char    colhdr[25];
        char    colpos[4];
        char    j[2];           /* Left/right justification      */
        char    notype[11];
        char    dtype[5];       /* Combination of A, N, B and S  */
        int     dmin;           /* Lower bound for the data value */
        int     dmax;           /* Upper bound for the data value */
        struct lstl_str  *parent;
        struct csnode_str  *Prev, *Next;
        struct csdata_str  *fchild;
        } ;



struct csdata_str
        {
        char    csdata[25];
        struct csnode_str  *parent;
        struct csdata_str  *Prev, *Next;
        } ;
```

```
struct tsnode_str
          {
          char   **tstext;
          struct lstl_str  *parent;
          } ;
```

```
typedef struct lst0_str list0;
typedef struct lstl_str listl;
typedef struct lsnode_str lsnode;
typedef struct csnode_str csfnode;
typedef struct csdata_str csdnode;
typedef struct tsnode_str tsnode;

/* C library functions */

char *malloc();
int   strcmp();
int   strlen();
char *strcpy();
char *strncpy();
char *strcat();

/* user defined functions */

int    atoin();
char   *upchar();
char   *upstr();
char   *bl_sup();
char   *bl_pad();
char   *bl_pad_front();
char   *readstr();
int    addscr_csf();
int    addscr_csline();
int    convert();
int    convlslist();
int    convcsflist();
int    convcsdlist();
int    convtsnode();
int    readtxt();
int    SET_handler();
int    FIELD_handler();
int    DFI_handler();
int    VALID();
int    freef_handler();
int    add0node();
int    addlnode();
int    add_ls_node();
int    add_csf_node();
int    add_csd_node();
int    add_ts_node();
```

```
int     del0node();
int     del1node();
int     del1snode();
int     del0list();
int     del1list();
int     delcsflist();
int     delcsfnode();
int     delcsdnode();
int     deltsnode();

#ifndef MAIN
   #include "ams.e"
#endif
```

```
$include "../../def/file.h"
#include "jms.h"

/* ------------------------------------------------------------------ */
/*   file name: dsp.c                                                 */
/*                                                                    */
/*   This file contains routines to display field level data on screen */
/* ------------------------------------------------------------------ */

/* ================================================================== */
/*     1. Get the field-level data from the columnar set list         */
/*     2. Display them on screen                                      */
/* ================================================================== */

addscr_csf(head)

csfnode  *head;
{
int    i=1, n;
csfnode  *ptr;

ptr=head;
y++;
wmove(main_win,y,1);
while(ptr)
    {
    n = atoin(ptr->colpos,strlen(ptr->colpos)-1);
    for(; i<n ; i++)
      waddch(main_win,' ');
    waddstr(main_win,ptr->colhdr);
    i=i+strlen(ptr->colhdr);
    ptr=ptr->Next;
    }
wrefresh(main_win);
}
```

```
/* ================================================================== */
/*     1. Get the field-level data from the columnar set list         */
/*     2. Display them on screen                                      */
/* ================================================================== */

addscr_csline(head)

csfnode  *head;
{
int    i=1, j;
int    POS, DIFF;
csfnode  *ptrc;

if (y < MWLINES-1)
   y++;
```

```
else
   waddch(main_win,'\n');
wmove(main_win,y,1);
ptrc=head;
while(ptrc)
    {
    POS = atoin(ptrc->colpos,strlen(ptrc->colpos)-1);
    for(; i<POS ; i++)  waddch(main_win,' ');
    if(EQUALS(ptrc->j,"L"))    /* Left Justification */
       {
        for(j=0; j<ptrc->dmax ; j++)  waddch(main_win,'_');
        i=i+ptrc->dmax;
       }
    else                       /* Right Justification */
       {
        DIFF = strlen(ptrc->colhdr) - ptrc->dmax;
        for(j=0; j<DIFF ; j++)  waddch(main_win,' ');
        for(j=0; j<ptrc->dmax ; j++)  waddch(main_win,'_');
        i=i+strlen(ptrc->colhdr);
       }
    ptrc=ptrc->Next;
    }
wrefresh(main_win);
}
```

```
/* ================================================================== */
/*    Add a data node as the sublevel of the corresponding field node  */
/* ================================================================== */

addscr_csd()

{
register int    k,j;
int    n;
int    LIVE = FALSE;
int    FIRST = TRUE;
char   str[31];
csfnode  *fptr;
csdnode  **dptr, **node;

fptr = (*ptr1).uval.cschild; /* point to the head of field list */
for(j=0; fptr; j++)  /* count the fields */
    fptr = fptr->Next;

/* allocate storage for pointers to field data nodes */
node = (csdnode **)malloc(j*sizeof(csdnode *)); /* data nodes */
dptr = (csdnode **)malloc(j*sizeof(csdnode *)); /* tail ptr   */
str[0] = NULL;
while(!EQUALS(str,"/"))  /* loop till user enters "/" - end of set */
     {
     fptr = ptr1->uval.cschild; /* point to the head of field list */
     addscr_csline(fptr);
```

```
    /* loop through the field nodes gathering data till user says quit */
    for(k=0; ((k<j) && (fptr)); k++)
        {
        wclear(msg_win);
        wrefresh(msg_win);
        if(EQUALS(fptr->j,"L"))
            {
             n = atoin(fptr->colpos,strlen(fptr->colpos)-1);
             wmove(main_win,y,n);
            }
        else
            {
             n = atoin(fptr->colpos,strlen(fptr->colpos)-1) +
                 strlen(fptr->colhdr) - fptr->dmax;
             wmove(main_win,y,n);
            }
        wrefresh(main_win);
        wget_field(main_win,str,fptr->dtype,fptr->dmin,fptr->dmax,'C',fptr->fdcat);
        while(EQUALS(bl_sup(str),"/") && EQUALS(bl_sup(ptr1->setcat),"M")
                && FIRST)
              {
               wclear(msg_win);
               mvwaddstr(msg_win,0,0,"The set is MANDATORY.");
               mvwaddstr(msg_win,1,0,"Reenter ");
               waddstr(msg_win,fptr->fdname);
               wrefresh(msg_win);
               wmove(main_win,y,n);
               wrefresh(main_win);
               wget_field(main_win,str,fptr->dtype,fptr->dmin,fptr->dmax,'C',fptr->fdcat);
              }
        FIRST = FALSE;
        if(EQUALS(bl_sup(str),"\0"))
              strcpy(str,"-");
        if(k==0 && EQUALS(str,"/"))
              break;
        while(k != 0 && EQUALS(bl_sup(str),"/"))
              {
               wclear(msg_win);
               mvwaddstr(msg_win,0,0,"The set is MANDATORY.");
               mvwaddstr(msg_win,1,0,"Reenter ");
               waddstr(msg_win,fptr->fdname);
               wrefresh(msg_win);
               wmove(main_win,y,n);
               wrefresh(main_win);
               wget_field(main_win,str,fptr->dtype,fptr->dmin,fptr->dmax,'C',fptr->fdcat);
              }
        node[k] = (csdnode *)malloc(sizeof(csdnode)); /*data space*/
        if(fptr->fchild == NULL) /* first field node in list */
                  {
                  fptr->fchild = node[k];
                  node[k]->Prev = NULL;
                  node[k]->Next = NULL;
                  node[k]->parent = fptr;
                  strcpy(node[k]->csdata,str);
                  dptr[k] = node[k];
                  }
        else      /* all following field nodes */
                  {
                  node[k]->Prev = dptr[k];
                  node[k]->Next = NULL;
                  node[k]->parent = dptr[k]->parent;
```

```
                    dptr[k]->Next = node[k];
                    strcpy(node[k]->csdata,str);
                    dptr[k] = node[k];
                    }
          fptr=fptr->Next;
          } /* end of for */
     } /* end of while */
/*  Test if the columnar set has some data in it  */
fptr = ptrl->uval.cschild;
while(fptr)
     {
      if(fptr->fchild == NULL)
         fptr = fptr->Next;
      else
         {
         LIVE = TRUE;
         break;
         }
     }
if(!LIVE)
   dellnode(ptrl);
}
```

```
/* ================================================================== */
/*     1. Get the field-level data from the columnar set list         */
/*     2. Display them on screen                                      */
/* ================================================================== */

addscr_lsf(head)

lsnode  *head;
{
int     i;
lsnode  *ptr;
ptr=head;
while(ptr)
     {
     wprintw(main_win,"\n%s  : ",bl_pad_front(ptr->fdname,40));
     getyx(main_win,y,x);
     for (i=0; i < ptr->dmax; i++) waddch(main_win,'_');
     wrefresh(main_win);
     ptr->y = y;
     ptr->x = x;
     ptr=ptr->Next;
     }
}
```

```
/* ================================================================== */
/*        Insert field value into the linear set node                 */
/* ================================================================== */

addscr_lsd()

{
int     THRU, FNO=0, POS=1;
char    DVAL[21], COMBI[30];
lsnode  *head, *ptr, *ptrpos, *oldptr;

head = (*ptr1).uval.lschild; /* point to the head of field list */

ptr = head;
while(ptr)                    /* Initialize dfi values to "-" */
    {
    strcpy(ptr->fval,"-");
    ptr = ptr->Next;
    }

ptr = head;
while(ptr)
    {
    wclear(msg_win);
    wrefresh(msg_win);
    wmove(main_win,ptr->y,ptr->x);
    wrefresh(main_win);
    getyx(main_win,y,x);
    wget_field(main_win,DVAL,ptr->dtype,ptr->dmin,ptr->dmax,'L',ptr->fdcat);
    THRU = FALSE;
    while(!THRU)
        {
            if(EQUALS(bl_sup(ptr1->SetID),"MSGID") && ptr->fdno==1)
               {
               strcpy(ALIAS,upstr(DVAL));
               while (acckey (msg,bl_pad(ALIAS,10)))
                  {
                  print_imsg("WARNING: ILLEGAL Message Type!");
                  wmove(main_win,y,x);
                  wrefresh(main_win);
                  wget_field(main_win,DVAL,ptr->dtype,ptr->dmin,ptr->dmax,'L',ptr->fdcat);
                  strcpy(ALIAS,upstr(DVAL));
                  }
               gfield (mno, MNO);
               MNO[4] = '\0';
               gfield(mtitle, TITLE);
               TITLE[30] = '\0';
               strcpy(ptr0->MsgNo,MNO);
               strcpy(ptr0->mid,ALIAS);
               }
        if(EQUALS(ptr->fdcat,"M"))
            while(EQUALS(bl_sup(DVAL),"\0"))
                {
                print_imsg("This field is MANDATORY.");
                wmove(main_win,y,x);
                wrefresh(main_win);
                wget_field(main_win,DVAL,ptr->dtype,ptr->dmin,ptr->dmax,'L',ptr->fdcat);
                }
        while (!VALID(bl_sup(DVAL),ptr->notype))
```

```
            {
            print_imsg("The valid data type is",ptr->notype);
            wmove(main_win,y,x);
            wrefresh(main_win);
            wget_field(main_win,DVAL,ptr->dtype,ptr->dmin,ptr->dmax,'L',ptr->fdcat);
            }
        if(EQUALS(DVAL,"\0"))
            strcpy(DVAL,"-");
        if(!EQUALS(ptr->fdesc,"\0"))
            {
            strcpy(COMBI,ptr->fdesc);
            strcat(COMBI,":");
            strcat(COMBI,DVAL);
            strcpy(ptr->fval,COMBI);
            }
        else if(!EQUALS(DVAL,"/"))
            strcpy(ptr->fval,DVAL);
        if(!EQUALS(DVAL,"-") && !EQUALS(DVAL,"") && !EQUALS(DVAL,"/"))
            POS = ptr->fdno + 1;
        else if(POS == ptr->fdno)
            ptrpos = ptr;
        THRU = TRUE;
        if(EQUALS(bl_sup(DVAL),"/"))
            if(!EQUALS(bl_sup(ptr1->setcat),"M"))
                SET_STOPPER = TRUE;
            else
                {
                print_imsg("This set is MANDATORY.");
                wmove(main_win,y,x);
                wrefresh(main_win);
                wget_field(main_win,DVAL,ptr->dtype,ptr->dmin,ptr->dmax,'L',ptr->fdcat);
                THRU = FALSE;
                }
        if (SET_STOPPER)
           {
            FNO = ptr->fdno;
            while (ptr)
                {
                oldptr = ptr;
                ptr = ptr->Next;
                }
            wmove(main_win,oldptr->y,oldptr->x);
            wrefresh(main_win);
            if(POS == FNO)
               dellslist(ptrpos);
            return;
           }
        } /* end of while(THRU) */
    wrefresh(main_win);
    FNO = ptr->fdno;
    ptr=ptr->Next;
    }  /* end of while */
if(POS <= FNO)
   dellslist(ptrpos);
```

```
/* ========================================================================== */

/* ========================================================================== */

waddchf(xwin,X)

WINDOW  *xwin;
char    X;
{
    waddch(xwin,X);
    fprintf("/usr/airmics/shinn/bin/ams.out","%c",X);
}




waddstrf(xwin,STR)

WINDOW  *xwin;
char    *STR;
{
    waddstr(xwin,STR);
    fprintf("/usr/airmics/shinn/bin/ams.out","%s",STR);
}




wprintwf(xwin,FORM,STR)

WINDOW  *xwin;
char    *FORM;
char    *STR;
{
    wprintw(xwin,FORM,STR);
    fprintf("/usr/airmics/shinn/bin/ams.out",STR);
}
```

```
#include "jms.h"

FILE   *WP_imgf;

int  LINE, COL;

/* ------------------------------------------------------------------- */
/*   file name: form.c                                                 */
/*                                                                     */
/*   This file contains routines to display messages in JINTACCS format */
/* ------------------------------------------------------------------- */

/* =================================================================== */
/*   Build up the message in JINTACCS format: Top Level Routine        */
/*     1. Get the message data from the linked lists                   */
/*     2. Convert them into standard output format                     */
/* =================================================================== */

int convert()

{
listl  *ptrl;

WP_imgf = fopen("imgf","w");
wclear(view_win);

if(!ptr0)
          {
           print_imsg(" *** ERROR: No message found !!! ");
           return;
          }

ptrl=ptr0->schild;
while(ptrl)
     {
     wprintw(view_win,"%s",ptrl->SetID);
     fprintf(WP_imgf,"%s",ptrl->SetID);
     switch(ptrl->settyp)
          {
          case 'C':     /*   columnar set   */
                        convcsflist((*ptrl).uval.cschild);
                        convcsdlist((*ptrl).uval.cschild);
                        break;
          case 'F':     /*   free text set   */
                        convtsnode((*ptrl).uval.tschild);
                        break;
          case 'L':     /*   linear set   */
                        convlslist((*ptrl).uval.lschild);
                        break;
          }
     wprintw(view_win,"//");
     fprintf(WP_imgf,"//");
     if (ptrl->Next)
        {
         if (is_last_line(view_win))
            {
             touchwin(view_box_win);
             wrefresh(view_box_win);
             getyx(command_win,LINE,COL);
             wclear(msg_win);
```

```
            wprintw(msg_win,"Press SPACE Bar for More");
            wrefresh(msg_win);
            wgetchar(msg_win);
            wclear(view_win);
            wrefresh(view_win);
            wmove(command_win,LINE,COL);
            wrefresh(command_win);
            wclear(msg_win);
            wrefresh(msg_win);
           }
         else
           {
            wprintw(view_win,"\n");
            fprintf(WP_imgf,"\n");
           }
       }
    ptrl=ptrl->Next;
    }
fclose(WP_imgf);
}
```

```
/* ========================================================================= */
/*     1. Get the field-level data from the linear set list                  */
/*     2. Convert them into standard output format                           */
/* ========================================================================= */

convlslist(ptr)

lsnode  *ptr;

{
while(ptr)
    {
    getyx(view_win,LINE,COL);
    if (is_last_line(view_win) && (COL+strlen(ptr->fval)+1 > view_win->_maxx))
       wprintw(view_win,"\n");
    wprintw(view_win,"/%s", ptr->fval);
    fprintf(WP_imgf,"/%s", ptr->fval);
    dsp_lin_set(ptr->fchild);
    ptr=ptr->Next;
    }
}
```

```
/* ================================================================== */
/*     1. Get the field-level data from the linear set list           */
/*     2. Display them on screen                                      */
/* ================================================================== */

dsp_lin_set(ptr)

lsnode  *ptr;

{
int    i;

col += 5;
while(ptr)
    {
    for(i=0; i<col; i++) waddch(main_win,' ');
    wprintw(main_win,"%s: %s\n", ptr->fdname, ptr->fval);
    wrefresh(main_win);
    dsp_lin_set(ptr->fchild);
    ptr=ptr->Next;
    }
col -= 5;
}
```

```
/* ================================================================== */
/*     1. Get the field-level data from the columnar set list         */
/*     2. Convert them into standard output format                    */
/* ================================================================== */

convcsflist(head)

csfnode  *head;
{
int    i=2, n;
csfnode  *ptr;

    if (is_last_line(view_win))
       {
        touchwin(view_box_win);
        wrefresh(view_box_win);
        getyx(command_win,LINE,COL);
        wclear(msg_win);
        wprintw(msg_win,"Press SPACE Bar for More");
        wrefresh(msg_win);
        wgetchar(msg_win);
        wclear(view_win);
        wprintw(view_win,"/");
        wrefresh(view_win);
        wclear(msg_win);
        wrefresh(msg_win);
```

```
        wmove(command_win,LINE,COL);
        wrefresh(command_win);
       }
    else
       {
        wprintw(view_win,"\n/");
       }
      fprintf(WP_imgf,"\n/");
      ptr=head;
      while(ptr)
         {
         n = atoin(ptr->colpos,strlen(ptr->colpos)-1);
         for(; i<n ; i++)
            {
            waddch(view_win,' ');
            fprintf(WP_imgf," ");
            }
         wprintw(view_win,"%s", ptr->colhdr);
         fprintf(WP_imgf,"%s", ptr->colhdr);
         i=i+strlen(ptr->colhdr)-1;
         ptr=ptr->Next;
         }
      }
```

```
/* ================================================================== */
/*     1. Get the field-level data from the columnar set data list    */
/*     2. Convert them into standard output format                    */
/* ================================================================== */

convcsdlist(head)

csfnode  *head;
{
int    i, j, k, n;
int    LR, m;
csfnode  *fptr;
csdnode  **dptr;

fptr = head; /* point to the head of field list */
for(j=0; fptr; j++)  /* count the fields */
       fptr = fptr->Next;
/* allocate storage for pointers to column value lists */
dptr = (csdnode **)malloc(j*sizeof(csdnode *));

/* set pointers to column lists */
for(k=0, fptr=head; ((fptr) && (k<j)); k++)
   {
    dptr[k] = fptr->fchild;
    fptr = fptr->Next;
   }

while(dptr[0]) /* while there are still data values to print */
```

```
        {
    if (is_last_line(view_win))
       {
        touchwin(view_box_win);
        wrefresh(view_box_win);
        getyx(command_win,LINE,COL);
        wclear(msg_win);
        wprintw(msg_win,"Press SPACE Bar for More");
        wrefresh(msg_win);
        wgetchar(msg_win);
        wclear(view_win);
        wprintw(view_win,"/");
        wrefresh(view_win);
        wclear(msg_win);
        wrefresh(msg_win);
        wmove(command_win,LINE,COL);
        wrefresh(command_win);
       }
    else
       {
        wprintw(view_win,"\n/");  /* begin new line with slash */
       }
          fprintf(WP_imgf,"\n/");  /* begin new line with slash */
          i = 2;
          fptr = head;
          /* print one line of data, tabbing to column fields */
          for(k=0; fptr; k++) /* indent appropriately and print data */
               {
                n = atoin(fptr->colpos,strlen(fptr->colpos)-1);
                LR = strlen(fptr->colpos);
                switch(fptr->colpos[LR-1]) /* tab to Next field */
                     {
                     case 'L':
                               for(; i<n ; i++)
                                  {
                                   waddch(view_win,' ');
                                   fprintf(WP_imgf," ");
                                  }
                               break;
                     case 'R':
                               m=n+strlen(fptr->colhdr)-strlen(dptr[k]->csdata);
                               for(; i<m ; i++)
                                  {
                                   waddch(view_win,' ');
                                   fprintf(WP_imgf," ");
                                  }
                               break;
                     }
                wprintw(view_win,"%s", dptr[k]->csdata);
                fprintf(WP_imgf,"%s", dptr[k]->csdata);
                i = i+strlen(dptr[k]->csdata)-1;
                dptr[k] = dptr[k]->Next;
                fptr = fptr->Next;
               }
          }
      free(dptr);
      }
```

```
/* ================================================================== */
/*     1. Get the field-level data from the free text set list        */
/*     2. Convert them into standard output format                    */
/* ================================================================== */

convtsnode(ptr)

tsnode  *ptr;
{
int    k=0;
wprintw(view_win,"/");
fprintf(WP_imgf,"/");
while(ptr->tstext[k])
    {
    wprintw(view_win,"%s", ptr->tstext[k]);
    fprintf(WP_imgf,"%s", ptr->tstext[k]);
    k++;
    if(ptr->tstext[k])
       {
        if (is_last_line(view_win))
           {
            touchwin(view_box_win);
            wrefresh(view_box_win);
            getyx(command_win,LINE,COL);
            wclear(msg_win);
            wprintw(msg_win,"Press SPACE Bar for More");
            wrefresh(msg_win);
            wgetchar(msg_win);
            wclear(view_win);
            wrefresh(view_win);
            wclear(msg_win);
            wrefresh(msg_win);
            wmove(command_win,LINE,COL);
            wrefresh(command_win);
           }
        else
           {
        wprintw(view_win,"\n");
      }
       fprintf(WP_imgf,"\n");
       }
    }
}




/* ================================================================== */

/* ================================================================== */

is_last_line(win)
```

```
WINDOW  *win;
{
int L, C;

getyx(win,L,C);
if(L == win->_maxy-1)
   return(1);
else
   return(0);
}
```

```
$include "../../def/file.h"
#define MAIN
#include "jms.h"

/* ------------------------------------------------------------------ */
/*   file name: jms.c                                                 */
/*                                                                    */
/*   This file includes main program and subroutines accessing Database */
/* ------------------------------------------------------------------ */

/* external definitions for global variables */

WINDOW  *main_box_win;
WINDOW  *main_win;
WINDOW  *maincom_win;
WINDOW  *command_win;
WINDOW  *msg_win;
WINDOW  *view_box_win;
WINDOW  *view_win;
WINDOW  *help_win;
WINDOW  *helpcom_win;

FILE   *fopen(), *popen();

int    x, y;         /*  screen position: (y,x) coordinate            */
int    col = 0;      /*  number of columns for indentation            */
int    SWITCH;       /*  Switch between COMMAND and INPUT modes       */
int    EMPTY_TEXT;   /*  Indicator for empty text in the free text set */
int    SET_STOPPER;  /*  indicator for leave-the-set                  */

char   ALIAS[11];    /*  Short Message Title     */
char   MNO[5];       /*  message number          */
char   TITLE[31];    /*  message title           */
char   FDESC[9];     /*  data field descriptor   */
char   FNAME[41];    /*  data field identifier   */
char   SCOL_J[4];    /*  start column            */
char   CHDR[25];     /*  column header           */

list0    *head;      /*  Head pointer to level 0 linked list       */
list0    *ptr0;      /*  Current position of level 0 linked list   */
list1    *ptr1;      /*  Current position of level 1 linked list   */

lsnode   *tails;     /*  Tail pointer to linear set list (level 2)          */
csfnode  *tailcsf;   /*  Tail pointer to columnar set header list (level 2) */
csdnode  *tailcsd;   /*  Tail pointer to columnar set data list (level 2)   */
tsnode   *tailts;    /*  Tail pointer to free text set list (level 2)       */

 /* ********************************************************************** */
 /*                                                                        */
 /*        ams.c                                                           */
 /*                                                                        */
 /*        THE MESSAGE SYSTEM                                              */
 /*                                                                        */
 /*        This is the main routine                                        */
 /*                                                                        */
 /* ********************************************************************** */

main ()

{
```

```
int die();   /* signal handler */

head = NULL;        /* Initialize a pointer to top level linked list */

initscr();
crmode();
noraw();
nonl();
noecho();

signal(SIGINT, die);

main_box_win = newwin(20,80,0,0);
main_win = newwin(18,76,1,2);
maincom_win = newwin(2,80,20,0);
command_win = newwin(2,80,20,0);
msg_win = newwin(2,80,22,0);
wstandout(msg_win);
view_box_win = newwin(11,74,8,2);
view_win = subwin(view_box_win, 9,70,9,4);
help_win = newwin(20,80,0,0);
helpcom_win = newwin(2,80,20,0);

scrollok(main_win,TRUE);

Frame(main_box_win);
header(main_box_win, "JINTACCS MESSAGE SYSTEM",1);
footnote(main_box_win, "Press ESC key to Switch Mode: INPUT <-> COMMAND",1);
wrefresh (main_box_win);

box(view_box_win,'|','-');
header(view_box_win, "JINTACCS MESSAGE OUTPUT",1);

wmove(maincom_win,0,0);
wprintw(maincom_win,"Type \"\/\" to leave the current set if it is optional");

wmove(help_win,0,0);
wprintw(help_win,"***  HELP Window  ***  ");
wmove(help_win,2,0);
wprintw(help_win,"1. COMMAND Menus: ");
wmove(help_win,3,3);
wprintw(help_win,"Create: Create a new message ");
wmove(help_win,4,3);
wprintw(help_win,"Movewin: Move VIEW window ");
wmove(help_win,5,3);
wprintw(help_win,"Print: Make a hard copy of the current message ");
wmove(help_win,6,3);
wprintw(help_win,"Quit: Quit the Message System ");
wmove(help_win,7,3);
wprintw(help_win,"Save: Save the current message in permanent MESSAGES file ");
wmove(help_win,8,3);
wprintw(help_win,"View: View the current message which is being created ");

wprintw(helpcom_win,"Press ENTER key to back to COMMAND menu");

wmove(command_win,0,0);
wprintw(command_win,"COMMAND: Create  Help  Movewin  Print  Quit  Save  View");
wmove(command_win,1,0);
wprintw(command_win,"Type Command Letter : ");

command_handler();
```

```
clear();
refresh();
endwin();
}     /*   end of main   */




/* ========================================================================= */
/*     Command Handler                                                       */
/* ========================================================================= */

command_handler()
{
SWITCH = 1;
while (SWITCH)
    {
     touchwin(command_win);
     wmove(command_win, 1, 22);
     wrefresh(command_win);
     commander(wgetchar(command_win));
    }
}




/* ========================================================================= */
/*     Actual Command Executioner                                            */
/* ========================================================================= */

commander(ICOM)

int   ICOM;
{
int   c;
int   curx, cury;
FILE  *AP_msgsf, *RP_imgf, *WP_lpr;

wclear(msg_win);
wrefresh(msg_win);

waddch(command_win, BS);   /* erase the command input character */
waddch(command_win, ' ');
waddch(command_win, BS);

switch(ICOM)
    {
```

```
    case 'c':
    case 'C':
                touchwin(maincom_win);
                wrefresh(maincom_win);
                create_JIN_msg();
                touchwin(command_win);
                wrefresh(command_win);
                break;
    case 'h':
    case 'H':
                touchwin(help_win);
                wrefresh(help_win);
                touchwin(helpcom_win);
                wrefresh(helpcom_win);
                while ((c=wgetch(helpcom_win)) != '\r')
                   ;
                touchwin(main_box_win);
                wrefresh(main_box_win);
                touchwin(main_win);
                wrefresh(main_win);
                touchwin(command_win);
                wrefresh(command_win);
                break;
    case 'm':
    case 'M':
                print_cmsg("use vi cursor keys to position window, then press ENTER key");
                cury = view_box_win->_begy;
                curx = view_box_win->_begx;
                wrefresh(view_box_win);
                wmove(main_box_win, cury, curx);
                wrefresh(main_box_win);
                while ((c=wgetch(main_box_win)) != '\r')
                   {
                    switch (c)
                       {
                        case 'h': if (curx <= 2) putchar(BEEP);
                                  else  --curx;
                                  break;
                        case 'j': if (cury >= 8) putchar(BEEP);
                                  else  ++cury;
                                  break;
                        case 'k': if (cury <= 2) putchar(BEEP);
                                  else  --cury;
                                  break;
                        case 'l': if (curx >= 8) putchar(BEEP);
                                  else  ++curx;
                                  break;
                       }
                    wmove(main_box_win, cury, curx);
                    wrefresh(main_box_win);
                   }
                wclear(msg_win);
                wrefresh(msg_win);
                mvwin(view_box_win, cury, curx);
                mvwin(view_win, cury+1, curx+2);
                touchwin(main_box_win);
                touchwin(main_win);
                wrefresh(main_box_win);
                wrefresh(main_win);
                wrefresh(view_box_win);
                wrefresh(view_win);
```

```
            break;
    case 'p':
    case 'P':
            RP_imgf = fopen("imgf","r");
            WP_lpr = popen("lpr","w");
            while((c=getc(RP_imgf)) != EOF) putc(c,WP_lpr);
            pclose(WP_lpr);
            fclose(RP_imgf);
            break;
    case 'q':
    case 'Q':
            clear();
            refresh();
            endwin();
            exit(1);
            break;
    case 's':
    case 'S':
            RP_imgf = fopen("imgf","r");
            AP_msgsf = fopen("msgsf","a");
            fprintf(AP_msgsf,"\n**********\n");
            convert();
            while((c=getc(RP_imgf)) != EOF) putc(c,AP_msgsf);
            fprintf(AP_msgsf,"\n");
            fclose(AP_msgsf);
            fclose(RP_imgf);
            print_cmsg(" The message has been saved in \"msgsf\" file ");
            break;
    case 'v':
    case 'V':
            convert();
            touchwin(view_box_win);
            wrefresh(view_box_win);
            break;
    case ESC:
            SWITCH = 0;
            wclear(msg_win);
            wrefresh(msg_win);
            touchwin(maincom_win);
            wrefresh(maincom_win);
            touchwin(main_win);
            wmove(main_win,y,x);
            wrefresh(main_win);
            break;
    default :
            waddch(command_win, ICOM);    /* show the invalid command letter */
            waddch(command_win, BS);      /* put the cursor on it */
            print_cmsg("Invalid command.  Try again.");
            break;
  } /* end of switch */

} /* end of proc */
```

```
/* ********************************************************************** */
/*      Create a JINTACCS Message : Message Level (or Level 0)            */
/* ********************************************************************** */

create_JIN_msg()

{
char    label1[16],label2[7];
char    option[5];          /*  EXER or OPER */
char    ALEXER[11];         /*  message alias for EXER   */
char    ALOPER[11];         /*  message alias for OPER   */
char    ALELSE[11];         /*  message alias for OTHERS */

touchwin(main_box_win);
wrefresh(main_box_win);
wclear(main_win);
wrefresh(main_win);

addOnode(label1,label2);

wprintw(main_win,"Enter EXER or OPER => ");
wrefresh(main_win);
wget_string(main_win,option);
if(EQUALS(bl_sup(option),"EXER") ||
   EQUALS(bl_sup(option),"exer"))
   {
   strcpy(ALEXER,"ITXT_EXER");
   SET_Handler(ALEXER); /* Build up Introductory Text Sets for EXER */
   }
else if(EQUALS(bl_sup(option),"OPER") ||
   EQUALS(bl_sup(option),"oper"))
   {
   strcpy(ALOPER,"ITXT_OPER");
   SET_Handler(ALOPER); /* Build up Introductory Text Sets for OPER */
   }
else
   {
   strcpy(ALELSE,"ITXT_ELSE");
   SET_Handler(ALELSE); /* Build up Introductory Text Sets for OTHERS */
   }

wclear(main_win);
wrefresh(main_win);

SET_Handler(ALIAS);          /* Build up Main Text Sets of Message */

SWITCH = 1;

}     /*   end of create_JIN_msg   */
```

```
/* ============================================================== */
/*          SET Handler : Set Level (or Level 1)                  */
/* ============================================================== */

SET_Handler(ALIAS)        /* Build up Main Text Sets of Message */

char  *ALIAS;
{
char   CAT[4];           /*  cat */
char   SETID[9];         /*  set id */

struct {
        char  SALIAS[10];
        int   SNO;
       } SETKEY;

/*   retrieve each SetID in the order of set no */
/*   add set node */

strncpy (SETKEY.SALIAS, bl_pad(ALIAS,10), 10);
for (SETKEY.SNO=1; acckey (sets,&SETKEY) == 0; SETKEY.SNO++)
   {
   gfield (scat, CAT);
   CAT[3] = '\0';
   gfield (setid, SETID);
   SETID[8] = '\0';
   /*   Manage the set according to its type   */
   if((SETID[0] >= '0') && (SETID[0] <= '9')) /* columnar set */
      CSET_mgr(SETID, bl_sup(CAT));
   else if(EQUALS(bl_sup(SETID),"AMPN") || EQUALS(SETID,"NARR") ||
           EQUALS(SETID,"RMKS")) /*   free text set  */
      FSET_mgr(SETID, bl_sup(CAT));
   else                     /*   linear set     */
      LSET_mgr(SETID, bl_sup(CAT));
   }
}
```

```
/* ============================================================== */
/*              Columnar Set Manager : Level 1                    */
/* ============================================================== */

CSET_mgr(SETID, CAT)

char *SETID;
char *CAT;

{
wclear(main_win);
y=0;
mvwaddstr(main_win,y,0,"SET ID: ");
```

```
waddstr(main_win,SETID);
wrefresh(main_win);
addlnode('C',CAT,SETID);
FIELD_handler('C',SETID);
wrefresh(main_win);
wclear(msg_win);
wrefresh(msg_win);
}
```

```
/* ================================================================ */
/*              Free Text Set Manager : Level 1                     */
/* ================================================================ */

FSET_mgr(SETID, CAT)

char *SETID;
char *CAT;

{
wclear(main_win);
wprintw(main_win,"SET ID: %s",SETID);
wrefresh(main_win);
addlnode('F',CAT,SETID);
freef_handler();
wclear(msg_win);
wrefresh(msg_win);
}
```

```
/* ================================================================ */
/*              Linear Set Manager : Level 1                        */
/* ================================================================ */

LSET_mgr(SETID, CAT)

char *SETID;
char *CAT;

{
if(EQUALS(CAT,"O,R"))
   for(;;)
      {
      wclear(main_win);
      wmove(main_win,0,0);
```

```
      wprintw(main_win,"SET ID: %s\n",SETID);
      wrefresh(main_win);
      addlnode('L',CAT,SETID);
      FIELD_handler('L',SETID);
      if(SET_STDPPER)
         {
         SET_STDPPER = FALSE;
         break;
         }
      }
else
   {
   wclear(main_win);
   wmove(main_win,0,0);
   wprintw(main_win,"SET ID: %s",SETID);
   wrefresh(main_win);
   addlnode('L',CAT,SETID);
   FIELD_handler('L',SETID);
   }
wclear(msg_win);
wrefresh(msg_win);
}
```

```
/* ================================================================ */
/*              FIELD Handler : Field Level (or Level 2)            */
/* ================================================================ */

FIELD_handler(STYP,SETID)

char   STYP;
char   *SETID;

{
char   DFI[6];       /*  data field identifier       */
char   DUI[4];       /*  data usage identifier       */
char   FCAT[4];       /*  cat */

struct {
        char  SID[8];
        int   FNO;
       } FLDKEY;

/*   retrieve each field id in the order of field no */ /*   add field node */
strncpy (FLDKEY.SID, bl_pad(SETID, 8), 8);
for (FLDKEY.FNO=1; acckey (field,&FLDKEY) == 0; FLDKEY.FNO++)
    {
    gfield (fcat, FCAT);
    FCAT[3] = '\0';
    gfield (fname, FNAME);
    FNAME[40] = '\0';
    gfield (fdfi, DFI);
    DFI[5] = '\0';
```

```
    gfield (fdui, DUI);
    DUI[3] = '\0';
    switch(STYP)
       {
       case 'C':                /*   columnar set   */
                 gfield (fcol, SCOL_J);
                 SCOL_J[3] = '\0';
                 DFI_handler('C',DFI,DUI,SETID,FLDKEY.FNO,FCAT);
                 break;
       case 'L':                /*   linear set     */
                 DFI_handler('L',DFI,DUI,SETID,FLDKEY.FNO,FCAT);
                 break;
       }
    }
    switch(STYP)
         {
          case 'C':
                    addscr_csf(ptr1->uval.cschild);
                    addscr_csd();
                    break;
          case 'L':
                    addscr_lsf(ptr1->uval.lschild);
                    addscr_lsd();
                    break;
         }
}
```

```
/* ================================================================== */
/*              DFI Handler : DFI Level (or Level 3)                  */
/* ================================================================== */

DFI_handler(KIND,FDFI,FDUI,SETID,FDNO,FCAT)

char   KIND;
char   *FDFI, *FDUI;
char   *SETID;
char   *FCAT;
int    FDNO;

{
char   LRJ[2];
char   NOTYP[11];
struct {
        char DFI[6];
        char DUI[4];
       } DFIKEY;

/*  retrieve NO-TYPE from dfis relation and add it to the field  */
strcpy (DFIKEY.DFI, FDFI);
strcpy (DFIKEY.DUI, FDUI);
if (acckey (dfis,&DFIKEY))
   {
```

```
    wmove(msg_win, 2, 0);
    wprintw(msg_win,"%-80s", "*** dfi not found !!!");
    wrefresh(msg_win);
   }
gfield (dlrj, LRJ);
LRJ[1] = '\0';
gfield (dformat, NOTYP);
NOTYP[10] = '\0';
switch(KIND)
   {
   case 'C': gfield(dcolhdr,CHDR);
             CHDR[23] = '\0';
             add_csf_node(bl_sup(FCAT),bl_sup(FNAME),bl_sup(CHDR),
                          bl_sup(SCOL_J),bl_sup(LRJ), bl_sup(NOTYP));
             break;
   case 'L': gfield(dfdesc,FDESC);
             FDESC[7] = '\0';
             add_ls_node(bl_sup(FDNO),bl_sup(FCAT),bl_sup(FNAME),
                         bl_sup(FDESC), bl_sup(LRJ),bl_sup(NOTYP));
             break;
   }  /* end of switch */
}  /* end of DFI_handler */
```

```
/* ========================================================================*/
/*              Free Text Input Handler                                    */
/* ========================================================================*/

freef_handler()

{
char   **freetext;

freetext=(char **)malloc((MAXLINES+1) * (sizeof(char *)));
readtxt(freetext);
if(EMPTY_TEXT)
    dellnode(ptr1);
else
    add_ts_node(freetext);
}
```

```
/* ========================================================================= */
/*     Read a free text set in                                               */
/* ========================================================================= */

readtxt(longtext)
char   **longtext;

{
   register int   i=0;
   register int   len=0;
   register int   DONE=FALSE;
   register int   line, column;
   char   tempbuf[81];

   EMPTY_TEXT = FALSE;
   getyx(main_win,y,x);
   if (y < MWLINES-1)
      wmove(main_win,y+1,0);
   else
      {
       waddch(main_win,'\n');
       wmove(main_win,y,0);
      }
   wrefresh(main_win);
   print_imsg(" *** Enter free text and terminate with / *** ");
   while (!DONE)
     {
      longtext[i] = (char *) 0;

      wget_string(main_win,tempbuf);
      getyx(main_win,y,x);
      if (y < MWLINES-1)
         wmove(main_win,y+1,0);
      else
         {
          waddch(main_win,'\n');
          wmove(main_win,y,0);
         }
      wrefresh(main_win);
      len=strlen(tempbuf);
      if (EQUALS(tempbuf+len-1, "/"))
         {
          DONE = TRUE;
          tempbuf[len-1] = '\0';
          if((i == 0) && (strlen(bl_sup(tempbuf))) == 0)
             EMPTY_TEXT = TRUE;
         }
      if (DONE && (strlen(tempbuf) == 0))
         continue;
      else
         {
          longtext[i] = malloc(len+1);
          strcpy(longtext[i],tempbuf);
          i++;
         }
     }
   longtext[i] = (char *) 0;
   wclear(msg_win);
   wrefresh(msg_win);
```

```
}

/* ================================================================*/
/*              Validity Checker for data format and value         */
/* ================================================================*/

VALID(DVAL,NOTYP)

char    *DVAL;
char    *NOTYP;

{
/*  check if DVAL is valid                                          */
/*  i.e check if the value of DVAL is within the valid range        */
/*            and its type is valid                                 */

return(1);
}

/* ================================================================ */
/*    Draw a frame for a window                                     */
/* ================================================================ */

int Frame(win)
WINDOW *win;
{
   wstandout(win);
   wmove(win,0,0);
   wprintw(win, "%80c", ' ');
   for (y=1; y<win->_maxy-1; y++)
      {
       wmove(win, y, 0);
       wprintw(win, "  ");
       wmove(win, y, win->_maxx-2);
       wprintw(win, "  ");
      }
   wmove(win, win->_maxy-1, 0);
   wprintw(win, "%80c", ' ');
   wstandend(win);
}
```

```
/* ================================================================== */
/*    Put a header at top of the frame                                */
/* ================================================================== */

int header(win, str,rvid)

WINDOW *win;
char   *str;
int    rvid;
{
   if(rvid)
      wstandout(win);
   wmove(win,0, (win->_maxx-strlen(str))/2);
   wprintw(win, "%s",str);
   if(rvid)
      wstandend(win);
}
```

```
/* ================================================================== */
/*    Put a footnote at bottom of the frame                           */
/* ================================================================== */

int footnote(win, str,rvid)

WINDOW *win;
char   *str;
int    rvid;
{
   if(rvid)
      wstandout(win);
   wmove(win,win->_maxy-1, (win->_maxx-strlen(str))/2);
   wprintw(win, "%s",str);
   if(rvid)
      wstandend(win);
}
```

```
/* ================================================================== */
/*    clear sceen and end if user presses DEL (SIGINT)                */
/* ================================================================== */

int die()
```

```
{
    clear();
    refresh();
    endwin();
    exit(1);
}
```

```
#include "jms.h"

/*  Global variables for these functions  */

char  DTYPE[5];   /* Data Type as a mixture of A, N, B, and S characters */
int   DMIN, DMAX;

/* ************************************************************************ */
/*   file name: list.c                                                      */
/*                                                                          */
/*   This file includes list handling routines                              */
/*                                                                          */
/* ************************************************************************ */

/* ------------------------------------------------------------------------ */
/*     Part I:  Routines to add nodes and lists                             */
/* ------------------------------------------------------------------------ */


/* ======================================================================== */
/*      Add message header into level 0 list                                */
/* ======================================================================== */

addOnode(str1,str3)

char str1[],str3[];

      {
      list0 *node0;
      node0=(list0 *)malloc(sizeof(list0));

      if (head == NULL) {
          head = node0;
          node0->Prev = NULL;
          node0->Next = NULL;
          node0->schild = NULL;
          strcpy(node0->mtag1,str1);
          strcpy(node0->mtag2,str3);
          ptr0=node0;
      }  else {
          node0->Prev = ptr0;
          node0->Next = NULL;
          node0->schild = NULL;
          ptr0->Next = node0;
          strcpy(node0->mtag1,str1);
          strcpy(node0->mtag2,str3);
          ptr0=node0;
      }

      }
```

```
/* ================================================================== */
/*       Add a SET node in level 1 list                               */
/* ================================================================== */

add1node(SETTYP,CAT,SETID)

char SETTYP;
char CAT[];
char SETID[];

      {
      list1 *node1;
      node1=(list1 *)malloc(sizeof(list1));

      /*  If the node is the first node of the parent node "ptr0" */
      if (ptr0->schild == NULL)
          {
          ptr0->schild = node1;
          node1->Prev = NULL;
          }
      else
          {
          node1->Prev = ptr1;
          ptr1->Next = node1;
          }
      node1->parent = ptr0;
      node1->Next = NULL;
      switch(SETTYP)
            {
            case 'C':    /*   columnar set   */
                         (*node1).uval.cschild = NULL;
                         break;
            case 'F':    /*   free text set   */
                         (*node1).uval.tschild = NULL;
                         break;
            case 'L':    /*   linear set   */
                         (*node1).uval.lschild = NULL;
                         break;
            }
      node1->settyp = SETTYP;
      strcpy(node1->setcat,bl_sup(CAT));
      strcpy(node1->SetID,bl_sup(SETID));
      ptr1=node1;
      }
```

```
/* ================================================================== */
/*       Add a field node in level 2 list of the linear set           */
/* ================================================================== */

add_ls_node(FNO,FCAT,FNAME,fdsc,lrj,format)
```

```
int  FNO;
char FCAT[],FNAME[],fdsc[],lrj[],format[];

{
lsnode *ptrls;
ptrls=(lsnode *)malloc(sizeof(lsnode));

/*  If the node is the first node of the parent node "ptrl" */
if ((*ptrl).uval.lschild == NULL)
    {
    (*ptrl).uval.lschild = ptrls;
    ptrls->Prev = NULL;
    }
else
    {
    ptrls->Prev = taills;
    taills->Next = ptrls;
    }
ptrls->parent = ptrl;
ptrls->Next = NULL;
ptrls->fchild = NULL;
ptrls->fdno = FNO;
strcpy(ptrls->fdcat,FCAT);
strcpy(ptrls->fdname,FNAME);
strcpy(ptrls->fdesc,fdsc);
strcpy(ptrls->j,lrj);
strcpy(ptrls->notype,format);

/* Partition notype into MIN, MAX, and TYPE */
   pars_form(ptrls->notype);
   strcpy(ptrls->dtype,DTYPE);
   ptrls->dmin = DMIN;
   ptrls->dmax = DMAX;

taills=ptrls;
}









/* ================================================================== */
/*        Add a field node in level 2 list of the columnar set        */
/* ================================================================== */

add_csf_node(FCAT,FNAME,field,column,lrj,format)

char FCAT[],FNAME[],field[],column[],lrj[],format[];
{
csfnode *ptrcsf;
ptrcsf=(csfnode *)malloc(sizeof(csfnode));

/*  If the node is the first node of the parent node "ptrl" */
if ((*ptrl).uval.cschild == NULL)
   {
   (*ptrl).uval.cschild = ptrcsf;
```

```
   ptrcsf->Prev = NULL;
   }
else
   {
   ptrcsf->Prev = tailcsf;
   tailcsf->Next = ptrcsf;
   }
ptrcsf->parent = ptr1;
ptrcsf->Next = NULL;
ptrcsf->fchild = NULL;
strcpy(ptrcsf->fdcat,FCAT);
strcpy(ptrcsf->fdname,FNAME);
strcpy(ptrcsf->colhdr,field);
strcpy(ptrcsf->colpos,column);
strcpy(ptrcsf->j,lrj);
strcpy(ptrcsf->notype,format);

/* Partition notype into MIN, MAX, and TYPE */
   pars_form(ptrcsf->notype);
   strcpy(ptrcsf->dtype,DTYPE);
   ptrcsf->dmin = DMIN;
   ptrcsf->dmax = DMAX;

tailcsf=ptrcsf;
}
```

```
/* ================================================================== */
/*    Add a data node as the sublevel of the corresponding field node  */
/* ================================================================== */

add_csd_node(SCAT)

char  *SCAT;
{
register int    k,j;
int    n;
int    LIVE = FALSE;
int    FIRST = TRUE;
char   str[31];
csfnode  *fptr;
csdnode  **dptr, **node;

fptr = (*ptr1).uval.cschild; /* point to the head of field list */
for(j=0; fptr; j++)  /* count the fields */
    fptr = fptr->Next;

/* allocate storage for pointers to field data nodes */
node = (csdnode **)malloc(j*sizeof(csdnode *)); /* data nodes */
dptr = (csdnode **)malloc(j*sizeof(csdnode *)); /* tail ptr   */
str[0] = NULL;
while(!EQUALS(str,"//"))  /* loop till user enters "//" - end of set */
     {
```

```
    fptr = ptr1->uval.cschild; /* point to the head of field list */
    addscr_csline(fptr);
    /* loop through the field nodes gathering data till user says quit */
    for(k=0; ((k<j) && (fptr)); k++)
        {
        wclear(msg_win);
        wrefresh(msg_win);
        n = atoin(fptr->colpos,strlen(fptr->colpos)-1);
        wmove(main_win,y,n);
        wrefresh(main_win);
        wget_string(main_win,str);
        while(EQUALS(bl_sup(str),"//") && EQUALS(SCAT,"M") && FIRST)
              {
               wclear(msg_win);
               mvwaddstr(msg_win,0,0,"The set is MANDATORY.");
               mvwaddstr(msg_win,1,0,"Reenter ");
               waddstr(msg_win,fptr->fdname);
               wrefresh(msg_win);
               wmove(main_win,y,n);
               wrefresh(main_win);
               wget_string(main_win,str);
              }
        FIRST = FALSE;
        if(EQUALS(bl_sup(str),"\0"))
              strcpy(str,"-");
        if(k==0 && EQUALS(str,"//"))
              break;
        while(k != 0 && EQUALS(bl_sup(str),"//"))
              {
               wclear(msg_win);
               mvwaddstr(msg_win,0,0,"The set is MANDATORY.");
               mvwaddstr(msg_win,1,0,"Reenter ");
               waddstr(msg_win,fptr->fdname);
               wrefresh(msg_win);
               wmove(main_win,y,n);
               wrefresh(main_win);
               wget_string(main_win,str);
              }
        node[k] = (csdnode *)malloc(sizeof(csdnode)); /*data space*/
        if(fptr->fchild == NULL) /* first field node in list */
                  {
                  fptr->fchild = node[k];
                  node[k]->Prev = NULL;
                  node[k]->Next = NULL;
                  node[k]->parent = fptr;
                  strcpy(node[k]->csdata,str);
                  dptr[k] = node[k];
                  }
        else      /* all following field nodes */
                  {
                  node[k]->Prev = dptr[k];
                  node[k]->Next = NULL;
                  node[k]->parent = dptr[k]->parent;
                  dptr[k]->Next = node[k];
                  strcpy(node[k]->csdata,str);
                  dptr[k] = node[k];
                  }
        fptr=fptr->Next;
        } /* end of for */
    } /* end of while */
/*  Test if the columnar set has some data in it  */
```

```
fptr = ptr1->uval.cschild;
while(fptr)
    {
     if(fptr->fchild == NULL)
        fptr = fptr->Next;
     else
        {
        LIVE = TRUE;
        break;
        }
    }
if(!LIVE)
   delnode(ptr1);
}
```

```
/* ================================================================== */
/*        Add a node for the free text set                            */
/* ================================================================== */

add_ts_node(freetxt)
char **freetxt;

     {
     tsnode *ptrts;
     ptrts=(tsnode *)malloc(sizeof(tsnode));

         (*ptr1).uval.tschild = ptrts;
         ptrts->parent = ptr1;
         ptrts->tstext = freetxt;
     }
```

```
/* ================================================================== */
/*  Parse Lower and Upper Bounds and TYPE out of NO_TYPE */
/* ================================================================== */

pars_form(FORM)

char  *FORM;
{
char   LB[4], UB[4];
int    i, j, k, l;

for (i=0; FORM[i] == ' '; i++)  ;
```

```
for (i, j=0; FORM[i] >= '0' && FORM[i] <= '9'; i++, j++)
    LB[j] = FORM[i];
LB[j] = '\0';
for (i; FORM[i] == ' '; i++)  ;
if (FORM[i] == '-')
    {
     for (i++; FORM[i] == ' '; i++)  ;
     for (i, k=0; FORM[i] >= '0' && FORM[i] <= '9'; i++, k++)
         UB[k] = FORM[i];
     UB[k] = '\0';
     for (i; FORM[i] == ' '; i++)  ;
    }
else
     strcpy(UB,LB);
for (i, l=0; FORM[i] >= 'A' && FORM[i] <= 'Z'; i++, l++)
    DTYPE[l] = FORM[i];
DTYPE[l] = '\0';
DMIN = atoi(LB);
DMAX = atoi(UB);
}
```

```
/* ------------------------------------------------------------------ */
/*     Part II: Routines to delete nodes and lists                    */
/* ------------------------------------------------------------------ */

/* ================================================================== */
/*    indirectly  recursive with del2list                             */
/*    delete the linear set node pointed to by ptrl2 and              */
/*    all its children                                                */
/* ================================================================== */

dellsnode(ptr)
lsnode *ptr;

      {
      if(ptr->fchild != NULL)
         dellslist(ptr->fchild);
      if(ptr->Prev == NULL)
         if(ptr->Next == NULL)
            {
            ptr->parent->uval.lschild = NULL;
            dellnode(ptr->parent);
            }
         else
            {
            ptr->parent->uval.lschild = ptr->Next;
            ptr->Next->Prev = NULL;
            }
      else
         {
         ptr->Prev->Next= ptr->Next;
         if(ptr->Next != NULL)
```

```
            ptr->Next->Prev = ptr->Prev;
         }
      free(ptr);
      }
```

```
/* ========================================================================= */
/*   delete the linear set list followed by the node pointed to              */
/*   by ptr12 and all the sublists                                           */
/* ========================================================================= */

dellslist(ptr2)
lsnode *ptr2;

      {
      if(ptr2->Next != NULL)
         dellslist(ptr2->Next);
      dellsnode(ptr2);
      }
```

```
/* ========================================================================= */
/*   delete the columnar set list followed by the node pointed to            */
/*   by ptr12 and all the sublists                                           */
/* ========================================================================= */

delcsflist(ptr2)
csfnode *ptr2;

      {
      if(ptr2->Next != NULL)
         delcsflist(ptr2->Next);
      delcsfnode(ptr2);
      }
```

```
/* ========================================================================= */
/*    indirectly  recursive with del2list                                    */
/*    delete the columnar set node pointed to by ptr12 and                   */
/*    all its children                                                       */
/* ========================================================================= */

delcsfnode(ptr)
csfnode *ptr;

      {
      if(ptr->fchild != NULL)
         delcsdlist(ptr->fchild);
      if(ptr->Prev == NULL)
         if(ptr->Next == NULL)
            ptr->parent = NULL;
         else {
            ptr->parent->uval.cschild = ptr->Next;
            ptr->Next->Prev = NULL;
         }
      else {
         ptr->Prev->Next= ptr->Next;
         if(ptr->Next != NULL)
            ptr->Next->Prev = ptr->Prev;
      }
      free(ptr);
      }
```

```
/* ========================================================================= */
/*   delete the columnar set data list followed by the node pointed to       */
/*   by ptr12 and all the sublists                                           */
/* ========================================================================= */

delcsdlist(ptr)
csdnode *ptr;

      {
      if(ptr->Next != NULL)
         delcsflist(ptr->Next);
      delcsdnode(ptr);
      }
```

```
/* ========================================================================= */
/*    indirectly  recursive with del2list                                    */
/*    delete the columnar set data node pointed to by ptr12 and              */
/*    all its children                                                       */
/* ========================================================================= */

delcsdnode(ptr)
csdnode *ptr;

      {
      if(ptr->Prev == NULL)
         if(ptr->Next == NULL)
            ptr->parent = NULL;
         else {
            ptr->parent->fchild = ptr->Next;
            ptr->Next->Prev = NULL;
         }
      else {
         ptr->Prev->Next= ptr->Next;
         if(ptr->Next != NULL)
            ptr->Next->Prev = ptr->Prev;
      }
      free(ptr);
      }
```

```
/* ========================================================================= */
/*    indirectly  recursive with del2list                                    */
/*    delete the free text set node pointed to by ptr12 and                  */
/*    all its children                                                       */
/* ========================================================================= */

deltsnode(ptr)
tsnode *ptr;

      {
      ptr->parent->uval.tschild = NULL;
      free(ptr);
      }
```

```
/* ========================================================================= */
/*     Delete all the SET nodes of a message from the level 1 list           */
/* ========================================================================= */

dellist(ptr1)
list1 *ptr1;

      {

      if(ptr1->Next != NULL)
         dellist(ptr1->Next);

      delnode(ptr1);

      }
```

```
/* ========================================================================= */
/*      Delete all the messages from the multi-level list                    */
/* ========================================================================= */

del0list(ptr10)
list0 *ptr10;

      {
      if(ptr10->Next != NULL)
         del0list(ptr10->Next);
      del0node(ptr10);
      }
```

```
/* ========================================================================= */
/*      Delete the whole message from the multi-level list                   */
/* ========================================================================= */

del0node(ptr10)
list0 *ptr10;

      {
      list0 *tptr0;
      tptr0 = ptr10;
      if(ptr10->schild != NULL)
         dellist(ptr10->schild);
```

```
      if(ptr10->Prev == NULL)
         {
         if(ptr10->Next == NULL)
            head = NULL;
         else
            {
            head = ptr10->Next;
            ptr10->Next->Prev = NULL;
            ptr10 = ptr10->Next;
            }
         }
      else
         {
         ptr10->Prev->Next= ptr10->Next;
         ptr10->Next->Prev = ptr10->Prev;
         ptr10 = ptr10->Prev;
         }

      free(tptr0);

      }
```

```
/* ================================================================== */
/*       Delete a SET node in the corresponding level 1 list          */
/* ================================================================== */

dellnode()

      {
       list1   *tptr1;
       tptr1 = ptr1;
       switch(ptr1->settyp)
             {
              case 'C':    /*   columnar set   */
                       if((*ptr1).uval.cschild != NULL)
                           delcsflist((*ptr1).uval.cschild);
                       break;
              case 'F':    /*   free text set   */
                       if((*ptr1).uval.tschild != NULL)
                           deltsnode((*ptr1).uval.tschild);
                       break;
              case 'L':    /*   linear set   */
                       if((*ptr1).uval.lschild != NULL)
                           dellslist((*ptr1).uval.lschild);
                       break;
             }
      if(ptr1->Prev == NULL)
         {
         if(ptr1->Next == NULL)
            {
            ptr0->schild = NULL;
            }
```

```
         else
            {
            ptr1->parent->schild = ptr1->Next;
            ptr1->Next->Prev = NULL;
            ptr1 = ptr1->Next;
            }
         }
      else
         {
         ptr1->Prev->Next= ptr1->Next;
         if(ptr1->Next != NULL)
            ptr1->Next->Prev = ptr1->Prev;
         ptr1 = ptr1->Prev;
         }

      free(tptr1);

      }
```

```
#include "jms.h"


/* ---------------------------------------------------------------- */
/*   file name: scr.c                                               */
/*                                                                  */
/*   This file contains screen I/O handling routines                */
/* ---------------------------------------------------------------- */

/* ================================================================ */

/* ================================================================ */

wget_string(win,str)

WINDOW *win;
char *str;
{
int c;
int i = 0;

while((c=getchar()) != '\r' && (c != DEL))
    {
    switch(c)
       {
        case '\b':
                  if (i > 0)
                     {
                      i--;
                      waddch(win,c);
                      waddch(win,' ');
                      waddch(win,c);
                     }
                  else
                     {
                      putchar (BEEP);
                     }
                  wrefresh(win);
                  break;
        case ESC:
                  getyx(main_win,y,x);
                  command_handler();
                  break;
        default:
                  waddch(win,c);
                  wrefresh(win);
                  str[i++] = c;
                  break;
       }
    }
if (c == DEL)
   ex_handler(1);
/* else if (c == '\r') waddch (win, '\n'); */
else
   str[i] = '\0';
}
```

```
/* ========================================================================== */

/* ========================================================================== */
wgetchar(win)
WINDOW *win;
{
int c;

switch(c = wgetch(win))
    {
     case DEL:
               ex_handler(1);
               break;
     case ESC:
               return (c);
               break;
     case '\r':
               return (c);
               break;
     default:
               waddch(win,c);
               wrefresh(win);
               return (c);
               break;
    }
}
```

```
/* ========================================================================== */

/* ========================================================================== */
char *readstr(str)

char *str;
{
char c;
int i = 0;

while((c=getchar()) != '\n' && c != EOF)
     {
     str[i++] = c;
     }
str[i] = '\0';
return(str);
}
```

```
/* ========================================================== */
/*  Print a message relevant to user input                    */
/* ========================================================== */

print_imsg(str)

char *str;
{
int line, column;
getyx(main_win,line,column);
wclear(msg_win);
wprintw(msg_win,"%s", str);
wrefresh(msg_win);
wmove(main_win,line,column);
wrefresh(main_win);
}
```

```
/* ========================================================== */
/*  Print a message relevant to command input                 */
/* ========================================================== */

print_cmsg(str)

char *str;
{
int line, column;
getyx(command_win,line,column);
wclear(msg_win);
wprintw(msg_win,"%s", str);
wrefresh(msg_win);
wmove(command_win,line,column);
wrefresh(command_win);
}
```

```
/* ========================================================== */
/*   Read in a field if the input is within the valid range   */
/* ========================================================== */

wget_field(win,str,DTYPE,DMIN,DMAX,SETTYPE,FCAT)

WINDOW *win;
```

```
char *str;
char *DTYPE, *FCAT;
int  SETTYPE;
int  DMIN, DMAX;

{
int    LINE, COL;
int c;
int i = 0;
int DONE = 0;

getyx(main_win,LINE,COL);

while(((c=wgetch(win)) != DEL) && (c != '/'))
    {
    switch(c)
       {
        case '\b':
                  if (i > 0)
                     {
                      i--;
                      waddch(win,c);
                      waddch(win,'_');
                      waddch(win,c);
                     }
                  else
                      putchar (BEEP);
                  wrefresh(win);
                  break;
        case ESC:
                  getyx(main_win,y,x);
                  command_handler();
                  break;
                  break;
        case '\r':
                  str[i] = '\0';
                  if(!EQUALS(FCAT,"M") && EQUALS(bl_sup(str),"\0"))
                    DONE = 1;
                  else if(i < DMIN)
                   {
                   getyx(win,LINE,COL);
                   wclear(msg_win);
                   wmove(msg_win,0,0);
                   wprintw(msg_win,
                           "WARNING! Range and Type of Input is %d - %d %s",
                           DMIN,DMAX,DTYPE);
                   putchar (BEEP);
                   wrefresh(msg_win);
                   wmove(win,LINE,COL);
                   wrefresh(win);
                   }
                  else
                    DONE = 1;
                  break;
        default:
                  if((i+1) > DMAX)
                    {
                    getyx(win,LINE,COL);
                    wclear(msg_win);
                    wmove(msg_win,0,0);
                    wprintw(msg_win,
```

```
                        "WARNING! Range and Type of Input is %d - %d %s",
                        DMIN,DMAX,DTYPE);
                    putchar (BEEP);
                    wrefresh(msg_win);
                    wmove(win,LINE,COL);
                    wrefresh(win);
                    }
                  else
                    {
                    waddch(win,c);
                    wrefresh(win);
                    str[i++] = c;
                    }
                  break;
        }
     if(DONE)  break;
     }
if (c == DEL)
   ex_handler(1);
else if (c == '/')
   {
   str[i++] = '/';
   str[i] = '\0';
   }
/* else if (c == '\r') waddch (win, '\n'); */
}
```

```
/* ========================================================== */
/*  Exception Handler                                         */
/* ========================================================== */

ex_handler(i)
int   i;
{
switch(i)
    {
     case 1:   /* Terminate the program */
               clear();
               refresh();
               endwin();
               exit(1);
               break;
     default:
               break;
    }
```

```
/* ------------------------------------------------------------------ */
/*   file name: util.c                                                */
/*                                                                    */
/*   This file contains utilities which may be used for any program   */
/* ------------------------------------------------------------------ */

/* ================================================================== */
/*     Convert the first k substring of "str" to integer              */
/* ================================================================== */

int atoin(str,k)
char str[];
int  k;

      {
      int i,n;
      n=0;
      for(i=0; i<k; ++i)
          n = 10 * n + str[i] - '0';
      return(n);
      }
```

```
/* ================================================================== */
/*     Convert char c to upper case; ASCII only                       */
/* ================================================================== */

char upchar(c)
char c;
{
 if(c >= 'a' && c <= 'z')
    return(c+'A'-'a');
 else
    return(c);
}
```

```
/* ================================================================== */
/*     Convert str s to upper case; ASCII only                        */
/* ================================================================== */

char *upstr(s)
char *s;
```

```
{
 int  i;
 for(i=0; s[i]; i++)
     s[i] = upchar(s[i]);
 return(s);
}
```

```
/* ========================================================================= */
/*     suppress blanks at the end of string                                  */
/* ========================================================================= */

char *bl_sup(str)
char   *str;
{
int   k;
k=strlen(str);
while((k > 0) && (str[k-1] == ' '))
     k--;
str[k] = '\0';
return(str);
}
```

```
/* ========================================================================= */
/*     pad blanks to make the string length n                                */
/* ========================================================================= */

char *bl_pad(str,n)
char   *str;
int   n;
{
int   k;
k=strlen(str);
while(k < n)
     {
      str[k] = ' ';
      k++;
     }
str[k] = '\0';
return(str);
}
```

```
/* ========================================================================== */
/*     pad blanks on the left of the input string to make the length n  */
/* ========================================================================== */

char *bl_pad_front(str,n)
char   *str;
int   n;
{
int   i, j, k;
char  temp[80];

k=strlen(str);

for (i=0; i < n-k; i++)
   temp[i] = ' ';

for (i=n-k, j=0; i < n; i++, j++)
   temp[i] = str[j];

strncpy (str, temp, n);

str[n] = '\0';

return(str);
}
```

AMS

This section contains the functions used only by the batch oriented JINTACCS message preparation program (AMS).

```
UCOMP=$(UNIFY)/../bin/ucc
ULOAD=$(UNIFY)/../bin/uld

clean:
        rm -f ams *.o

ams: ams.h ams.o add.o del.o disp.o form.o\
     conv.o inp.o blanks.o
        $(ULOAD) ams ams.o add.o del.o disp.o form.o\
           conv.o inp.o blanks.o
        strip ams

ams.o:  ams.c ams.h ../../def/file.h sysdef.h ams.e
        $(UCOMP) -c -Mm ams.c

add.o:  add.c ams.h
        cc -c -Mm add.c

del.o:  del.c ams.h
        cc -c -Mm del.c

disp.o: disp.c ams.h
        cc -c -Mm disp.c

form.o: form.c ams.h
        cc -c -Mm form.c

blanks.o: blanks.c ams.h
        cc -c -Mm blanks.c

conv.o: conv.c ams.h
        cc -c -Mm conv.c

inp.o: inp.c ams.h
        cc -c -Mm inp.c
```

```
/* extern declarations for global variables */

extern int    col;       /*  number of columns for indentation   */
extern int    DEAD;      /*  Indicator for no text */
extern int    getout;    /*  Indicator */

extern char   ALIAS[];
extern char   MNO[];
extern char   TITLE[];
extern char   FNAME[];       /*  data field identifier       */
extern char   FDESC[];
extern char   SCOL_J[];      /*  start column                */
extern char   CHDR[];

extern list0    *head;
extern list0    *ptr0;
extern list1    *ptr1;

extern lsnode      *ptrpos;
extern lsnode      *taills;
extern csfieldnode *tailcsf;
extern csdatanode  *tailcsd;
extern tsnode      *tailts;
```

```
/* constants and definitions for message entry system */

#include  <stdio.h>
#include  <curses.h>
#include  <signal.h>

#define   EQUALS     !strcmp
#define   MAXLINES   20

struct lst0_str
          {
          char   mtag1[16],mno[5];
          char   mtag2[7],mid[11];
          struct lst0_str  *prev, *next;
          struct lst1_str  *schild;
          } ;




struct lst1_str
          {
          char   settyp;
          char   setid[9];
          struct lst0_str  *parent;
          struct lst1_str  *prev, *next;
          union {
                  struct lsnode_str  *lschild;
                  struct csnode_str  *cschild;
                  struct tsnode_str  *tschild;
                } uval;
          } ;




struct lsnode_str
          {
          char   fdname[41];
          char   fdesc[9];
          char   fval[25];
          char   j[2];
          char   notype[11];
          struct lst1_str  *parent;
          struct lsnode_str  *prev, *next, *fchild;
          } ;
```

```
struct csnode_str
        {
        char    fdname[41];
        char    colhdr[25];
        char    colpos[4];
        char    j[2];
        char    notype[11];
        struct lstl_str  *parent;
        struct csnode_str  *prev, *next;
        struct csdata_str  *fchild;
        } ;
```

```
struct csdata_str
        {
        char    csdata[25];
        struct csnode_str  *parent;
        struct csdata_str  *prev, *next;
        } ;
```

```
struct tsnode_str
        {
        char    **tstext;
        struct lstl_str  *parent;
        } ;
```

```
typedef struct lst0_str list0;
typedef struct lstl_str listl;
typedef struct lsnode_str lsnode;
typedef struct csnode_str csfieldnode;
typedef struct csdata_str csdatanode;
typedef struct tsnode_str tsnode;
```

```
/* C library functions */

char *malloc();
int   strcmp();
int   strlen();
char *strcpy();
char *strcat();

/* user defined functions */

int    atoin();
char   *upchar();
char   *upstr();
char   *bl_sup();
char   *bl_pad();
int    convert();
int    convlslist();
int    convcsflist();
int    convcsdlist();
int    convtsnode();
int    display();
int    dsp_lin_set();
int    dsp_col_set();
int    dsp_free_set();
int    dsp_csdata();
int    readtxt();
int    SET_handler();
int    FIELD_handler();
int    DFI_handler();
int    VALID();
int    freef_handler();
int    addOnode();
int    insOnode();
int    addlnode();
int    add_ls_node();
int    add_csf_node();
int    add_csd_node();
int    add_ts_node();
int    delOnode();
int    dellnode();
int    dellsnode();
int    delOlist();
int    delllist();
int    delcsflist();
int    delcsfnode();
int    delcsdnode();
int    deltsnode();
int    ins_ls_data();

#ifndef MAIN
#include "ams.e"
#endif

#ifdef ONYX
#define acckey(X,Y) access(X,Y)
#endif
```

```
#include "ams.h"

/* functions to add nodes and lists */

/* ================================================================== */
/*      Add message header into level 0 list                          */
/* ================================================================== */

addOnode(str1,str3)
char str1[],str3[];

      {
      list0 *node0;
      node0=(list0 *)malloc(sizeof(list0));

      if (head == NULL) {
          head = node0;
          node0->prev = NULL;
          node0->next = NULL;
          node0->schild = NULL;
          strcpy(node0->mtag1,str1);
          strcpy(node0->mtag2,str3);
          ptr0=node0;
      }  else {
          node0->prev = ptr0;
          node0->next = NULL;
          node0->schild = NULL;
          ptr0->next = node0;
          strcpy(node0->mtag1,str1);
          strcpy(node0->mtag2,str3);
          ptr0=node0;
      }

      }









/* ================================================================== */
/*      Add message header into level 0 list                          */
/* ================================================================== */

insOnode(str2,str4)
char str2[],str4[];

      {
          strcpy(ptr0->mno,str2);
          strcpy(ptr0->mid,str4);
      }
```

```
/* ========================================================================= */
/*       Add a SET IDENT in the corresponding level 1 list                   */
/* ========================================================================= */

addlnode(SETTYP,SETID)

char SETTYP;
char SETID[];

      {
      list1 *node1;
      node1=(list1 *)malloc(sizeof(list1));

      /*  If the node is the first node of the parent node "ptr0" */
      if (ptr0->schild == NULL)
          {
          ptr0->schild = node1;
          node1->prev = NULL;
          }
      else
          {
          node1->prev = ptr1;
          ptr1->next = node1;
          }
      node1->parent = ptr0;
      node1->next = NULL;
      switch(SETTYP)
            {
            case 'C':     /*   columnar set   */
                          (*node1).uval.cschild = NULL;
                          break;
            case 'F':     /*   free text set   */
                          (*node1).uval.tschild = NULL;
                          break;
            case 'L':     /*   linear set   */
                          (*node1).uval.lschild = NULL;
                          break;
            }
      node1->settyp = SETTYP;
      strcpy(node1->setid,bl_sup(SETID));
      ptr1=node1;

      }
```

```
/* ========================================================================= */
/*       Add a node for a field of the linear set                            */
/* ========================================================================= */

add_ls_node(fname,fdsc,lrj,format)
```

```
char fname[],fdsc[],lrj[],format[];

      {
      lsnode *ptrls;
      ptrls=(lsnode *)malloc(sizeof(lsnode));

      /*  If the node is the first node of the parent node "ptrl" */
      if ((*ptrl).uval.lschild == NULL)
          {
          (*ptrl).uval.lschild = ptrls;
          ptrls->prev = NULL;
          }
      else
          {
          ptrls->prev = taills;
          taills->next = ptrls;
          }
      ptrls->parent = ptrl;
      ptrls->next = NULL;
      ptrls->fchild = NULL;
      strcpy(ptrls->fdname,fname);
      strcpy(ptrls->fdesc,fdsc);
      strcpy(ptrls->j,lrj);
      strcpy(ptrls->notype,format);
      taills=ptrls;
      }
```

```
/* ================================================================== */
/*        Insert field value into the linear set node                 */
/* ================================================================== */

ins_ls_data(datastr)

char datastr[];

{
strcpy(taills->fval,datastr);
}
```

```
/* ================================================================== */
/*        Add a node for a field of the columnar set                  */
/* ================================================================== */
```

```
add_csf_node(fname,field,column,lrj,format)

char fname[],field[],column[],lrj[],format[];
{
csfieldnode *ptrcsf;
ptrcsf=(csfieldnode *)malloc(sizeof(csfieldnode));

/*  If the node is the first node of the parent node "ptrl" */
if ((*ptrl).uval.cschild == NULL)
   {
   (*ptrl).uval.cschild = ptrcsf;
   ptrcsf->prev = NULL;
   }
else
   {
   ptrcsf->prev = tailcsf;
   tailcsf->next = ptrcsf;
   }
ptrcsf->parent = ptrl;
ptrcsf->next = NULL;
ptrcsf->fchild = NULL;
strcpy(ptrcsf->fdname,fname);
strcpy(ptrcsf->colhdr,field);
strcpy(ptrcsf->colpos,column);
strcpy(ptrcsf->j,lrj);
strcpy(ptrcsf->notype,format);
tailcsf=ptrcsf;
}
```

```
/* ================================================================== */
/*    Add a data node as the sublevel of the corresponding field node  */
/* ================================================================== */

add_csd_node(SCAT)

char  *SCAT;
{
register int    k,j;
int    LIVE = FALSE;
int    FIRST = TRUE;
char   str[31];
csfieldnode  *fptr;
csdatanode   **dptr, **node;

fptr = (*ptrl).uval.cschild; /* point to the head of field list */
for(j=0; fptr; j++)  /* count the fields */
    fptr = fptr->next;

/* allocate storage for pointers to field data nodes */
node = (csdatanode **)malloc(j*sizeof(csdatanode *)); /* data nodes */
dptr = (csdatanode **)malloc(j*sizeof(csdatanode *)); /* tail ptr   */
str[0] = NULL;
```

```
while(!EQUALS(str,"//"))  /* loop till user enters "//" - end of set */
     {
     fptr = (*ptr1).uval.cschild; /* point to the head of field list */
     /* loop through the field nodes gathering data till user says quit */
     for(k=0; ((k<j) && (fptr)); k++)
        {
         fprintf(stderr, "\n enter %s ==>", fptr->fdname);
         scanf("%[^\n]%*c", str);
         str[30]='\0';
         while(EQUALS(bl_sup(str),"//") && EQUALS(SCAT,"M") && FIRST)
              {
               fprintf(stderr, "\n The set is MANDATORY.");
               fprintf(stderr, "\n Reenter %s ==>", fptr->fdname);
               scanf("%[^\n]%*c", str);
               str[30]='\0';
              }
         FIRST = FALSE;
         if(EQUALS(bl_sup(str),"\0"))
               strcpy(str,"-");
         if(k==0 && EQUALS(str,"//"))
               break;
         while(k != 0 && EQUALS(bl_sup(str),"//"))
              {
               fprintf(stderr, "\n Reenter %s ==>", fptr->fdname);
               scanf("%[^\n]%*c", str);
               str[30]='\0';
              }
         node[k] = (csdatanode *)malloc(sizeof(csdatanode)); /*data space*/
         if(fptr->fchild == NULL) /* first field node in list */
                   {
                   fptr->fchild = node[k];
                   node[k]->prev = NULL;
                   node[k]->next = NULL;
                   node[k]->parent = fptr;
                   strcpy(node[k]->csdata,str);
                   dptr[k] = node[k];
                   }
         else      /* all following field nodes */
                   {
                   node[k]->prev = dptr[k];
                   node[k]->next = NULL;
                   node[k]->parent = dptr[k]->parent;
                   dptr[k]->next = node[k];
                   strcpy(node[k]->csdata,str);
                   dptr[k] = node[k];
                   }
         fptr=fptr->next;
         } /* end of for */
     } /* end of while */
/*  Test if the columnar set has some data in it  */
fptr = ptr1->uval.cschild;
while(fptr)
    {
     if(fptr->fchild == NULL)
        fptr = fptr->next;
     else
        {
        LIVE = TRUE;
        break;
        }
    }
```

```
if(!LIVE)
   dellnode(ptrl);
}
```

```
/* ========================================================================= */
/*        Add a node for the free text set                                   */
/* ========================================================================= */

add_ts_node(freetxt)
char **freetxt;

      {
      tsnode *ptrts;
      ptrts=(tsnode *)malloc(sizeof(tsnode));

          (*ptrl).uval.tschild = ptrts;
          ptrts->parent = ptrl;
          ptrts->tstext = freetxt;
      }
```

```
$include "../../def/file.h"
#define MAIN
#include "sysdef.h"
#include "ams.h"

/* external definitions for global variables */

int    col = 0;       /*  number of columns for indentation   */
int    pos;
int    DEAD;          /*  Indicator for no text in the free text set */
int    getout;
char   ALIAS[11];        /*  Short Message Title */
char   MNO[5];            /*  message number     */
char   TITLE[31];         /*  message title      */
char   FDESC[9];
char   FNAME[41];        /*  data field identifier       */
char   FDESC[9];
char   SCOL_J[4];        /*  start column                */
char   CHDR[25];

list0    *head;
list0    *ptr0;
list1    *ptr1;

lsnode      *taills, *ptrpos;
csfieldnode *tailcsf;
csdatanode  *tailcsd;
tsnode      *tailts;

 /* ********************************************************************** */
 /*                                                                        */
 /*        ams.c                                                           */
 /*                                                                        */
 /*        MESSAGE ENTRY SYSTEM                                            */
 /*                                                                        */
 /*        This is the main routine:                                       */
 /*                                                                        */
 /* ********************************************************************** */

main ()

{
      char   label1[16],label2[7];
      char   option[5];         /*  EXER or OPER */
      char   ALEXER[11];       /*  message alias for EXER   */
      char   ALOPER[11];       /*  message alias for OPER   */
      char   ALELSE[11];       /*  message alias for OTHERS */

      addOnode(label1,label2);

      fprintf(stderr,"\n Enter EXER or OPER => ");
      scanf("%[^\n]%*c",option);
      if(EQUALS(bl_sup(option),"EXER") ||
         EQUALS(bl_sup(option),"exer"))
         {
         strcpy(ALEXER,"ITXT_EXER");
         SET_Handler(ALEXER); /* Build up Introductory Text Sets for EXER */
         }
      else if(EQUALS(bl_sup(option),"OPER") ||
              EQUALS(bl_sup(option),"oper"))
```

```
        {
        strcpy(ALOPER,"ITXT_OPER");
        SET_Handler(ALOPER); /* Build up Introductory Text Sets for OPER */
        }
    else
        {
        strcpy(ALELSE,"ITXT_ELSE");
        SET_Handler(ALELSE); /* Build up Introductory Text Sets for OTHERS */
        }

    SET_Handler(ALIAS);          /* Build up Main Text Sets of Message */

#ifdef DEBUG
    display();
#endif

    convert();

}   /*   end of main   */
```

```
    /* ================================================================== */
    /*            SET Handler                                             */
    /* ================================================================== */

    SET_Handler(ALIAS)          /* Build up Main Text Sets of Message */

    char  *ALIAS;

    {
    char   CAT[4];              /*  cat */
    char   SETID[9];            /*  set id */

    struct {
            char  SALIAS[10];
            int   SNO;
           } SETKEY;

    /*   retrieve each setid in the order of set no */
    /*   add set node */

    strncpy (SETKEY.SALIAS, bl_pad(ALIAS,10), 10);
    for (SETKEY.SNO=1; acckey (sets,&SETKEY) == 0; SETKEY.SNO++)
         {
         gfield (scat, CAT);
         CAT[3] = '\0';
         gfield (setid, SETID);
         SETID[8] = '\0';
         /*   Manage the set according to its type   */
         if((SETID[0] >= '0') && (SETID[0] <= '9')) /* columnar set */
               CSET_mgr(SETID, bl_sup(CAT));
         else if(EQUALS(bl_sup(SETID),"AMPN") || EQUALS(SETID,"NARR") ||
                   EQUALS(SETID,"RMKS")) /*   free text set  */
```

```
            FSET_mgr(SETID, bl_sup(CAT));
        else                        /*   linear set     */
            LSET_mgr(SETID, bl_sup(CAT));
        }
    }
```

```
/* ================================================================ */
/*              Columnar Set Manager                                */
/* ================================================================ */

CSET_mgr(SETID, CAT)

char *SETID;
char *CAT;

{
 fprintf(stderr,"\nSET ID: %s",SETID);
 addlnode('C',SETID);
 FIELD_handler('C',SETID,CAT);
}
```

```
/* ================================================================ */
/*              Free Text Set Manager                               */
/* ================================================================ */

FSET_mgr(SETID, CAT)

char *SETID;
char *CAT;

{
    fprintf(stderr,"\nSET ID: %s",SETID);
    addlnode('F',SETID);
    freef_handler();
}
```

```
/* ================================================================ */
/*                Linear Set Manager                                */
/* ================================================================ */

LSET_mgr(SETID, CAT)

char *SETID;
char *CAT;

{
 if(EQUALS(CAT,"O,R"))
    for(;;)
        {
         fprintf(stderr,"\nSET ID: %s",SETID);
         addlnode('L',SETID);
         FIELD_handler('L',SETID,CAT);
         if(getout)
           {
            getout = FALSE;
            break;
           }
        }
 else
    {
     fprintf(stderr,"\nSET ID: %s",SETID);
     addlnode('L',SETID);
     FIELD_handler('L',SETID,CAT);
    }
}
```

```
/* ================================================================ */
/*                FIELD Handler                                     */
/* ================================================================ */

FIELD_handler(STYP,SETID,SCAT)

char   STYP;
char   *SETID, *SCAT;

{
char   DFI[6];        /*  data field identifier       */
char   DUI[4];        /*  data usage identifier       */
char   FCAT[4];        /*  cat */

struct {
        char  SID[8];
        int   FNO;
       } FLDKEY;

/*   retrieve each field id in the order of field no */ /*   add field node */
```

```
    pos = 1;
    strncpy (FLDKEY.SID, bl_pad(SETID, 8), 8);
    for (FLDKEY.FNO=1; acckey (field,&FLDKEY) == 0; FLDKEY.FNO++)
        {
        gfield (fcat, FCAT);
        FCAT[3] = '\0';
        gfield (fname, FNAME);
        FNAME[40] = '\0';
        gfield (fdfi, DFI);
        DFI[5] = '\0';
        gfield (fdui, DUI);
        DUI[3] = '\0';
        switch(STYP)
           {
           case 'C':                /*   columnar set   */
               gfield (fcol, SCOL_J);
               SCOL_J[3] = '\0';
               DFI_handler('C',DFI,DUI,SETID,SCAT,FLDKEY.FNO,FCAT);
               break;
           case 'L':                /*   linear set     */
               DFI_handler('L',DFI,DUI,SETID,SCAT,FLDKEY.FNO,FCAT);
               break;
           }
        if(getout)
            break;
        }
    switch(STYP)
       {
        case 'C': add_csd_node(SCAT);
                  break;
        case 'L':
                  if(pos < FLDKEY.FNO)
                     dellslist(ptrpos);
                  else if((getout == TRUE) && (pos == FLDKEY.FNO))
                     dellslist(ptrpos);
                  break;
       }
    }
```

```
/* ================================================================ */
/*                  DFI Handler                                     */
/* ================================================================ */

DFI_handler(KIND,FDFI,FDUI,SETID,SCAT,FDNO,FCAT)

char    KIND;
char    *FDFI, *FDUI;
char    *SETID, *SCAT;
char    *FCAT;
int     FDNO;

{
```

```
int     THRU;
char    LRJ[2];
char    NOTYP[11];
char    DVAL[21];
char    COMBI[30];
struct
     {
     char DFI[6];
     char DUI[4];
     } DFIKEY;
/*  retrieve NO-TYPE from dfis relation and add it to the field  */
strcpy (DFIKEY.DFI, FDFI);
strcpy (DFIKEY.DUI, FDUI);
if (acckey (dfis,&DFIKEY))
    {
       fprintf(stderr,"\n *** dfi not found !!!");
    }
gfield (dlrj, LRJ);
LRJ[1] = '\0';
gfield (dformat, NOTYP);
NOTYP[10] = '\0';
switch(KIND)
     {
      case 'C': gfield(dcolhdr,CHDR);
                CHDR[23] = '\0';
                add_csf_node(bl_sup(FNAME),bl_sup(CHDR),
                             bl_sup(SCOL_J),bl_sup(LRJ),
                             bl_sup(NOTYP));
                break;
      case 'L': gfield(dfdesc,FDESC);
                FDESC[7] = '\0';
                add_ls_node(bl_sup(FNAME),bl_sup(FDESC),
                            bl_sup(LRJ),bl_sup(NOTYP));
                fprintf(stderr,"\n enter %s ==>",FNAME);
                scanf("%[^\n]%*c",DVAL);
                DVAL[20]='\0';
                THRU = FALSE;
                while(!THRU)
                     {
                if(EQUALS(SETID,"MSGID   ") && FDNO==1)
                    {
                    strcpy(ALIAS,upstr(DVAL));
                    while (acckey (msg,bl_pad(ALIAS,10)))
                         {
                         fprintf(stderr,"\n WARNING: ILLEGAL message type!");
                         fprintf(stderr,"\n Reenter %s ==>",FNAME);
                         scanf("%[^\n]%*c",DVAL);
                         DVAL[20]='\0';
                         strcpy(ALIAS,upstr(DVAL));
                         }
                    gfield (mno, MNO);
                    MNO[4] = '\0';
                    gfield (mtitle, TITLE);
                    TITLE[30] = '\0';
                    insOnode(MNO,ALIAS);
                    }
                if(EQUALS(bl_sup(FCAT),"M"))
                   while(EQUALS(bl_sup(DVAL),"\0"))
                         {
                         fprintf(stderr,"\n This field is MANDATORY.");
                         fprintf(stderr,"\n Enter data ==>");
```

```
                        scanf("%[^\n]%*c",DVAL);
                        DVAL[20]='\0';
                        }
                    while (!VALID(bl_sup(DVAL),NOTYP))
                        {
                        fprintf(stderr,"\n The input is not valid");
                        fprintf(stderr,"\n The valid data type is",NOTYP);
                        fprintf(stderr,"\n Reenter %s ==>",FNAME);
                        scanf("%[^\n]%*c",DVAL);
                        DVAL[20]='\0';
                        }
                    if(EQUALS(DVAL,"\0"))
                         strcpy(DVAL,"-");
                    if(!EQUALS(FDESC,"\0"))
                        {
                         strcpy(COMBI,FDESC);
                         strcat(COMBI,":");
                         strcat(COMBI,DVAL);
                         ins_ls_data(COMBI);
                        }
                    else if(!EQUALS(DVAL,"//"))
                         ins_ls_data(DVAL);
                    if(!EQUALS(DVAL,"-") && !EQUALS(DVAL,"")
                       && !EQUALS(DVAL,"//"))
                       pos = FDNO+1;
                    else if(pos==FDNO)
                       ptrpos = taills;
                    THRU = TRUE;
                    if(EQUALS(bl_sup(DVAL),"//"))
                       if(!EQUALS(SCAT,"M"))
                          getout = TRUE;
                       else
                          {
                            fprintf(stderr,"\n This set is MANDATORY.");
                            fprintf(stderr,"\n Enter data ==>");
                            scanf("%[^\n]%*c",DVAL);
                            DVAL[20]='\0';
                            THRU = FALSE;
                          }
                         } /* end of while(THRU) */
                    break;
        } /* end of switch */
    } /* end of DFI_handler */
```

```
/* ================================================================*/
/*              Type Handler for  Construct "freeform"             */
/* ================================================================*/

freef_handler()

{
char   **freetext;
```

```
freetext=(char **)malloc((MAXLINES+1) * (sizeof(char *)));
readtxt(freetext);
if(DEAD)
   dellnode(ptr1);
else
   add_ts_node(freetext);
}
```

```
/* ================================================================*/
/*              Validity Checker for data format and value         */
/* ================================================================*/

VALID(DVAL,NOTYP)

char   *DVAL;
char   *NOTYP;

{
/*  check if DVAL is valid                                         */
/*     i.e. f.minw <= length(DVAL) <= f.maxw for f.type=FORMATTYPE  */

/*  check if DVAL is in f.ftype  */
return(1);
}
```

```
#include "ams.h"

/* ========================================================================= */
/*     suppress blanks at the end of string                                  */
/* ========================================================================= */

char *bl_sup(str)
char   *str;
{
int   k;
k=strlen(str);
while((k > 0) && (str[k-1] == ' '))
     k--;
str[k] = '\0';
return(str);
}

/* ========================================================================= */
/*     pad blanks to make the string length n                                */
/* ========================================================================= */

char *bl_pad(str,n)
char   *str;
int   n;
{
int   k;
k=strlen(str);
while(k < n)
     {
      str[k] = ' ';
      k++;
     }
str[k] = '\0';
return(str);
}
```

```
/* ========================================================================= */
/*     Convert the first k substring of "str" to integer                     */
/* ========================================================================= */

int atoin(str,k)
char str[];
int  k;

     {
     int i,n;
     n=0;
     for(i=0; i<k; ++i)
         n = 10 * n + str[i] - '0';
     return(n);
     }
```

```
/* ========================================================================= */
/*     Convert char c to upper case; ASCII only                              */
/* ========================================================================= */

char upchar(c)
char c;
{
 if(c >= 'a' && c <= 'z')
    return(c+'A'-'a');
 else
    return(c);
}
```

```
/* ========================================================================= */
/*     Convert str s to upper case; ASCII only                               */
/* ========================================================================= */

char *upstr(s)
char *s;
{
 int  i;
 for(i=0; s[i]; i++)
     s[i] = upchar(s[i]);
 return(s);
}
```

```
#include "ams.h"

/* functions to delete nodes and lists */

/* ================================================================== */
/*    indirectly  recursive with del2list                             */
/*    delete the node pointed to by ptrl2 and                         */
/*    all its children                                                */
/* ================================================================== */

dellsnode(ptr)
lsnode *ptr;

      {
      if(ptr->fchild != NULL)
         dellslist(ptr->fchild);
      if(ptr->prev == NULL)
         if(ptr->next == NULL)
            {
            ptr->parent->uval.lschild = NULL;
            dellnode(ptr->parent);
            }
         else
            {
            ptr->parent->uval.lschild = ptr->next;
            ptr->next->prev = NULL;
            }
      else
         {
         ptr->prev->next= ptr->next;
         if(ptr->next != NULL)
            ptr->next->prev = ptr->prev;
         }
      free(ptr);
      }
```

```
/* ================================================================== */
/*   delete the list followed by the node pointed to by ptrl2         */
/*   and all the sublists                                             */
/* ================================================================== */

dellslist(ptr2)
lsnode *ptr2;

      {
      if(ptr2->next != NULL)
         dellslist(ptr2->next);
      dellsnode(ptr2);
      }
```

```
/* ========================================================================= */
/*   delete the list followed by the node pointed to by ptr12                */
/*   and all the sublists                                                    */
/* ========================================================================= */

delcsflist(ptr2)
csfieldnode *ptr2;

      {
      if(ptr2->next != NULL)
         delcsflist(ptr2->next);
      delcsfnode(ptr2);
      }
```

```
/* ========================================================================= */
/*                                                                           */
/*    indirectly  recursive with del2list                                    */
/*    delete the node pointed to by ptr12 and                                */
/*    all its children                                                       */
/* ========================================================================= */

delcsfnode(ptr)
csfieldnode *ptr;

      {
      if(ptr->fchild != NULL)
         delcsdlist(ptr->fchild);
      if(ptr->prev == NULL)
         if(ptr->next == NULL)
            ptr->parent = NULL;
         else {
            ptr->parent->uval.cschild = ptr->next;
            ptr->next->prev = NULL;
         }
      else {
         ptr->prev->next= ptr->next;
         if(ptr->next != NULL)
            ptr->next->prev = ptr->prev;
      }
      free(ptr);
      }
```

```
/* ========================================================================= */
/*   delete the list followed by the node pointed to by ptr12                */
/*   and all the sublists                                                    */
/* ========================================================================= */

delcsdlist(ptr)
csdatanode *ptr;

      {
      if(ptr->next != NULL)
         delcsflist(ptr->next);
      delcsdnode(ptr);
      }
```

```
/* ========================================================================= */
/*                                                                           */
/*    indirectly  recursive with del2list                                    */
/*    delete the node pointed to by ptr12 and                                */
/*    all its children                                                       */
/* ========================================================================= */

delcsdnode(ptr)
csdatanode *ptr;

      {
      if(ptr->prev == NULL)
         if(ptr->next == NULL)
            ptr->parent = NULL;
         else {
            ptr->parent->fchild = ptr->next;
            ptr->next->prev = NULL;
         }
      else {
         ptr->prev->next= ptr->next;
         if(ptr->next != NULL)
            ptr->next->prev = ptr->prev;
      }
      free(ptr);
      }
```

```
/* ========================================================================= */
/*                                                                           */
/*    indirectly  recursive with del2list                                    */
/*    delete the node pointed to by ptr12 and                                */
/*    all its children                                                       */
/* ========================================================================= */

deltsnode(ptr)
tsnode *ptr;

      {
      ptr->parent->uval.tschild = NULL;
      free(ptr);
      }









/* ========================================================================= */
/*     Delete all the SET IDENTs of a message from the level 1 list          */
/* ========================================================================= */

delllist(ptr1)
list1 *ptr1;

      {

      if(ptr1->next != NULL)
         delllist(ptr1->next);

      dellnode(ptr1);

      }









/* ========================================================================= */
/*      Delete all the messages from the linked lists                       */
/* ========================================================================= */

del0list(ptr10)
list0 *ptr10;

      {
      if(ptr10->next != NULL)
         del0list(ptr10->next);
```

```
      del0node(ptr10);
      }
```

```
/* ========================================================================= */
/*      Delete the whole message from the linked lists                       */
/* ========================================================================= */

del0node(ptr10)
list0 *ptr10;

      {
      list0 *tptr0;
      tptr0 = ptr10;
      if(ptr10->schild != NULL)
         delllist(ptr10->schild);
      if(ptr10->prev == NULL)
         {
         if(ptr10->next == NULL)
            head = NULL;
         else
             {
             head = ptr10->next;
             ptr10->next->prev = NULL;
             ptr10 = ptr10->next;
             }
         }
      else
         {
         ptr10->prev->next= ptr10->next;
         ptr10->next->prev = ptr10->prev;
         ptr10 = ptr10->prev;
         }

      free(tptr0);

      }
```

```
/* ========================================================================= */
/*      Delete a SET IDENT in the corresponding level 1 list                 */
/* ========================================================================= */

del1node()
```

```
{
 listl   *tptrl;
 tptrl = ptrl;
 switch(ptrl->settyp)
        {
         case 'C':    /*   columnar set   */
                  if((*ptrl).uval.cschild != NULL)
                      delcsflist((*ptrl).uval.cschild);
                  break;
         case 'F':    /*   free text set   */
                  if((*ptrl).uval.tschild != NULL)
                      deltsnode((*ptrl).uval.tschild);
                  break;
         case 'L':    /*   linear set   */
                  if((*ptrl).uval.lschild != NULL)
                      dellslist((*ptrl).uval.lschild);
                  break;
        }
if(ptrl->prev == NULL)
   {
   if(ptrl->next == NULL)
      {
      ptr0->schild = NULL;
      }
   else
      {
      ptrl->parent->schild = ptrl->next;
      ptrl->next->prev = NULL;
      ptrl = ptrl->next;
      }
   }
else
   {
   ptrl->prev->next= ptrl->next;
   if(ptrl->next != NULL)
      ptrl->next->prev = ptrl->prev;
   ptrl = ptrl->prev;
   }

free(tptrl);

}
```

```
#include "ams.h"

/* ================================================================== */
/*     1. Get the message data from the linked lists                  */
/*     2. Display it on screen                                        */
/* ================================================================== */

int display()

      {
      int    i;
      listl  *ptr;

      if(ptr0)
         {
         printf("\n\n\n%s %s\n",ptr0->mtag1,ptr0->mno);
         printf("%s %s\n",ptr0->mtag2,ptr0->mid);

         col += 5;
         ptr=ptr0->schild;

         while(ptr)
              {
              for(i=0; i<col; i++) putchar(' ');
              printf("SET IDENT:%s\n",ptr->setid);
              switch(ptr->settyp)
                   {
                   case 'C':     /*   columnar set   */
                             dsp_col_set(ptr->uval.cschild);
                             dsp_csdata(ptr->uval.cschild);
                             break;
                   case 'F':     /*   free text set   */
                             dsp_free_set(ptr->uval.tschild);
                             printf("\n");
                             break;
                   case 'L':     /*   linear set   */
                             dsp_lin_set(ptr->uval.lschild);
                             break;
                   }
              ptr=ptr->next;
              }
         col -= 5;
         }

      }

      /* ================================================================== */
      /*     1. Get the field-level data from the linked lists              */
      /*     2. Display them on screen                                      */
      /* ================================================================== */
```

```
dsp_lin_set(ptr)

lsnode  *ptr;

{

int     i;

   col += 5;

   while(ptr)
        {
        for(i=0; i<col; i++) putchar(' ');
        printf("%s: %s\n", ptr->fdname, ptr->fval);
        dsp_lin_set(ptr->fchild);
        ptr=ptr->next;
        }

   col -= 5;

}
```

```
/* ================================================================== */
/*     1. Get the field-level data from the linked lists              */
/*     2. Display them on screen                                      */
/* ================================================================== */

dsp_col_set(head)

csfieldnode  *head;
{
int     i=1, n;
csfieldnode  *ptr;
ptr=head;
while(ptr)
    {
    n = atoin(ptr->colpos,strlen(ptr->colpos)-1);
    for(; i<n ; i++)  putchar(' ');
    printf("%s", ptr->colhdr);
    i=i+strlen(ptr->colhdr)-1;
    ptr=ptr->next;
    }
}
```

```
/* ========================================================================= */
/*     1. Get the field-level data from the linked lists                     */
/*     2. Display them on screen                                             */
/* ========================================================================= */

dsp_csdata(head)

csfieldnode  *head;
{
int    i, j, k, n;
int    LR, m;
csfieldnode  *fptr;
csdatanode   **dptr;

printf("\n");
fptr = head; /* point to the head of column list */
for(j=0; fptr; j++)  /* count the columns */
       fptr = fptr->next;

/* allocate storage for pointers to column lists */
dptr = (csdatanode **)malloc(j*sizeof(csdatanode *));

/* set pointers to column lists */
for(k=0, fptr=head; ((fptr) && (k<j)); k++)
   {
    dptr[k] = fptr->fchild;
    fptr = fptr->next;
   }

while(dptr[0]) /* for each line of data in column lists */
   {
    i = 1;
    fptr = head; /* point to start of list of columns */

    /* print one line of data, tabbing to column fields */
    for(k=0; fptr; k++) /* indent appropriately and print data */
        {
         n = atoin(fptr->colpos,strlen(fptr->colpos)-1);
         LR = strlen(fptr->colpos);
         switch(fptr->colpos[LR-1]) /* tab to next field */
              {
              case 'L':
                       for(; i<n ; i++)  putchar(' ');
                       break;
              case 'R':
                       m=n+strlen(fptr->colhdr)-strlen(dptr[k]->csdata);
                       for(; i<m ; i++)  putchar(' ');
                       break;
              }
         printf("%s", dptr[k]->csdata);
         i=i+strlen(dptr[k]->csdata)-1;
         dptr[k] = dptr[k]->next;
         fptr=fptr->next;
        }
    printf("\n");
   }
free(dptr);
}
```

```
/* ========================================================================= */
/*     1. Get the field-level data from the linked lists                     */
/*     2. Display them on screen                                             */
/* ========================================================================= */

dsp_free_set(ptr)

tsnode  *ptr;
{
int    k=0;
while(ptr->tstext[k])
   {
    printf("%s", ptr->tstext[k]);
    k++;
    if(ptr->tstext[k])
       printf("\n");
   }
}
```

```
#include "ams.h"

/* ========================================================================= */
/*     1. Get the message data from the linked lists                         */
/*     2. Convert them into standard output format                           */
/* ========================================================================= */

int convert()

      {
      listl  *ptrl;

      if(!ptr0)
         {
          fprintf(stderr, "\n *** ERROR: No message found !!! ");
          return;
         }

      ptrl=ptr0->schild;
      while(ptrl)
           {
            printf("%s",ptrl->setid);
            switch(ptrl->settyp)
                 {
                  case 'C':    /*   columnar set   */
                           convcsflist((*ptrl).uval.cschild);
                           convcsdlist((*ptrl).uval.cschild);
                           break;
                  case 'F':    /*   free text set   */
                           convtsnode((*ptrl).uval.tschild);
                           break;
                  case 'L':    /*   linear set   */
                           convlslist((*ptrl).uval.lschild);
                           break;
                 }
            printf("//\n");
            ptrl=ptrl->next;
           }
      }




      /* ========================================================================= */
      /*     1. Get the field-level data from the linked lists                     */
      /*     2. Convert them into standard output format                           */
      /* ========================================================================= */

      convlslist(ptr)

      lsnode  *ptr;
      {
         while(ptr)
              {
```

```
        printf("/%s", ptr->fval);
        dsp_lin_set(ptr->fchild);
        ptr=ptr->next;
        }
}
```

```
/* ================================================================== */
/*     1. Get the field-level data from the linked lists              */
/*     2. Convert them into standard output format                    */
/* ================================================================== */

convcsflist(head)

csfieldnode  *head;
{
int     i=2, n;
csfieldnode  *ptr;

printf("\n/");
ptr=head;
while(ptr)
   {
   n = atoin(ptr->colpos,strlen(ptr->colpos)-1);
   for(; i<n ; i++)  putchar(' ');
   printf("%s", ptr->colhdr);
   i=i+strlen(ptr->colhdr)-1;
   ptr=ptr->next;
   }
}
```

```
/* ================================================================== */
/*     1. Get the field-level data from the linked lists              */
/*     2. Convert them into standard output format                    */
/* ================================================================== */

convcsdlist(head)

csfieldnode  *head;
{
int     i, j, k, n;
int     LR, m;
csfieldnode  *fptr;
```

```
    csdatanode   **dptr;

    fptr = head; /* point to the head of field list */
    for(j=0; fptr; j++)  /* count the fields */
          fptr = fptr->next;
    /* allocate storage for pointers to column value lists */
    dptr = (csdatanode **)malloc(j*sizeof(csdatanode *));

    /* set pointers to column lists */
    for(k=0, fptr=head; ((fptr) && (k<j)); k++)
       {
        dptr[k] = fptr->fchild;
        fptr = fptr->next;
       }

    while(dptr[0]) /* while there are still data values to print */
        {
        printf("\n/");  /* begin new line with slash */
        i = 2;
        fptr = head;
        /* print one line of data, tabbing to column fields */
        for(k=0; fptr; k++) /* indent appropriately and print data */
            {
             n = atoin(fptr->colpos,strlen(fptr->colpos)-1);
             LR = strlen(fptr->colpos);
             switch(fptr->colpos[LR-1]) /* tab to next field */
                  {
                  case 'L':
                            for(; i<n ; i++)  putchar(' ');
                            break;
                  case 'R':
                            m=n+strlen(fptr->colhdr)-strlen(dptr[k]->csdata);
                            for(; i<m ; i++)  putchar(' ');
                            break;
                  }
             printf("%s", dptr[k]->csdata);
             i = i+strlen(dptr[k]->csdata)-1;
             dptr[k] = dptr[k]->next;
             fptr = fptr->next;
            }
        }
    free(dptr);
    }
```

```
    /* ================================================================== */
    /*     1. Get the field-level data from the linked lists              */
    /*     2. Convert them into standard output format                    */
    /* ================================================================== */

    convtsnode(ptr)
```

```
	tsnode  *ptr;
	{
		printf("/");
		dsp_free_set(ptr);
	}
```

```
#include "ams.h"

/* ========================================================================== */
/*     Read in free text                                                      */
/* ========================================================================== */

readtxt(longtext)
char   **longtext;

{
   register int   i=0;
   register int   len=0;
   register int   DONE=FALSE;
   char  tempbuf[81];

   DEAD = FALSE;
   fprintf(stderr,"\n *** Enter free text and terminate with // ***\n");
   while (!DONE)
     {
      if (scanf ("%[^\n]%*c",tempbuf) == 0) continue;
      len=strlen(tempbuf);
      if (EQUALS(tempbuf+len-2, "//"))
         {
          DONE = TRUE;
          tempbuf[len-2] = '\0';
          if(strlen(bl_sup(tempbuf)) == 0)
             DEAD = TRUE;
         }
      longtext[i] = malloc(len+1);
      strcpy(longtext[i],tempbuf);
      i++;
     }
   longtext[i] = NULL;

}
```

CONSOLE

This section contains the functions used only by the screen oriented TACCNET monitoring program (CONSOLE).

```
console: console.o startup.o initwindow.o getmenu.o perform.o update.o \
   closewindows.o printmenu.o tailfile.o dispqueue.o addcolumn.o \
   getdir.o lastlines.o printtitle.o printborder.o close.o open.o \
   cmdprompt.o getconf.o saveconf.o readfile.o redraw.o clear.o \
   changespeed.o newsys.o view.o select.o dispfile.o admin.o runshell.o \
   usershell.o status.o changed.o writeconf.o lockfile.o readsite.o \
   writelog.o queuesize.o ports.o myname.o fileopen.o
        cc -O -o console console.o startup.o initwindow.o \
   getmenu.o perform.o update.o closewindows.o printmenu.o tailfile.o \
   dispqueue.o addcolumn.o getdir.o lastlines.o printtitle.o printborder.o \
   close.o open.o cmdprompt.o getconf.o saveconf.o readfile.o redraw.o \
   clear.o changespeed.o newsys.o view.o select.o dispfile.o admin.o \
   runshell.o usershell.o status.o changed.o writeconf.o lockfile.o \
   writelog.o readsite.o queuesize.o ports.o myname.o fileopen.o \
   -lcurses -ltermcap

console.o: console.c
        cc -O -c console.c

startup.o: startup.c
        cc -O -c startup.c

initwindow.o: initwindow.c
        cc -O -c initwindow.c

getmenu.o: getmenu.c
        cc -O -c getmenu.c

perform.o: perform.c
        cc -O -c perform.c

update.o: update.c
        cc -O -c update.c

closewindows.o: closewindows.c
        cc -O -c closewindows.c

printmenu.o: printmenu.c
        cc -O -c printmenu.c

tailfile.o: tailfile.c
        cc -O -c tailfile.c

dispqueue.o: dispqueue.c
        cc -O -c dispqueue.c

addcolumn.o: addcolumn.c
        cc -O -c addcolumn.c

getdir.o: getdir.c
        cc -O -c getdir.c

lastlines.o: lastlines.c
        cc -O -c lastlines.c

printtitle.o: printtitle.c
        cc -O -c printtitle.c

printborder.o: printborder.c
        cc -O -c printborder.c
```

```
close.o: close.c
        cc -0 -c close.c

open.o: open.c
        cc -0 -c open.c

cmdprompt.o: cmdprompt.c
        cc -0 -c cmdprompt.c

getconf.o: getconf.c
        cc -0 -c getconf.c

saveconf.o: saveconf.c
        cc -0 -c saveconf.c

readfile.o: readfile.c
        cc -0 -c readfile.c

redraw.o: redraw.c
        cc -0 -c redraw.c

clear.o: clear.c
        cc -0 -c clear.c

changespeed.o: changespeed.c
        cc -0 -c changespeed.c

newsys.o: newsys.c
        cc -0 -c newsys.c
```

```
#include <stdio.h>
#include <fcntl.h>
#include <curses.h>
#include <signal.h>
#include <sys/types.h>

/*
   System-wide data structures.  If this is the main program,
   define the actual data; otherwise, define external references.
*/

#define MAXWINDOWS 4            /* Maximum number of windows allowed        */
#define MAXOPTIONS 11           /* Maximum selections on command menu       */
#define MINCHARS   80L*3L       /* Minimum characters for 'tail' files      */
#define BOTTOM     22           /* Bottom line of terminal before menu      */
#define BKSP       0x08         /* Backspace key                            */
#define CTLL       0x0c         /* Control-L (redraw key)                   */
#define BASECOL    4            /* Base column for columnar window info     */
#define TABSIZE    16           /* Width of a column in a columnar window   */

#define LOGWINDOW 1             /* Constant defining window type of LOG     */
#define QWINDOW   2             /* Constant defining window type of QUEUE   */

#define SWINDOWS   0            /* Standard console windows screen          */
#define SSTATUS    1            /* System status screen                     */

#define CLEAR     0             /* Clear one of the windows                 */
#define CLOSE     1             /* Close one of four static windows if open */
#define GETCONF   2             /* Retrieve a saved configuration           */
#define NEWSYSMON 3             /* Monitor new system                       */
#define OPEN      4             /* Open one of four static windows          */
#define REDRAW    5             /* Repaint the screen in case of a problem  */
#define SAVECONF  6             /* Save configuration                       */
#define SPEED     7             /* Set polling interval (faster or slower)  */
#define VIEW      8             /* View a message file using 'vi'           */
#define ADMIN     10            /* Administer file in .../tables            */
#define COMMAND   11            /* Run a shell command from .../bin         */
#define PORTS     12            /* Display current ports table              */
#define SHELL     13            /* Let user run a shell (/bin/csh)          */
#define SITES     14            /* Display current sites table              */
#define STATUS    15            /* Display system status (up, down)         */

#define MENUFILE ".menufile"  /* File name for menu file in current dir   */
#define DEFDIR "/usr/taccnet" /* Default TACCNET directory                 */

/* Macro definitions for use with window structure */

#define WindowOut(win,str) waddstr(Window[win].Win,str)
#define IsLogWindow(win)  (Window[win].Type==LOGWINDOW)
#define IsQWindow(win)  (Window[win].Type==QUEUE)
#define GoodWindow(win)  (Window[win].Good)

struct window {
               int Good;        /* Indicator that window is active          */
               int TopLine;     /* Line on which window starts              */
               int LeftSide;    /* Column in which window starts            */
               int Width;       /* Width of window in characters            */
               int Size;        /* Number of lines in window                */
               int Type;        /* Window type (defined above)              */
               FILE *Fp;        /* File pointer associated with window      */
```

```
    char Name[60]; /* Path of file (if associated with window) */
    long LastPos;  /* Last file position during 'tail'         */
    long LastTime; /* Last mod time for queue directory        */
    WINDOW *Win;   /* Curses window associated herewith        */
};

/* Macro definitions for correct inverse video processing */

#define StandOut()      if (GoodTerminal) standout()
#define StandEnd()      if (GoodTerminal) standend()
#define wStandOut(win)  if (GoodTerminal) wstandout(win)
#define wStandEnd(win)  if (GoodTerminal) wstandend(win)

/* Type definitions */

typedef char sitename [14];
typedef char filename [40];
typedef char pathname [80];

#ifdef MAINPROGRAM

/* Data declaration - data will be accessible at all levels          */

int NumWindows = 0;             /* Number of windows currently open  */
int MenuSize[2];                /* Number of entries in menu         */
int CurSel = 0;                 /* Current menu selection            */
int CurTier = 1;                /* Current menu tier number (level)  */
int GoodTerminal = 1;           /* Value for good inverse video support */
int Speed = 1;                  /* Number of seconds between refreshes */
char *SysName;                  /* Pointer to argv[0] for program name */
char Menu[2][MAXOPTIONS][20];   /* Storage for menu items            */
struct window Window [MAXWINDOWS]; /* Information about each open window */
char CurSys[12];                /* Name of current system being monitored */
char *ThisName;                 /* The name of this site             */
WINDOW *Command;                /* Command menu window for curses    */

#else

/* External definitions - see above comments for corresponding structures */

extern int NumWindows;
extern int MenuSize[2];
extern int CurSel;
extern int CurTier;
extern int GoodTerminal;
extern int Speed;
extern char *SysName;
extern char Menu[2][MAXOPTIONS][20];
extern struct window Window [MAXWINDOWS];
extern char CurSys[12];
extern char *ThisName;
extern WINDOW *Command;
```

```
#endif
```

```
/*
 * Structure of the information in the first word returned by
 * wait.  If w_stopval==WSTOPPED, then the second structure
 * describes the information returned, else the first.
 */
union wait      {
        int     w_status;               /* used in syscall */
        /*
         * Terminated process status.
         */
        struct {
                unsigned short  w_Termsig:7;    /* termination signal */
                unsigned short  w_Coredump:1;   /* core dump indicator */
                unsigned short  w_Retcode:8;    /* exit code if w_termsig==0 */
        } w_T;
        /*
         * Stopped process status.
         */
        struct {
                unsigned short  w_Stopval:8;    /* == W_STOPPED if stopped */
                unsigned short  w_Stopsig:8;    /* signal that stopped us */
        } w_S;
};

#define WSTOPPED        0177    /* value of s.stopval if process is stopped */
```

```
#include "console.h"

int CurPos = BASECOL;

/*
   AddColumn - Given a string to present in a window, add it in columnar
               form into the next valid position.  Return ERR if there
               is but one position left in the window, so that the '--more--'
               message can be displayed therein.
*/

int AddColumn (str, num, line)

char *str;   /* String to be displayed in the window */
int  num;    /* Window number of window into which columnar data is displayed */
int  *line;  /* Current line number within window */

{
   if ((*line == 0) && (CurPos == BASECOL))
      wclear (Window[num].Win);

   if (*line == Window[num].Size - 1)
      {
       if (CurPos > BASECOL)
          return (ERR);

       CurPos = BASECOL + TABSIZE;

       *line = 0;
      }

   wmove (Window[num].Win, (*line)++, CurPos);

   wprintw (Window[num].Win, "%s", str);
}
```

```
#include "console.h"

static char DefaultEditor[] = "/bin/vi";

/*
   Admin - Provide interface by which knowledgeable user can edit and
           repair table files, including system parameters.
*/

void Admin ()

{
   char Temp[80];  /* Temporary storage for CmdPrompt results */
   char FileName[80];
   char *Editor;   /* Name of user's editor (set in environment) */

   char *getenv ();
   char *malloc ();
   void ExitProcessor ();

   CmdPrompt ("Enter name of table file to edit (i.e., \"ports\"): ",
              Temp, 60);

   if (strlen (Temp))
      {
       if (Temp[0] != '/')  /* Path is not absolute */
          sprintf (FileName, "tables/%s", Temp);    /* Construct file name */
       else
          strcpy (FileName, Temp);                  /* Copy file name as is */

       /* Throw the user into the editor */

       addstr ("x");
       refresh ();

       clear ();
       addstr ("Please stand by...");
       refresh ();

       echo ();
       nocrmode ();

       signal (SIGINT, SIG_IGN);

       if (fork () == 0)  /* This is the child process */
          {
           signal (SIGINT, SIG_DFL);

           if ((Editor = getenv ("EDITOR")) == (char *) 0)
              {
               Editor = malloc (strlen (DefaultEditor) + 1);
               strcpy (Editor, DefaultEditor);
              }

           execl (Editor, Editor, FileName, 0);

           fprintf (stderr, "\n\nError: cannot load editor.\n");
           fprintf (stderr, "Define the environment variable EDITOR");
           fprintf (stderr, " to be the pathname to your editor.\n");
           fprintf (stderr, "\n\n\nPress CR to continue: ");
```

```
            fflush (stderr);

            getchar ();

            exit (0);
           }
         else
            wait ((int *) 0);

         signal (SIGINT, ExitProcessor);

         noecho ();
         crmode ();

         ReDraw ();     /* Put things back the way they were */
        }
}
```

```
#include "console.h"
#include <sys/stat.h>

/*
   Changed - return TRUE if the given window structure's file has been
             modified according to its last modification time, FALSE
             otherwise.
*/

int Changed (num)

int num;  /* Window number of window to be checked */

{
   struct stat buf;  /* Storage for result from stat(2) call */
   int Result;       /* Return value (TRUE/FALSE)            */

   Result = FALSE;

   if (stat (Window[num].Name, &buf) == 0)  /* Check file status */
      {
       Result = (buf.st_mtime != Window[num].LastTime);
       if (Result)
          Window[num].LastTime = buf.st_mtime;  /* Record new mtime */
      }

   return (Result);
}
```

```
#include "console.h"

/*
   Speed - Set polling speed.
*/

void ChangeSpeed ()

{
   char Temp[80];    /* Return storage for menu prompt routine */
   int TempSpeed;
   char OutStr[80];  /* Prompt string */

   sprintf (OutStr, "Polling speed is %d.  Enter new value in seconds (0-10): ",
            Speed);

   CmdPrompt (OutStr, Temp, 80);

   if (strlen (Temp))
      {
       TempSpeed = atoi (Temp);   /* Convert to integer */
       if ((TempSpeed >= 0) && (TempSpeed <= 10))
          Speed = TempSpeed;
      }
}
```

```
#include "console.h"

/*
   Clear - Erase contents of a window to be updated in the next pass.
*/

void Clear ()

{
   char Temp[2];     /* Temporary storage for window number string */
   int  WindowNum;   /* Window number to clear */

   CmdPrompt ("Enter window number to clear (1-4): ", Temp, 2);

   if (strlen (Temp))
      {
       WindowNum = atoi (Temp) - 1;  /* Convert to integer */

       if ((WindowNum > 0) && (WindowNum < MAXWINDOWS))
          {
           wclear (Window[WindowNum].Win);     /* Clear specified window */
           werase (Window[WindowNum].Win);
           wrefresh (Window[WindowNum].Win);
          }
      }
}
```

```
#include "console.h"

/*
   Close - Mark a window as closed, so that its contents will be cleared
           and it will not be updated during a pass until it is reopened.
*/

void Close ()

{
   int WindowNum;
   char Temp[2];

   CmdPrompt ("Window number to close (1-4) : ", Temp, 2);  /* Prompt user */

   if (strlen (Temp))
      WindowNum = atoi (Temp);    /* Convert response to integer */
   else
      WindowNum = ERR;            /* Response was in error */

   if ((WindowNum != ERR) && (WindowNum > 0) && (WindowNum < 5))
      {
       WindowNum--;

       Window[WindowNum].Good = FALSE;       /* Turn this window off */
       wclear (Window[WindowNum].Win);       /* Make it go away */
       wrefresh (Window[WindowNum].Win);
       PrintBorder (WindowNum, FALSE);       /* Erase window border */
      }
}
```

```
#include "console.h"

extern int AutoConfig;  /* Flag indicating auto-save mode for .config# */

/*
   CloseWindows - gracefully remove contents of all defined windows
                  and mark them as no longer active.
*/

void CloseWindows ()

{
   if (AutoConfig == TRUE)  /* Auto-configuration mode on, save configuration */
      WriteConf (".config#");

   move (0,0);
   addstr ("x");
   refresh ();
   erase ();
   clear ();
   refresh ();

   endwin ();
}
```

```
#include "console.h"

/*
   CmdPrompt - Display a prompt on the command line (in the window Command),
               and get user's response into given buffer, maximum of max
               characters.  Echo user input and read until CR or NL.
*/

void CmdPrompt (str, target, max)

char *str;  /* String to be displayed as prompt */
char *target;
int max;

{
   int c;
   int Count = 0;

   wmove (Command, 1, 0);     /* Move to the command line */
   wclrtoeol (Command);       /* Get rid of command line menu */

   *target = '\0';            /* Force empty string in case of error */

   wmove (Command, 1, 0);
   waddstr (Command, str);    /* Display the prompt */
   wrefresh (Command);

Top:
   while (Count < max)
      {
       c = wgetch (Command);        /* Read a character of user's response */

       if ((c == (int) '\n') || (c == (int) '\r'))
          {
           target[Count] = '\0';    /* Terminate string */
           return;
          }
       else if (c == '')          /* User hit ctrl-u */
          {
           target[0] = '\0';
           return;
          }
       else if (c == 0x08)          /* User hit backspace */
          if (Count)
             {
              Count--;              /* Back user up */
              waddch (Command, c);
              wrefresh (Command);
              waddch (Command, ' ');
              wrefresh (Command);
              waddch (Command, c);
              wrefresh (Command);
              continue;             /* Reiterate */
             }
          else
             continue;

       waddch (Command, c);         /* Echo user's response */
       wrefresh (Command);
```

```
        target[Count++] = (char) c; /* Construct result string */
       }

    waddch (Command, (int) ''); /* Let user know he's at margin */
    Count--;
    waddch (Command, 0x08);      /* Back up one */
    wrefresh (Command);
    waddch (Command, ' ');
    wrefresh (Command);
    waddch (Command, 0x08);
    wrefresh (Command);
    goto Top;
}
```

```
#define MAINPROGRAM
#include "console.h"

/*
   Console - TACCNET system console program.  Provide a user interface
             to the TACCNET system, including constant display of system
             logs and message flow, permitting on-demand examination
             and hand processing of messages.  The interface is
             screen oriented, utilizing the 'curses' software package
             for full terminal compatibility.

             Usage: console <taccnet-root> [ -<options> ] ...
*/

int CurScreen = SWINDOWS;  /* Which screen to update: windows or status */

main (argc, argv)

int  argc;
char **argv;

{
   int i;
   int Exit = FALSE;                /* Local exit flag */
   int Function;                    /* Local function request indicator */

   int ExitProcessor ();            /* Interrupt handler */

   signal (SIGINT, ExitProcessor); /* Define interrupt service routine */

   SysName = argv[0];               /* Global system name pointer */

   StartUp (argc, argv);            /* Process start-up options and init data */

   InitWindow ();                   /* Initialize main window and menu */

   while (!Exit)
      {
       /* Update all windows before processing command */

       if (CurScreen == SWINDOWS)
          {
           for (i = 0; i < MAXWINDOWS; i++)
              if (GoodWindow(i))
                  Update (i);       /* Update windows only if defined */
          }
       else
          Status (TRUE);            /* Monitor system status (tier 2, sel 6) */

       Function = GetMenu ();       /* Get menu selection item */

       if (Function > 0)
          Perform (Function - 1);   /* Perform requested function */

       Exit = (Function == 0);      /* Determine exit condition */
      }

   CloseWindows ();                 /* Prepare to exit */

   exit (0);
```

```
}



int ExitProcessor ()

{
   CloseWindows ();
   exit (1);
}
```

```
#include "console.h"

/*
   DispFile - Show the contents of a small data file on the screen,
              prompting the user to press CR to return to normal display.
*/

void DispFile (fname)

char *fname;  /* Name of file to display */

{
   FILE *Fp;  /* File pointer for reading from specified file */
   char Temp[8];
   int c;

   Fp = fopen (fname, "r");  /* Try to open the file */

   if (Fp == (FILE *) NULL)
      {
       CmdPrompt ("Cannot open table file.  Press CR to continue: ", Temp, 7);
       return;
      }

   addstr ("x ");  /* Must take care of rampant sticky inverse video bug */
   refresh ();

   clear ();
   refresh ();

   StandOut ();
   printw ("The file \"%s\" contains: ", fname);
   StandEnd ();

   addstr ("\n\n");

   /* Display file */

   while ((c = getc (Fp)) != EOF)  /* Loop through the file */
      addch (c);

   refresh ();

   CmdPrompt ("Press CR to return to normal display: ", Temp, 7);

   fclose (Fp);  /* Return used file pointer */

   ReDraw ();
}
```

```
#include "console.h"

extern int CurPos;

/*
   DispQueue - Display queue contents in window 'num' by reading the
               names of the files in the queue's Unix directory and
               updating the window to reflect the contents.
*/

void DispQueue (num)

int num;  /* Window number of window in which queue should be displayed */

{
   static filename *MsgFiles;   /* Pointer to list of files in the queue */
   static int CurFilePtr = 0;   /* Index of current filename */
   static pathname QueueName;   /* Name of the current queue */
   int    Line = 0;             /* Current line on queue window */

   filename *GetDir ();

   /* If list exhausted or new queue, read again for more file names */

   CurPos = BASECOL;

   if ((strcmp(Window[num].Name, QueueName)) || (CurFilePtr == 0) ||
       (*MsgFiles [CurFilePtr] == NULL))
      {
       if (!Changed (num))  /* If window Window[num] has not changed.. */
          return;

       MsgFiles = GetDir (Window[num].Name);  /* Load directory structure */
       strcpy (QueueName, Window[num].Name);  /* Reset queue name */
       CurFilePtr = 0;                        /* Reset list index */
      }

   wmove (Window[num].Win, 0, 0);

   if (MsgFiles == NULL)
      {
       wclrtobot (Window[num].Win);
       wrefresh (Window[num].Win);
       return;
      }

   /* Format display now, given that the files are in the data structure */

   while ((MsgFiles[CurFilePtr] != NULL) && (*MsgFiles[CurFilePtr] != NULL))
      {
       if (AddColumn (MsgFiles[CurFilePtr++], num, &Line) == ERR) /* Add data */
          {
           wmove (Window[num].Win, Window[num].Size-1, Window[num].Width-10);
           wprintw (Window[num].Win, "--more--");
           wrefresh (Window[num].Win);
           CurFilePtr = 0;
           return;     /* No need to continue if out of space in window */
          }
      }
```

```
    CurFilePtr = 0;
    CurPos = 0;

    wrefresh (Window[num].Win);  /* Repaint window */
}
```

```
#include "console.h"

/*
   Expand - Make one large window out of two similar windows to occupy
            the same space.
*/

void Expand ()

{
   char Temp [80];  /* Storage for returned ASCII window number */
   int WindowNum;   /* Window that is desired to span two windows */

   CmdPrompt ("Enter window number to expand (1-4): ", Temp, 80);
   if (strlen (Temp))
      {
       WindowNum = atoi (Temp) - 1;  /* Conver to integer */

       if (!GoodWindow(WindowNum))    /* Window must be open to be expanded */
          return;

       switch (WindowNum) /* Establish correspondence between similar windows */
          {
           case 0  : OtherWindowNum = 1;    /* Windows 0 and 1 go together */
                     break;
           case 1  : OtherWindowNum = 0;
                     break;
           case 2  : OtherWindowNum = 3;    /* Windows 2 and 3 go together */
                     break;
           case 3  : OtherWindowNum = 2;
                     break;
           default : return;
          }

       /* Make sure corresponding other-window is closed */

       if (!GoodWindow(OtherWindowNum))
          {
           CmdPrompt ("The new window would overwrite an existing window.  Is this OK? ",
                      Temp, 80);

           if ((Temp[0] != 'y') && (Temp[0] != 'Y'))
              return;

           Window[OtherWindowNum].Good = FALSE;  /* Close other window */

          }

      }
}
```

```
#include "console.h"

/*
   GetConf - restore configuration from saved configuration file.
*/

void GetConf ()

{
   char ConfFileName[80];  /* Configuration file name    */
   int i;

   CmdPrompt ("Enter configuration file name: ", ConfFileName, 80);

   if (strlen (ConfFileName))
      {
       ReadFile (ConfFileName);                 /* Read configuration file    */

       /* Redraw screen based on new configuration */

       for (i = 0; i < MAXWINDOWS; i++)
           {
            wclear (Window[i].Win);             /* Clear text on all windows   */
            wrefresh (Window[i].Win);

            PrintBorder (i, GoodWindow(i));     /* Draw or erase given borders */
            if (GoodWindow(i))
               PrintTitle (i);                  /* Print open windows' title   */
           }
      }
}
```

```
#include "console.h"
#include <sys/dir.h>

#define NFILES 96   /* Max files per directory */
#define Size (sizeof (struct direct))

filename *GetDir (Dir)
char *Dir;

{
   long lseek ();
   struct direct DirEntry;
   register int Num;
   static filename Entries [NFILES];
   int DirFd; /* Directory file descriptor */

   Num = 0;

   if ((DirFd = open(Dir, O_RDONLY)) < 0)
      {
       return (NULL);  /* Return empty list if can't open directory */
      }

   if (lseek (DirFd, 32L, 0) == (long) ERR)   /* Skip . and .. */
      {
       close (DirFd);
       return (NULL);
      }

   while (read (DirFd, &DirEntry, Size) != 0)
       if ((DirEntry.d_ino != (ino_t) 0) && (DirEntry.d_name [0] != '.'))
          {
           strcpy (Entries [Num], DirEntry.d_name);
           Num++;
          }

   close (DirFd);

   if (Num > 0)
      {
       Sort (Entries, Num);
       *Entries [Num] = NULL;   /* Set the last one to NULL */
       return (Entries);
      }
   else
       return (NULL);
}
```

```
Sort (Entries, Num)
filename Entries [];
register int Num;

{
   register int Gap, i, j;
   filename Temp;

   for (Gap = Num / 2; Gap > 0; Gap /= 2)
       for (i = Gap; i < Num; i++)
           for (j = i - Gap; j >= 0; j -= Gap)
               {
                if (strcmp (Entries [j], Entries [j+Gap]) <= 0)
                    break;
                strcpy (Temp, Entries [j]);
                strcpy (Entries [j], Entries [j+Gap]);
                strcpy (Entries [j+Gap], Temp);
               }
}
```

```
#include "console.h"

extern int CurScreen;

/*
   GetMenu - get user's selection of a menu item, if available.  If user
             does not select a menu item within a certain threshold, return
             with a -1, so that the top level can refresh windows as
             needed.  Otherwise, return audited input selection.  EXIT
             request is 0, errors are also -1, and all other requests
             are indices corresponding to order in the menu file.
*/

int GetMenu ()

{
   int c;

   int StopMenu ();          /* Define interrupt handler for 'alarm' */

   alarm (0);

Repeat:
   if (CurScreen == SSTATUS)
      CurTier = 1;           /* Don't permit tier 0 from status screen */

   PrintMenu ();

   signal (SIGALRM, StopMenu);

   alarm (Speed);            /* Set alarm for keyboard timeout */

   c = getchar ();           /* Get character from stdin */

   alarm (0);
   signal (SIGALRM, SIG_DFL);

   if (c > 0)                /* A character was pressed within threshold */
      {
       switch ((char) c)
          {
           case '/'  :       CurSel = 0;
                             CurTier = (CurTier + 1) % 2;
                             goto Repeat;

           case '\\'  :      CurSel = 0;
                             CurTier = (CurTier ? 0 : 1);
                             goto Repeat;

           case ' '  :       CurSel = (CurSel + 1) % MenuSize[CurTier];
                             goto Repeat;

           case BKSP :       CurSel = (CurSel ?
                                       CurSel - 1 : MenuSize[CurTier] - 1);
                             goto Repeat;

           case CTLL :       ReDraw ();
                             goto Repeat;

           case '\n' :
```

```
            case '\r' :     if (CurSel == (MenuSize[CurTier]-1))
                               return (0);      /* Last item must be quit */
                            return (CurTier * 10 + CurSel + 1);

            case '0'  :     return (0);

            case '1'  :
            case '2'  :
            case '3'  :
            case '4'  :
            case '5'  :
            case '6'  :
            case '7'  :
            case '8'  :
            case '9'  :     CurSel = c - (int) '0' - 1;
                            return (CurTier * 10 + CurSel + 1);

            default   :     return (-1);
           }
      }

   return (-1);
}



int StopMenu ()
{
   return (-1);
}
```

```
#include "console.h"

extern int NeedRead;     /* Global flag indicating need to read config file */
extern char CFileName[80]; /* Global configuration file name */
extern int AskUser;      /* Flag saying to ask user to start system if down */
extern int AutoConfig;   /* Flag indicating auto load of previous config */
extern int CurScreen;

/*
   InitWindow - configure curses and initialize main screen, reading
                menu information and presenting menu line on bottom
                of screen.  (Menu is not a 'window' to this system,
                but rather part of main screen.)
*/

void InitWindow ()

{
   FILE *MenuFp;                          /* File descriptor for menu file */
   int i, j;
   char Temp[5];

   if ((MenuFp = fopen (MENUFILE, "r")) == NULL)
      {
       fprintf (stderr, "%s: cannot open menu file\n", SysName);
       exit (1);
      }

   for (j = 0; j < 2; j++)
      {
       fscanf (MenuFp, "%d\n", &MenuSize[j]);  /* Get number entries in file */

       if (MenuSize[j] > MAXOPTIONS)
          {
           fprintf (stderr, "%s: too many entries in menu file\n", SysName);
           exit (1);
          }

       for (i = 0; i < MenuSize[j]; i++)
           fscanf (MenuFp, "%s", Menu[j][i]);  /* Read next menu item */

      }

   fclose (MenuFp);

   initscr ();                            /* Initialize curses system */

   noecho ();                             /* Turn off character echo */
   crmode ();

   clear ();
   refresh ();

   Command = newwin (2, 80, BOTTOM, 0); /* Define command menu window */

   if ((AutoConfig == TRUE) || (NeedRead == TRUE))
      ReadFile (CFileName);          /* Read configuration file */

   for (i = 0; i < 4; i++)
       {
```

```
        if (CurScreen == SWINDOWS)
           {
            PrintBorder(i, GoodWindow(i)); /* Paint border around each window */

            if (GoodWindow(i))
               PrintTitle (i);
           }

        Window[i].Win = newwin (Window[i].Size, Window[i].Width-2,
                                Window[i].TopLine+1, Window[i].LeftSide+1);
       }

   move (0, 0);
   refresh ();

   scrollok (Window[2].Win, TRUE);     /* Bottom two windows scroll */
   scrollok (Window[3].Win, TRUE);

   if (!Status (FALSE) && (AskUser == TRUE))
      {
       CmdPrompt ("The system is not up.  Do you wish to start it? ",
                  Temp, 4);
       if (tolower(Temp[0]) == 'y')
          {
           RunShell ("taccnet", FALSE);  /* Start TACCNET for the user */
           Status (TRUE);
          }
      }

   PrintMenu ();
}
```

```
#include "console.h"

/*
   LastLines - Position the given file to approximately four lines from
               the end of the file, so that only the last several lines
               will be displayed at startup.
*/

void LastLines (fp)

FILE *fp;  /* File pointer for file to be positioned */

{
   int c;  /* Storage for characters read during scan for newline */
   int i;
   long Length;  /* Storage for length of file */

   fseek (fp, 0L, 2);  /* Position to end of file to determine length */

   Length = ftell (fp);

   if (Length < MINCHARS)
      fseek (fp, 0L, 0);  /* Reposition file to beginning - it is too short */
   else
      {
       fseek (fp, -MINCHARS, 1);  /* Back up the minimum length */
       while ((c = getc(fp)) != '\n')
          if (c == EOF)  /* There were no newlines in all the text */
             fseek (fp, -MINCHARS, 1);  /* Go back again */

       /* File is now near end, at beginning of new line if possible */
      }
}
```

```
#include "console.h"

/*
   NewSys - Monitor a given system by setting the first log file and first
            queue window for that system.
*/

void NewSys ()

{
   char Temp[80];  /* Return storage for CmdPrompt */

   CmdPrompt ("Enter new system name to be monitored in windows 1 & 3 : ",
              Temp, 11);

   if (strlen (Temp))
      {
       strcpy (CurSys, Temp);  /* Make change known if used later */

       wclear (Window[0].Win); /* Get any text off those windows */
       werase (Window[0].Win);
       wclear (Window[2].Win);
       werase (Window[2].Win);

       Window[0].Good = TRUE;  /* Make sure window is good */
       Window[2].Good = TRUE;
       Window[0].LastTime = 0L; /* Reset last mod time */
       Window[2].LastPos = 0L;  /* Reset last file pointer */

       strcpy (Window[0].Name, CurSys);  /* Set queue name */
       sprintf (Window[2].Name, "log/%s.log", CurSys);  /* Set log file name */

       if (Window[0].Fp != (FILE *) NULL)
          {
           fclose (Window[0].Fp);
           Window[0].Fp = (FILE *) NULL;  /* Enforce such conditions */
          }

       if (Window[2].Fp != (FILE *) NULL)
          {
           fclose (Window[2].Fp);
           Window[2].Fp = (FILE *) NULL;  /* Ditto for window 2 */
          }

       ReDraw ();
      }
}
```

```
#include "console.h"

#define BAD -1

/*
   Open - Define a window to be updated with either a queue list or
          a log file tail.
*/

void Open ()

{
   int WindowNum;     /* Number of window to be opened */
   char Temp[80];     /* Temporary string storage */

   CmdPrompt ("Window number to open (1-4) : ", Temp, 2);

   if (strlen (Temp))
      WindowNum = atoi (Temp);  /* Convert to integer */
   else
      WindowNum = BAD;          /* Response was in error */

   if ((WindowNum != BAD) && (WindowNum > 0) && (WindowNum < 5))
      {
       WindowNum--;

       if (GoodWindow(WindowNum))
          {
           if (Window[WindowNum].Fp != (FILE *) NULL)
              {
               fclose (Window[WindowNum].Fp);
               Window[WindowNum].Fp = (FILE *) NULL;
              }
          }

       if (IsLogWindow(WindowNum))
          {
           CmdPrompt ("Enter system name for log window: ", Temp, 40);
           if (strlen (Temp))
              {
               Window[WindowNum].Good = TRUE;
               if (Temp[0] != '/')
                  sprintf (Window[WindowNum].Name, "log/%s.log", Temp);
               else
                  strcpy (Window[WindowNum].Name, Temp);
              }
           else
              WindowNum = BAD;
          }
       else
          {
           CmdPrompt ("Enter system queue name to be monitored: ", Temp, 40);
           if (strlen (Temp))
              {
               Window[WindowNum].Good = TRUE;
               strcpy (Window[WindowNum].Name, Temp);
              }
           else
              WindowNum = BAD;
          }
```

```
        if (WindowNum != BAD)
           {
            wclear (Window[WindowNum].Win);
            werase (Window[WindowNum].Win);

            wrefresh (Window[WindowNum].Win);

            Window[WindowNum].Fp = NULL;
            Window[WindowNum].LastPos = 0L;
            Window[WindowNum].LastTime = 0L;

            PrintBorder (WindowNum, TRUE);
            PrintTitle (WindowNum);
           }
      }
}
```

```
#include "console.h"

extern int CurScreen;  /* Current screen */

/*
   Perform - given a function code (entered from menu), perform desired
             operation, such as opening a new status window, closing
             a window, changing system to monitor, saving configuration,
             etc.
*/

void Perform (code)

int code;  /* Function code to be performed for operator */

{
   char Temp[80];
   int  i;

   switch (code)
      {
       case NEWSYSMON : /* Monitor new system */
                        NewSys ();
                        break;
       case SAVECONF  : /* Save configuration */
                        SaveConf ();
                        break;
       case OPEN      : /* Open one of four static windows if not open */
                        Open ();
                        break;
       case CLOSE     : /* Close one of four static windows if open */
                        Close ();
                        break;
       case SPEED     : /* Set polling interval (faster or slower) */
                        ChangeSpeed ();
                        break;
       case VIEW      : /* View a message using 'vi' - selected via hjkl keys */
                        View ();
                        break;
       case REDRAW    : /* Repaint the screen in the case of a problem */
                        ReDraw ();
                        break;
       case GETCONF   : /* Retrieve a configuration previously saved */
                        GetConf ();
                        break;
       case CLEAR     : /* Clear a window's contents */
                        Clear ();
                        break;
       case ADMIN     : /* Administer files in .../tables */
                        Admin ();
                        break;
       case PORTS     : /* Display current ports table */
                        DispFile ("tables/ports");
                        break;
       case SITES     : /* Display current sites table */
                        DispFile ("tables/sites");
                        break;
       case COMMAND   : /* Run a shell command */
                        CmdPrompt ("Enter TACCNET command: ", Temp, 80);
                        if (strlen (Temp))
```

```
                        RunShell (Temp, TRUE);
                     break;
      case SHELL     : /* Let user execute a subshell (ends with ctrl-D) */
                     UserShell ();
                     break;
      case STATUS    : /* Monitor system status */
                     addstr (" x");  /* Sticky inverse video bug */
                     wclear (Command);
                     wrefresh (Command);
                     refresh ();
                     clear ();
                     refresh ();
                     if (CurScreen != SSTATUS)
                        CurScreen = SSTATUS;
                     else
                        {
                         CurScreen = SWINDOWS; /* Return to main windows */
                         for (i = 2; i < 4; i++)
                              {
                               if (Window[i].Fp != NULL)
                                  {
                                   fclose (Window[i].Fp);
                                   Window[i].Fp = NULL;
                                  }

                               Window[i].LastPos = 0L;

                               wclear (Window[i].Win);
                              }
                        }

                     ReDraw ();
                     break;
      }
}
```

```
#include "console.h"

#define BORDERCHAR(c) ((GoodTerminal == 1) ? ' ' : (flag ? c : ' '))
#define BORDERCHAR(c) ((GoodTerminal == 1) ? ' ' : (flag ? c : ' '))

/*
   PrintBorder - Given a window number, place a border around it on
                 the standard window screen.  If parameter 'flag' is
                 FALSE, erase the border.
*/

void PrintBorder (num, flag)

int num;  /* Window number of window to have border drawn */

{
   int i;

   if (flag && (GoodTerminal == 1))
      standout ();     /* Borders are inverse video */

   move (Window[num].TopLine, Window[num].LeftSide);
   for (i = 0; i < Window[num].Width; i++)
      addch (BORDERCHAR('-'));

   for (i = 1; i < Window[num].Size;  i++)
      {
       mvaddch (Window[num].TopLine + i, Window[num].LeftSide, BORDERCHAR('¦'));
       mvaddch (Window[num].TopLine + i,
                Window[num].LeftSide + Window[num].Width - 1, BORDERCHAR('¦'));
      }

   if (flag && (GoodTerminal == 1))
      standend ();

   refresh ();
}
```

```
#include "console.h"

/*
   PrintMenu - display the menu items at the bottom of the standard screen.
*/

void PrintMenu ()

{
   int i, j;
   int x, y;  /* Coordinates returned from getyx */
   long t;
   char *Clock;

   char *ctime();

   time (&t);
   Clock = ctime (&t);

   wmove (Command, 0, 0);           /* Move to the bottom of the screen */

   if (GoodTerminal == 1)
      wstandout (Command);

   for (i = 11; i < 19; i++)
      waddch (Command, Clock[i]);

   for (i = 8; i < 80; i++)
      waddch (Command, '.');

   wmove (Command, 0, 80-strlen(ThisName));

   waddstr (Command, ThisName);

   if (GoodTerminal == 1)
      wstandend (Command);

   wmove (Command, 1, 0);           /* Move to beginning of menu line */
   wclrtoeol (Command);             /* Erase previous menu */
   waddstr (Command, "Command:");

   waddstr (Command, "  ");

   for (i = 0; i < MenuSize[CurTier]; i++)
       {
        if (i == CurSel)
           {
            wStandOut (Command);     /* Highlight selection if current */

            if (GoodTerminal)
               waddstr (Command, Menu[CurTier][i]); /* Display menu item text */
            else
               for (j = 0; j < strlen (Menu[CurTier][i]); j++)
                   waddch (Command, toupper(Menu[CurTier][i][j]));
           }
        else
           waddstr (Command, Menu[CurTier][i]);

        getyx (Command, y, x);      /* Get current coords */
        wmove (Command, y-1, x-1-strlen(Menu[CurTier][i])/2);
```

```
        if (i == CurSel)
           {
            wStandEnd (Command);
            waddch (Command, 'v');
           }
        else
           {
            wStandOut (Command);

            if (i != MenuSize[CurTier]-1)
               waddch (Command, (char) (i + (int) '0' + 1));
            else
               waddch (Command, '0');

            wStandEnd (Command);     /* Turn off highlight after selection */
           }

        wmove (Command, y, x);  /* Restore old coordinates */
        waddstr (Command, "  ");       /* Space over after each item */
       }

   waddstr (Command, "  ");
   wrefresh (Command);               /* Refresh command window */

   wrefresh (Command);
}
```

```
#include "console.h"

/*
   PrintTitle - Display a window's title in its proper position.
*/

void PrintTitle (num)

int num;  /* Window number for title to be displayed */

{
   int Middle;

   if (GoodWindow(num))
      {
       Middle = Window[num].LeftSide + (Window[num].Width / 2) -
                                       (strlen (Window[num].Name) / 2) - 3;
       move (Window[num].TopLine, Middle);
       StandOut ();
       printw ("%d: ", num+1);
       addstr (Window[num].Name);
       StandEnd ();
       refresh ();
      }
}
```

```
#include "console.h"

/*
   QueueSize - Return number of entries in a given directory (queue).
*/

int QueueSize (queue)

char *queue;  /* Name of queue to be examined */

{
   filename *FileList;
   int i = 0;

   filename *GetDir();

   FileList = GetDir (queue);  /* Get queue list */

   if (FileList == (char **) NULL)
      return (0);

   while (*FileList[i++] != '\0')
      ;

   return (i-1);
}
```

```
#include "console.h"

/*
   ReadFile - Read configuration file into global memory structures.
*/

void ReadFile (conffilename)

char *conffilename;        /* Configuration file name passed in */

{
   int  ConfFileFd;        /* Configuration file pointer */
   WINDOW *TempWin;        /* Temporary window pointer   */
   int  i;

   ConfFileFd = open (conffilename, O_RDONLY);  /* Try to open file */
   if ((ConfFileFd < 0) && (strcmp (conffilename, ".config#") != 0))
      {
       CmdPrompt ("Cannot open configuration file.  Press CR to return: ",
                  conffilename, 80);
       return;
      }
   else if (ConfFileFd < 0)
      return;   /* Could not open previous configuration -- no big loss */

   for (i = 0; i < MAXWINDOWS; i++)
      {
       TempWin = Window[i].Win;     /* Save window pointer */
       if (GoodWindow(i))
          if (Window[i].Fp != (FILE *) NULL)
             fclose (Window[i].Fp);   /* Close the file if open */

       read (ConfFileFd, (char *) &(Window[i]), sizeof (struct window));

       Window[i].Win = TempWin;     /* Only part that isn't saved */
       Window[i].LastPos = 0L;      /* Don't remember old position */
       Window[i].LastTime = 0L;     /* Don't remember old mtime */
      }

   read (ConfFileFd, (char *) &Speed, sizeof (Speed));
   read (ConfFileFd, (char *) CurSys, sizeof (CurSys));

   close (ConfFileFd);
}
```

```
#include "console.h"

extern int CurScreen; /* Current type of screen being displayed */

/*
   ReDraw - Repaint entire screen upon request of user.
*/

void ReDraw ()

{
   int i;

   standend ();  /* Enforce no inverse video condition */
   move (0,0);
   addch ('x');
   refresh ();

   clear ();
   refresh ();

   PrintMenu ();

   touchwin (Command);
   wrefresh (Command);

   if (CurScreen == SWINDOWS)
      {
       for (i = 0; i < MAXWINDOWS; i++)
          {
           PrintBorder (i, GoodWindow(i));

           if (GoodWindow(i))
             {
              touchwin (Window[i].Win);
              wrefresh (Window[i].Win);  /* Refresh the window */
              PrintTitle (i);            /* Print window's title */
             }
          }
      }
   else
      Status (TRUE);

   touchwin (Command);
   wrefresh (Command);
}
```

```
#include "console.h"
#include "wait.h"
#include "signal.h"

static char DefaultShell[] = "/bin/sh";

/*
   RunShell - Run a shell with the user's command in the foreground.
              Return ERR if command could not execute.
*/

int RunShell (shellfile, flag)

char *shellfile;  /* Name of file to be run by Bourne shell */
int  flag;        /* TRUE if user typed command manually    */

{
   int Result;    /* Return code from fork(2) system call   */
   char Path[40]; /* Storage for complete path name         */
   char Temp[80];
   union wait stat_loc;
   char *Shell;

   char *getenv ();
   char *malloc ();
   void ExitProcessor ();

   strcpy (Temp, shellfile);

Top:
   move (0,0);
   addstr ("  x");    /* Kludge city */
   refresh ();

   clear ();
   refresh ();

   echo ();
   nocrmode ();

   signal (SIGINT, SIG_IGN);  /* Ignore interrupt */

   if ((Result = fork ()) == 0)
      {
       signal (SIGINT, SIG_DFL);
       chdir ("bin");
       sprintf (Path, "%s", Temp);

       if ((Shell = getenv ("SHELL")) == (char *) 0)
          {
           Shell = malloc (strlen (DefaultShell) + 1);
           strcpy (Shell, DefaultShell);
          }

       execl (Shell, Shell, "-c", Path, 0);

       printf ("\n\nError: Cannot invoke user shell.  Set your SHELL");
       printf (" environment variable\naccordingly.  ");
       printf ("(%s was not found.)\n", Shell);
       fflush (stdout);
```

```
        exit (1);
       }

   wait (&stat_loc);

   signal (SIGINT, ExitProcessor);

   noecho ();
   crmode ();

   if (stat_loc.w_S.w_Stopval == WSTOPPED)  /* Check for stopped vs. term'ed */
      {
       addstr ("\n\nThe background process was STOPPED unexpectedly; ");
       printw ("The signal was %d\n", stat_loc.w_S.w_Stopsig);
      }

   if (flag == TRUE)  /* The user had typed the previous command himself */
      {
       CmdPrompt ("Next command: ", Temp, 64);

       if (strlen (Temp))
          goto Top;
      }
   else
      CmdPrompt ("Press CR to return to console: ", Temp, 10);

   ReDraw ();

   return (Result);  /* Only in parent's case */
}
```

```
#include "console.h"

/*
   SaveConf - save current configuration for later retrieval.
*/

void SaveConf ()

{
   char ConfFileName[80];   /* Configuration file name    */

   CmdPrompt ("Enter configuration file name: ", ConfFileName, 80);

   WriteConf (ConfFileName);
}
```

```
#include "console.h"

/*
   Select - Given a window number, permit user to move around within
            that window to select one of the files displayed therein
            to be edited.  Return the file's name in the string 'str'.
*/

void Select (num, str)

int  num;  /* Window number that file is to be selected from */
char *str; /* String pointer to place for resultant string   */

{
   CmdPrompt ("Enter message file name: ", str, 80);
}
```

```
#include "console.h"

int NeedRead = FALSE;           /* Flag indicating need to read config */
char CFileName[80];             /* Global storage for config file name */
int AskUser = TRUE;             /* Flag telling to prompt if system down */
int AutoConfig = TRUE;          /* Flag indicating load of old configuration */

extern int CurScreen;

/*
   StartUp - initialize global options from argv and data for windows.
*/

void StartUp (argc, argv)

int  argc;
char **argv;

{
   int i;
   char *CurrentDir;            /* Pointer to new working directory name */

   char *getenv ();
   char *malloc ();
   char *MyName ();

   /* Initialize queue and log window structures -- these are laid out
      so that changes can be made later to allow 'expanding' windows,
      etc.  Running console with '-f' will preload better defaults.     */

   Window[0].Good = FALSE;
   Window[0].Type = QWINDOW;
   Window[0].TopLine = 0;
   Window[0].LeftSide = 1;
   Window[0].Width = 37;
   Window[0].Size = 9;
   Window[0].LastTime = 0L;

   Window[1].Good = TRUE;
   Window[1].Type = QWINDOW;
   Window[1].TopLine = 0;
   Window[1].LeftSide = 42;
   Window[1].Width = 37;
   Window[1].Size = 9;
   Window[1].LastTime = 0L;
   strcpy (Window[1].Name, "msgprocq");

   Window[2].Good = FALSE;
   Window[2].Type = LOGWINDOW;
   Window[2].TopLine = 10;
   Window[2].LeftSide = 0;
   Window[2].Width = 80;
   Window[2].Size = 5;
   Window[2].Fp = NULL;
   Window[2].LastPos = 0L;

   Window[3].Good = FALSE;
   Window[3].Type = LOGWINDOW;
   Window[3].TopLine = 16;
   Window[3].LeftSide = 0;
```

```
   Window[3].Width = 80;
   Window[3].Size = 5;
   Window[3].Fp = NULL;
   Window[3].LastPos = 0L;

   if ((CurrentDir = getenv ("MASTERQ")) == (char *) 0)
      {
       CurrentDir = malloc (strlen (DEFDIR) + 1);
       strcpy (CurrentDir, DEFDIR);  /* Set to default directory */
      }

   strcpy (CFileName, ".config#");  /* Configuration file from last run */

   if (argc > 1)                    /* We have arguments */
      {
       if (argv[1][0] != '-')       /* First argument could be pathname */
          {
           free (CurrentDir);
           CurrentDir = malloc (strlen (argv[1]) + 1);
           strcpy (CurrentDir, argv[1]);     /* Get name of working dir */
           argv++;                  /* Skip to the next entry */
           argc--;
          }

       while (argc > 1)
          {
           if (strcmp (argv[1], "-f") == 0)
              {
               if ((argc < 3) || (argv[2][0] == '-'))
                  strcpy (CFileName, ".config");    /* Read default file */
               else
                  {
                   strcpy (CFileName, argv[2]);     /* Read requested file */
                   argv++;
                   argc--;
                  }

               NeedRead = TRUE;
              }
           else if (strcmp (argv[1], "-i") == 0)
              GoodTerminal = -1;  /* Some inverse video support, not borders */
           else if (strcmp (argv[1], "-I") == 0)
              GoodTerminal = 0;   /* Absolutely no inverse video support */
           else if (strcmp (argv[1], "-S") == 0)
              CurScreen = SSTATUS;
           else if (strcmp (argv[1], "-n") == 0)
              AskUser = FALSE;    /* Don't ask user to start system if down */
           else if (strcmp (argv[1], "-x") == 0)
              AutoConfig = FALSE; /* Don't load old config file unless asked */
           else if (strcmp (argv[1], "-s") == 0)
              {
               if ((argc < 3) || (argv[2][0] == '-'))
                  {
                   fprintf (stderr, "Usage: %s [-f [file]] [-s sysname]\n",
                            SysName);
                   exit (-1);
                  }

               AutoConfig = FALSE;
               strcpy (CurSys, argv[2]);           /* Set current system */
               Window[0].Good = TRUE;
```

```
                Window[2].Good = TRUE;
                strcpy (Window[0].Name, argv[2]);
                sprintf (Window[2].Name, "log/%s.log", argv[2]);

                argv++;
                argc--;
               }

            argv++;
            argc--;
           }
      }

   /* Change to defined working directory (TACCNET root) structure */

   if (chdir (CurrentDir))
      {
       fprintf (stderr, "%s: cannot change directory to %s\n",
                SysName, CurrentDir);
       exit (1);
      }

   ThisName = MyName ();
}
```

```
#include "console.h"

#define MAXPHONENUMS 4
#define DOWN         0
#define RETRY        2

typedef struct          /* site table entry */
   {
    sitename SiteName;                  /* name of the site                */
    short Status;                       /* up, down, priority, busy        */
    short NumCalls;                     /* number of times we called so far */
    long  TimeToCall;                   /* don't call before this time     */
    char  SysType;                      /* operating system type           */
    filename Password;                  /* password for taccnet login      */
    pathname PhoneNum [MAXPHONENUMS+1]; /* array of phone numbers to try   */
   } site ;



typedef struct         /* port table entry */
   {
    char *Port;    /* port name (unix path name)                          */
    char *Site;    /* remote site name, if connected                      */
    int  State;    /* port state (Available, Routine, Priority)           */
   } portentry ;



typedef portentry *portlist;

#define LQMS       "qms.LCK"
#define LMSGPROC   "msgprocq.LCK"
#define LSERVER    "bin/server.LCK"
#define SITETABLE  "tables/sites"
#define PORTTABLE  "tables/ports"

#define UpDown(flag)  (flag ? "UP" : "down")

#define EMULATED     'E'

#define TOP1 8
#define TOP2 34
#define TOP3 61

#define COL1 6
#define COL2 16
```

```
#define COL3 27
#define COL4 39

/*
   Status - check to see whether or not the system is up and running
            in the current queue.  Do this by expecting a .LCK file for
            each process expected to be running.  If the input parameter
            'display' is TRUE, make a point of displaying the results
            on the screen;  otherwise, return TRUE or FALSE if the system
            is up or down, in that order.
*/

int Status (display)

int display;  /* Flag indicating whether or not to display the results */

{
   int QmsUp, MsgUp, SerUp; /* Flags for each module */
   int Fd;      /* File descriptor to be returned from open(2) system call */
   int i, c;
   char SysLockFile[100];
   FILE *SiteTableFp;
   site *NextSite;
   portlist *PortList;
   int  OnLine;
   int  NumFiles = 0;
   int  Found = 0;
   short y;

   site *ReadSite();
   portlist *GetPorts ();
   void PSFree();
   int  QueueSize();
   int  ExitProcessor();
   void LocalExit();

   QmsUp = TRUE;
   MsgUp = TRUE;
   SerUp = TRUE;

   Fd = open (LQMS, O_RDONLY);  /* Try to open qms lock file */

   if (Fd < 0)  /* The qms is not running */
      QmsUp = FALSE;
   else
      close (Fd);

   Fd = open (LMSGPROC, O_RDONLY);  /* Try to open msgproc lock file */

   if (Fd < 0)
      MsgUp = FALSE;  /* The message processor is not running */
   else
      close (Fd);

   Fd = open (LSERVER, O_RDONLY);  /* Try to open server lock file */

   if (Fd < 0)
      SerUp = FALSE;  /* The server isn't running */
   else
      close (Fd);
```

```
    if (display)        /* If the caller requested visual display, give it */
       {
        PrintMenu ();

        move (0, 0);
        StandOut ();

        for (i = 0; i < 80; i++)
            addch (' ');

        move (0, 39 - strlen ("TACCNET System Status")/2);
        addstr ("TACCNET System Status");

        StandEnd ();

        move (2, 0); clrtoeol ();

        move (2, TOP1);  printw ("qms [%s]", UpDown(QmsUp));
        move (2, TOP2);  printw ("msgproc [%s]", UpDown(MsgUp));
        move (2, TOP3);  printw ("server [%s]", UpDown(SerUp));

        addch ('\n');
       }
    else
       return (QmsUp && MsgUp);  /* Indicate if major components are up */

    /* Rest of status display is obviously to be made to screen */

    signal (SIGINT, LocalExit);

    if (Lock (SITETABLE) == -1)
       {
        /* Cannot lock site table */
        addch ('\n');
        StandOut();
        addstr ("Error");
        StandEnd();
        addstr (": can't lock site table; probably left locked accidentally.\n");
        addch ('\n');
        refresh ();
       }

    SiteTableFp = fopen (SITETABLE, "r");

    if (SiteTableFp == (FILE *) 0)
       {
        addstr ("\n\nCannot open site table; no status is available.\n");
        goto WrapUp;
       }

    (void) getc (SiteTableFp);  /* Digest initial field marker */

    addch ('\n');

    y = stdscr->_cury;

    move (y, COL1);  addstr ("Site");
    move (y, COL2);  addstr ("Status");
    move (y, COL3);  addstr ("Line");
    move (y, COL4);  addstr ("Messages in Queue");
```

```
   y++;

   move (y, COL1);  addstr ("----");
   move (y, COL2);  addstr ("------");
   move (y, COL3);  addstr ("----");
   move (y, COL4);  addstr ("-----------------");

   while ((!feof (SiteTableFp)) && (!ferror (SiteTableFp)))
      {
       y++;

       move (y, 0);

       clrtoeol ();

       NextSite = ReadSite (SiteTableFp);  /* Read NEXT site */

       sprintf (SysLockFile, "%s.LCK", NextSite->SiteName);

       move (y, COL1);

       OnLine = FALSE;

       if ((Fd = open (SysLockFile, O_RDONLY)) >= 0)  /* Lock file exists */
          {
           StandOut();             /* Highlight active (on-line) entries */
           if (!GoodTerminal)
              for (i = 0; NextSite->SiteName[i] != '\0'; i++)
                  NextSite->SiteName[i] = toupper (NextSite->SiteName[i]);

           close (Fd);
           OnLine = TRUE;
           /* To maintain a global list of on-line sites, add code here */
          }

       addstr (NextSite->SiteName);  /* Display site name */

       StandEnd();

       move (y, COL2);

       if (OnLine == TRUE)           /* Don't try to say "if (OnLine)"... */
          {
           addstr ("on-line");

           Lock (PORTTABLE);         /* Don't care if it fails */

           move (y, COL3);

           if ((PortList = GetPorts ()) == NULL)
              {
               addstr ("[error]");
               UnLock (PORTTABLE);
              }
           else
              {
               UnLock (PORTTABLE);

               Found = 0;

               for (i = 0; (!Found && (PortList[i] != NULL)); i++)
```

```
                if (strcmp (PortList[i]->Site, NextSite->SiteName) == 0)
                   Found = 1;

             if (Found)
                {
                 int j, k;

                 for (j = strlen (PortList[i-1]->Port); j > 0; j--)
                     if (PortList[i-1]->Port[j] == '/')
                        break;

                 for (k = j+1; PortList[i-1]->Port[k] != '\0'; k++)
                     addch (PortList[i-1]->Port[k]); /* Display port */
                }
             else
                addstr ("(dialin)");     /* Site must have called us */

             PSFree (PortList);
            }
      }
   else
      {
       if (NextSite->SysType == EMULATED)
          addstr ("emulated ");
       else
          switch (NextSite->Status)      /* Display site status */
             {
              case DOWN   : addstr ("DOWN");
                            break;
              case RETRY  : addstr ("retry");
                            if (NextSite->NumCalls)
                               printw ("(%d)", NextSite->NumCalls);
                            break;
              default     : break;
             }
      }

   /* Find out how many entries are in the system's queue */

   NumFiles = QueueSize (NextSite->SiteName);  /* Get site's queue size */

   move (y, COL4);

   if (NumFiles)
      printw ("%d", NumFiles);

   move (y, COL4 + 4);

   StandOut ();

   for (i = 0; (i < NumFiles) && (i < 79-COL4-4); i++)
       addch ('*');

   if (NumFiles > 29)
      addstr ("\b+");

   clrtoeol ();

   StandEnd ();
  } /* End of major 'while ((!feof(...))...)' */
```

```
    wclrtobot (stdscr);
    fclose (SiteTableFp);
    signal (SIGINT, ExitProcessor);  /* Restore interrupt vector */
WrapUp:
    UnLock (SITETABLE);
    refresh ();
}



void LocalExit ()
{
    UnLock (SITETABLE);
    ExitProcessor();      /* Now you can exit */
}
```

```
#include "console.h"

#define BUZZER 5      /* Buzzer set for 5 seconds */

int Buzzer = FALSE;  /* Timer sets Buzzer after while */

/*
   TailFile - read and display contents of a file, starting at the
              current file position, until end of file.  The file
              descriptor and the curses window are given by the
              Window global structure, indexed by input parameter
              'num'.  Don't run more than BUZZER seconds.
*/

TailFile (num)

int num;        /* Window number of window into which file should be listed */

{
   char String[150];      /* Storage for each line to be read and displayed */
   FILE *Fp;              /* File descriptor for window */
   char *Result;
   int  i, n;

   char *fgets();
   long ftell();
   void SetBuzzer();      /* Signal catcher for alarm */

   signal (SIGALRM, SetBuzzer);

   Fp = Window[num].Fp;  /* Ease naming convention */

   if (Fp == NULL) /* We will need to open it before we continue */
      {
       Fp = fopen (Window[num].Name, "r");  /* Try to reopen the file */
       if (Fp != NULL)
          {
           if (Window[num].LastPos > 0L)
              fseek (Fp, Window[num].LastPos, 0);         /* Find old EOF */
           else
              LastLines (Fp);            /* Position to last part of file */
          }
      }

   alarm (0);                 /* Reset clock */

   Buzzer = FALSE;

   alarm (BUZZER);            /* Always return after BUZZER seconds */

   while ((Fp != NULL) && !Buzzer) /* If the file is open, continue 'til end */
      {
       Window[num].LastPos = ftell (Fp);  /* Remember where we were */
       Result = fgets (String, 149, Fp);  /* Get the next line from input */

       if (Result == NULL)    /* The file is ended */
          {
           fclose (Fp);       /* The stream will need to be reopened */
           Fp = NULL;         /* Enforce this condition */
          }
```

```
        else
            {
             n = 0;
             for (i = 0; i < strlen (String); i++)
                 {
                  if ((String[i] != '\r') && (String[i] != '\n'))
                      {
                       waddch (Window[num].Win, String[i]);
                       n++;
                      }
                  else if ((n != 78) && (String[i] != '\r'))
                      {
                       waddch (Window[num].Win, '\n');
                       n = 0;
                      }
                 }

             wrefresh (Window[num].Win);
            }
       }

    Window[num].Fp = Fp;        /* Copy new pointer back */

    alarm (0);

    signal (SIGALRM, SIG_DFL);
}




void SetBuzzer ()

{
    Buzzer = TRUE;
}
```

```
#include "console.h"

/*
   Update - Call the appropriate routine for the given window, to display
            current information (status) in that window.  Two types of
            displays exist: Queue Displays (file lists), and Logfile Displays
            (parallel to 'tail -f logfile').  Static information
            for each window is kept in the global Window data structure.
*/

Update (num)

int num;  /* Number of window to be updated */

{
   if (IsLogWindow(num))    /* If the window is a log window */
      {
       TailFile (num);      /* Halfway simulate Unix 'tail -f' command */
      }
   else                     /* The window must be a queue window */
      {
       DispQueue (num);     /* Display the queue in window 'num' */
      }
}
```

```
#include "console.h"

static char DefaultShell[] = "/bin/sh";

/*
   UserShell - run a shell for the user, such as /bin/sh.  Eventually
               use the environment variable to indicate which shell to
               run, but for now just run /bin/csh.
*/

void UserShell ()

{
   char *Shell;

   char *getenv ();
   char *malloc ();
   int ExitProcessor ();

   echo ();       /* Give user a sane terminal */
   nocrmode ();

   move (0,0);
   addstr (" x ");
   refresh ();

   clear ();
   refresh ();

   signal (SIGINT, SIG_IGN);

   if (fork () == 0)
      {
       signal (SIGINT, SIG_DFL);

       if ((Shell = getenv ("SHELL")) == (char *) 0)
          {
           Shell = malloc (strlen (DefaultShell) + 1);
           strcpy (Shell, DefaultShell);
          }

       execl (Shell, Shell, 0);

       printf ("\n\nCannot execute user shell.  Define the SHELL environment");
       printf (" variable to point\nto your favorite shell.  (Could not");
       printf (" find %s.)\n", Shell);
       printf ("\n\n\n\n\nPress CR to return to menu: ");
       fflush (stdout);

       getchar ();

       exit (1);
      }

   wait ((int *) 0);

   signal (SIGINT, ExitProcessor);

   ReDraw ();  /* Repaint the windows, etc. */
```

```
    noecho ();
    crmode ();
}
```

```
#include "console.h"

static char DefaultEditor[] = "/bin/vi";

/*
   View - Permit user to select a file within a queue window to be viewed
          using the defined editor.  During the period the file is being
          edited, the console display will be turned off so that the
          entire screen may be used for the editor session.  (This is
          also much easier to implement.)
*/

void View ()

{
   char Temp[80];  /* Storage for CmdPrompt calls */
   int WindowNum;  /* Window number from which file will be selected */
   int Fd;         /* File descriptor */
   char MsgName[80];
   char Msg[80];   /* Message space */
   char *Editor;

   char *getenv ();
   char *malloc ();
   void ExitProcessor ();  /* Declare exit routine */

   CmdPrompt ("Enter window number from which message will be selected (1-2): ",
              Temp, 80);

   if (strlen (Temp))
      {
       WindowNum = atoi (Temp) - 1;  /* Convert to integer */

       if ((WindowNum == 0) || (WindowNum == 1))
          if (GoodWindow(WindowNum))  /* Only operate on open windows */
             {
              Select (WindowNum, Temp);  /* Select a file name into Temp */

              if (strlen (Temp) == 0)
                 return;

              sprintf (MsgName, "%s/%s", Window[WindowNum].Name, Temp);

              if ((Fd = open (MsgName, O_RDONLY)) < 0)  /* Try to open file */
                 {
                  sprintf (Msg, "Message %s does not exist; press CR %s",
                           MsgName, "to continue: ");
                  CmdPrompt (Msg, Temp, 4);
                  return;
                 }

              close (Fd);

              clear ();
              refresh ();
              echo ();
              nocrmode ();

              signal (SIGINT, SIG_IGN);  /* Disable interrupts */
```

```
            if (fork () == 0)  /* I am the child */
               {
                signal (SIGINT, SIG_DFL);
                addstr ("Please stand by..");
                refresh ();

                if ((Editor = getenv ("VIEWER")) == NULL)
                   if ((Editor = getenv ("EDITOR")) == NULL)
                      {
                       Editor = malloc (strlen (DefaultEditor) + 1);
                       strcpy (Editor, DefaultEditor);
                      }

                execl (Editor, Editor, MsgName, 0); /* View the file */

                printf ("\n\nError: could not load viewer/editor.\n");
                printf ("Define an environment variable VIEWER or EDITOR");
                printf (" to be the pathname of\nthe program you wish");
                printf (" to utilize here.\n\n\n");
                fflush (stdout);

                exit (0);
               }
            else
               wait ((int *) 0);  /* Wait for child to terminate */

            noecho ();
            crmode ();

            signal (SIGINT, ExitProcessor);

            CmdPrompt ("Press CR to continue: ", Temp, 2);

            ReDraw ();              /* Repaint the screen after viewing */
           }
      }
}
```

```
#include "console.h"

/*
   WriteConf - write current configuration to the file named.
*/

void WriteConf (cfilename)

char *cfilename;   /* File to write the configuration data into [sic] */

{
   int  ConfFileFd;         /* Configuration file pointer */
   WINDOW *TempWin;         /* Temporary window pointer   */
   char Temp[4];
   int  i;

   if (strlen (cfilename))
      {
       ConfFileFd = open (cfilename, O_WRONLY|O_CREAT, 0666);
       if ((ConfFileFd < 0) && (strcmp (cfilename, ".config#") != 0))
          {
           CmdPrompt ("Cannot create configuration file.  Press CR to return: ",
                      Temp, 4);
           return;
          }
       else if (ConfFileFd < 0)
          return;    /* No great loss */

       for (i = 0; i < MAXWINDOWS; i++)
          {
           TempWin = Window[i].Win;          /* Save window pointer        */
           Window[i].Win = (WINDOW *) NULL;  /* Will make no sense in file */
           Window[i].Fp = (FILE *) NULL;     /* Will also make no sense    */

           write (ConfFileFd, (char *) &(Window[i]), sizeof (struct window));

           Window[i].Win = TempWin;          /* Restore window pointer     */
          }

       write (ConfFileFd, (char *) &Speed, sizeof (Speed));
       write (ConfFileFd, (char *) CurSys, sizeof (CurSys));

       close (ConfFileFd);
      }
}
```

```
WriteLog(a,b,c,d)
char *a, *b, *c, *d;
{
}
```

C Language Program Listings for the TACCNET System GCOS Implementation

by

Principal Investigator:
Alton P. Jensen, Professor

Project Manager:
William O. Putnam, Research Scientist II

Project Staff:
Steven L. Goldberg, Research Assistant
Hong S. Shinn, Graduate Research Assistant

School Of Information and Computer Science
Georgia Institute Of Technology
Atlanta, Georgia 30332

Presented to:

U.S. Army Institute for Research In Management Information, Communications, and Computer Science (AIRMICS)

April 30, 1987

Contract No. DAHC0-85-C00012
Research Project No. G36-633

Introduction

The GCOS implementation of the TACCNET system is divided into three programs, each of which is composed of a number of functions ‡. The three programs are GENMSG, CALLER and IOCONTROL. These are the basic elements of TACCNET necessary for a system to function as a TACCNET network node. The other TACCNET programs, QMS, MSGPROC, and SERVER were not implemented under GCOS because they are not required for the DAS3 elements in the CSSCS network.

Each function or group of related functions is contained in a file. These files are compiled with the M4_CC compiler and linked with a main program segment to form an executable program. Each program has a primary function called *main()* in the file with the same name as the program.

All C language source file names end in the suffix ".c". Files whose names end in ".h" are header files containing global constants and data structure definitions. Files whose names end in ".e" contain declarations of external variables to be used by subprograms.

The job files used to compile and link the TACCNET programs are provided at the beginning of the source listings.

The reader of this document is expected to be familiar with the contents of the TACCNET technical specification titled *Considerations in the Design and Development of a Combat Service Support Computer System* wherein the usage and operation of these programs is explained.

It will be difficult for persons not familiar with the C programming language to use this document. It will also be helpful to be familiar with the UNIX and GCOS operating systems under which TACCNET was developed. The definitive reference for the C language is *The C Programming Language* by Brian Kernighan and Dennis Ritchie (Prentice-Hall, 1978). There are numerous introductory books on the C language and the UNIX† operating system available.

---

‡ In the C programming language subprograms are called functions. A program must consist of at least one function but may call many other functions. Functions may call other functions and may even call themselves recursively.

† UNIX is a trademark of AT&T Bell Laboratories.

GCOS Job Files for Compilation and Linking

This section contains the listings of the GCOS command files used to compile and link the TACCNET programs GENMSG, CALLER, and IOCONTROL.

```
PATH:  ^ZSYS72>UDD>GOLDBERG>UNIX>COMP1.EC

TIME:  1987/02/12 1732:42.6


M4_CC IOCONT.C -AS -OP -D PROTOCOL_DBG
M4_CC CHECKS.C -AS -OP -D PROTOCOL_DBG
M4_CC GETBLO.C -AS -OP -D PROTOCOL_DBG
M4_CC GETFIL.C -AS -OP -D PROTOCOL_DBG
M4_CC GETPAC.C -AS -OP -D PROTOCOL_DBG
M4_CC PREEMP.C -AS -OP -D PROTOCOL_DBG
M4_CC RECEIV.C -AS -OP -D PROTOCOL_DBG
M4_CC SEND.C -AS -OP -D PROTOCOL_DBG
M4_CC SENDBL.C -AS -OP -D PROTOCOL_DBG
M4_CC SENDBY.C -AS -OP -D PROTOCOL_DBG
M4_CC SENDEN.C -AS -OP -D PROTOCOL_DBG
M4_CC SENDNA.C -AS -OP -D PROTOCOL_DBG
M4_CC SENDFI.C -AS -OP -D PROTOCOL_DBG
M4_CC SENDPA.C -AS -OP -D PROTOCOL_DBG
M4_CC WAITAC.C -AS -OP -D PROTOCOL_DBG
M4_CC ARCHIV.C -AS -OP -D PROTOCOL_DBG
M4_CC WAITNA.C -AS -OP -D PROTOCOL_DBG
M4_CC FILENQ.C -AS -OP -D PROTOCOL_DBG
M4_CC MOVMEM.C -AS -OP -D PROTOCOL_DBG
M4_CC WAITEN.C -AS -OP -D PROTOCOL_DBG
M4_CC DEQUEU.C -AS -OP -D PROTOCOL_DBG
M4_CC WRITEL.C -AS -OP -D PROTOCOL_DBG
M4_CC DATETI.C -AS -OP -D PROTOCOL_DBG
M4_CC VALIDS.C -AS -OP -D PROTOCOL_DBG
M4_CC MYNAME.C -AS -OP -D PROTOCOL_DBG
M4_CC READDI.C -AS -OP -D PROTOCOL_DBG
M4_CC LOCK.C -AS -OP -D PROTOCOL_DBG
M4_CC UNLOCK.C -AS -OP -D PROTOCOL_DBG
M4_CC RDSITE.C -AS -OP -D PROTOCOL_DBG
M4_CC FILEOP.C -AS -OP -D PROTOCOL_DBG
M4_CC SETPOR.C -AS -OP -D PROTOCOL_DBG
M4_CC REMOVE.C -AS -OP -D PROTOCOL_DBG
M4_CC CREATE.C -AS -OP -D PROTOCOL_DBG
M4_CC SENDHE.C -AS -OP -D PROTOCOL_DBG
M4_CC GETHEA.C -AS -OP -D PROTOCOL_DBG


EOF
```

```
PATH:  ^ZSYS72>UDD>GOLDBERG>UNIX>COMPJOB.EC

TIME:  1987/02/12 1734:23.9


M4_CC CALLER.C -AS -LE
M4_CC CHECKDOWN.C -AS -LE
M4_CC CHECKSUM.C -AS -LE
M4_CC CREATEFILE.C -AS -LE
M4_CC DATETIME.C -AS -LE
M4_CC DEQUEUE.C -AS -LE
M4_CC DIAL.C -AS -LE
M4_CC DIR.C -AS -LE
M4_CC ENDSWITH.C -AS -LE
M4_CC FILENQ.C -AS -LE
M4_CC FILEOPEN.C -AS -LE
M4_CC FRENAME.C -AS -LE
M4_CC GETBLOCK.C -AS -LE
M4_CC GETFILE.C -AS -LE
M4_CC GETHEADER.C -AS -LE
M4_CC GETPACKET.C -AS -LE
M4_CC GETPROMPT.C -AS -LE
M4_CC GIVETOMP.C -AS -LE
M4_CC HANGUP.C -AS -LE
M4_CC IOCONTROL.C -AS -LE
M4_CC LOCKFILE.C -AS -LE
M4_CC LOGIN.C -AS -LE
M4_CC MOVMEM.C -AS -LE
M4_CC MYNAME.C -AS -LE
M4_CC NEWFILE.C -AS -LE
M4_CC OPENMODEM.C -AS -LE
M4_CC PORTS.C -AS -LE
M4_CC PREEMPTION.C -AS -LE
M4_CC PUTSITE.C -AS -LE
M4_CC READSITE.C -AS -LE
M4_CC READSTR.C -AS -LE
M4_CC RECEIVE.C -AS -LE
M4_CC REMOVE.C -AS -LE
M4_CC SEND.C -AS -LE
M4_CC SENDBLOCK.C -AS -LE
M4_CC SENDBYTE.C -AS -LE
M4_CC SENDENQ.C -AS -LE
M4_CC SENDFILE.C -AS -LE
M4_CC SENDHEADER.C -AS -LE
M4_CC SENDNAME.C -AS -LE
M4_CC SENDPACKET.C -AS -LE
M4_CC SETHAYES.C -AS -LE
M4_CC SETPORT.C -AS -LE
M4_CC STATE.C -AS -LE
M4_CC STRIPME.C -AS -LE
M4_CC VALIDSITE.C -AS -LE
M4_CC WAITACK.C -AS -LE
M4_CC WAITENQ.C -AS -LE
M4_CC WAITNAME.C -AS -LE
M4_CC WRITELOG.C -AS -LE


EOF
```

```
PATH:   ^ZSYS72>UDD>GOLDBERG>UNIX>COMPJOB2.EC

TIME:   1987/02/12 1735:18.1


M4_CC DIR.C -MR -LE -AS
M4_CC LOCKFILE.C -MR -LE -AS
M4_CC OPENMODEM.C -MR -LE -AS
M4_CC SENDBLOCK.C -MR -LE -AS
M4_CC SENDHEADER.C -MR -LE -AS
M4_CC SETPORT.C -MR -LE -AS


EOF
```

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>CALLER.LN

TIME: 1987/02/12 1738:07.7

CALLER
GIVETO
CHECKD
DIAL
VALIDS
PUTSIT
OPENMO
STATE
HANGUP
WRITEL
FRENAM
LOCK
UNLOCK
RDSITE
DATETI
LOGIN
MYNAME
FILEOP
PORTS

EOF

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>IOCONT.LN

TIME: 1987/02/12 1738:40.0

```
IOCONT
CHECKS
GETBLO
GETFIL
GETPAC
PREEMP
SEND
SENDBL
SENDBY
SENDEN
SENDNA
SENDFI
SENDPA
WAITAC
ARCHIV
WAITNA
FILENQ
MOVMEM
WAITEN
DEQUEU
WRITEL
DATETI
VALIDS
MYNAME
READDI
LOCK
UNLOCK
RDSITE
FILEOP
SETPOR
REMOVE
CREATE
SENDHE
GETHEA


EOF
```

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>GENMSG.LN

TIME: 1987/02/12 1738:25.5

GENMSG
VALIDS
VALIDP
MYNAME
FILENQ
WRITEL
DATETI
NEWFIL
FILEOP
LOCK
UNLOCK
READSI
STRIPM

EOF

Common Functions

This section contains common functions used by many different programs and functions in the TACCNET system. If the source code for a function is not given in the separate program listings it will be in this section.

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>DATETI.C

TIME: 1987/02/12 1601:26.1

```
#include "net.h"
int DateTime (Str)
char *Str;
{
   long BDate;
   BDate = time ((long *) 0);
   sprintf (Str, "%s", ctime(&BDate));
   Str [strlen (Str)-1] = '\0';          /* Zap NL placed by TIME (2) */
   return (0);
}
```

EOF

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>DATETM.C

TIME: 1987/02/12 1601:59.3

```
#include "net.h"
int DateTime (Str)
char *Str;
{
   long BDate;
   BDate = time ((long *) 0);
   sprintf (Str, "%s", ctime(&BDate));
   Str [strlen (Str)-1] = '\0';          /* Zap NL placed by TIME (2) */
   return (0);
}
```

EOF

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>DEQUEU.C

TIME: 1987/02/12 1602:24.1

```
#include "net.h"
/*
   DeQueue - Return the name of the first message in the input queue
             opened via GetDir. Return NULL if the queue is empty.
*/

int CurFilePtr;    /* Pointer to current filename */


char *DeQueue (MsgQueueName)

char *MsgQueueName;

{
   static char **MsgFiles;    /* Pointer to all the files in the directory */

   if ((CurFilePtr == 0) || (MsgFiles [CurFilePtr] == NULL))
      {
       /* List exhausted, read again for more file names */

       MsgFiles = ReadDir (MsgQueueName);
       CurFilePtr = 0;  /* Point to the first filename */
      }

   if (MsgFiles == (char **) NULL)
      return (NULL);  /* Return end of list condition */

   CurFilePtr++;       /* Point to next file for later */

   return (MsgFiles [CurFilePtr-1]);  /* Return the file name */
}


EOF
```

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>DIR.C

TIME: 1987/02/12 1603:42.0

```
#include "net.h"
#define NFILES 100

/*char Entries [NFILES][FILENAMELEN];*/

/*
   GetDir - Return directory structure expected by "dequeue()" containing
            entries in directory specified by parameter Dir.  This version
            is rewritten specifically for the Honeywell DPS/6.
*/

char **GetDir (Dir)
char *Dir; /* Name of directory to read */

{
   char *star_name(); /* Honeywell directory function */

   char *DirEntry;  /* Pointer to directory entries */
   register int Num = 0;
   char **Entries;  /* Returned vector containing directory entries */

   Entries = (char **) malloc (NFILES * (sizeof (char *)));

   /* Call Honeywell-specific routine to get directory listing */

   DirEntry = star_name ("*", Dir);  /* Match all file names for now */

   while (*(DirEntry++) != '\0')
      {
       Entries [Num] = malloc (strlen (DirEntry) + 1);
       strcpy (Entries [Num], DirEntry);
       DirEntry += strlen (Entries[Num]) + 1;  /* Bump pointer past name */
       Num++;
      }

   if (Num > 0)
      {
       Sort (Entries, Num);
       *Entries [Num] = '\0';                  /* Set the last one to NULL */
       return (Entries);
      }
   else
       return (NULL);
}

Sort (Entries, Num)
filename Entries [];
int Num;
{
   /* register */ int Gap, i, j;
```

```
    filename Temp;
    for (Gap = Num / 2; Gap > 0; Gap /= 2)
        for (i = Gap; i < Num; i++)
            for (j = i - Gap; j >= 0; j -= Gap)
                {
                 if (strcmp (Entries [j], Entries [j+Gap]) <= 0)
                     break;
                 strcpy (Temp, Entries [j]);
                 strcpy (Entries [j], Entries [j+Gap]);
                 strcpy (Entries [j+Gap], Temp);
                }
}


EOF
```

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>ENDSWI.C

TIME: 1987/02/12 1605:28.0

```
#include "net.h"
int EndsWith (String, Target)
char *String;
char *Target;
{
   while ( *Target != '\0' )
       if (EQUALS(String, Target++))
          return (TRUE);
   return (FALSE);
}
```

EOF

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>FILENQ.C

TIME: 1987/02/12 1605:43.9

```
#include "net.h"
/*
   FileNQ - Make a file named by FileName visible in the directory given
            in Queue by moving it up from the ZTEMP subdirectory.  Return
            TRUE or FALSE result.
            Files left in ZTEMP will be removed at end of run.
*/

int FileNQ (FileName, Queue)

char *FileName;
char *Queue;

{
   pathname PathName;  /* Storage for directory path */
   char CmdText [100];
   pathname DirName;

   char *getdir ();

   getdir (DirName, 0);  /* Get name of working directory */

   sprintf (PathName, "%s>%s", Queue, "ZTEMP");

   if (chdir (PathName) != 0)
      {
       WriteLog ("FileNQ:", "cannot change directory to", Queue, "");
       return (ERR);
      }

   sprintf (CmdText, "CP %s <==", FileName); /* Prepare stmt to copy up */

   if (system (CmdText))   /* Rename the file, removing the first char */
      {
       WriteLog ("FileNQ: can't copy", FileName, "from temp directory to",
                 Queue);
       chdir (DirName);
       return (ERR);
      }

   chdir (DirName);

   return;
}
```

EOF

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>FILEOP.C

TIME: 1987/02/12 1606:25.9

```
#include "net.h"
/*
   FileOpen - Open the file FileName in the queue QueueName by constructing
              the full pathname of the file given these two components.
              Return a file descriptor to the file, or NULL if the file
              could not be opened.
*/
FILE *FileOpen (FileName, QueueName, FileType)
char *FileName;
char *QueueName;
char *FileType;
{
   pathname TempFileName;  /* Complete pathname of file to be opened */

   if (strlen (QueueName) != 0)
      sprintf (TempFileName, "%s/%s", QueueName, FileName);
   else
      strcpy (TempFileName, FileName);

   return (fopen (TempFileName, FileType));
}
```

EOF

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>FRENAM.C

TIME: 1987/02/12 1607:40.9

```
#include "net.h"
int FRename (Path1, Path2)
char *Path1;
char *Path2;
/* Renames Path1 to Path2 */
{
   if (link (Path1, Path2) == ERR)
       return (ERR);
   if (unlink (Path1) == ERR)
       return (ERR);
   return (GOOD);
}
```

EOF

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>LOCK.C

TIME: 1987/02/12 1626:59.3

```
#include "net.h"
int Lock (FileName)
/*  returns ERR if the file is already locked, GOOD otherwise */
char *FileName;
/*
   Simple file locking mechanism using open(). We try to lock FileName by
   creating a file called FileName.LCK. Setting Oflags O_CREAT and O_EXCL
   causes open() to return ERR if the file already exists.
*/
{
   pathname LockFile;
   register int i,Fd;
   int Oflags = O_CREATE | O_EXCL;   /* return ERR if file exists */
   int Mode = 0664;
   sprintf (LockFile, "%s.LCK", FileName);
   /* try and lock the file several times before giving up */
   for (i = 0; i < 10; i++)
       if ((Fd = open (LockFile, Oflags, Mode)) == ERR)
           sleep (2);
       else
           break;
   if (Fd == ERR)
      return (ERR);       /* file was already locked */
   else
      {
       close (Fd);        /* don't forget to close the file */
       return (GOOD);     /* file is now locked */
      }
}
```

EOF

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>MOVMEM.C

TIME: 1987/02/12 1638:55.9

```
#include <stdio.h>
/*
   movmem - copy memory contents from one location to another.  Locations
            are passed in as string pointers indicating 'from' and 'to'
            addresses, respectively. Simply copy bytes one-by-one
            until all bytes copied.  Length is the third parameter.
*/
movmem (from, to, length)
unsigned char from[];  /* Memory address to copy from */
unsigned char to[];    /* Address to copy to */
int  length;  /* Length (in bytes) to copy */
{
    register int i=0;
    while (i < length)
        {
         to[i] = from[i];
         i++;
        }
}
```

EOF

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>MYNAME.C

TIME: 1987/02/12 1639:23.1

```
#include "net.h"
/*
   MyName - Return the name of this site, found in the file MYNAME.
*/
char *MyName ()
{
   sitename TempSiteName;    /* Temporary storage for site name */
   FILE     *MyNameFd;       /* MYNAME file descriptor */
   char     *RetPtr;         /* Pointer to return to caller */
   if ((MyNameFd = FileOpen (MYNAME, MASTERQ, "r")) == NULL)
      {
       fprintf (stderr, "MyName: I don't know my own name.\n");
       fclose (MyNameFd);
       return ((char *) NULL);  /* Try to recover */
      }

   fscanf (MyNameFd, "%s", TempSiteName);     /* Get this site's name */
   fclose (MyNameFd);
   RetPtr = malloc (strlen (TempSiteName) + 1);   /* Get perm. storage */
   strcpy (RetPtr, TempSiteName);
   return (RetPtr);
}
```

EOF

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>NEWFIL.C

TIME: 1987/02/12 1639:54.2

```
#include "net.h"
/*
          NewFile - build a file name in the directory given by Queue, in the
                    ZTEMP subdirectory so as to make it invisible. Return
                    the name of the file in FileName. Open the file and
                    return a file descriptor.  The file name will be
                    composed of the system name plus a date-time stamp.
*/

FILE *NewFile (FileName, Type, Queue)

char *FileName;      /* pointer to new file name */
int   Type;          /* character to indicate message type */
char *Queue;         /* directory in which to put the file */

{
   pathname PathName;      /* pointer to the full path name of the file */
   long int AbsTime;

   AbsTime = NOW;

   sprintf (FileName, "%c%s%x%.4x", Type, MyName(),
               (int) ((AbsTime & (long) 0x0f0000) >> 16);
               (int) (AbsTime & (long) 0x00ffff));

   sleep (1);  /* Ensure that clock increments */

   sprintf (PathName, "%s/ZTEMP/%s", Queue, FileName);

   return (fopen(PathName, "w"));
}
```

EOF

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>OPENMO.C

TIME: 1987/02/12 1641:41.7

```
#include "net.h"
extern int ModemFd;

int OpenModem (PortName)
char *PortName;
/*
Open the named port for use by IOControl.  Line parameters cannot be set
by this program, since the Honeywell C compiler does not have support for the
IOCTL call in Unix.  Sets global variable ModemFd to the file descriptor for
the modem, ERR otherwise.
*/
{
   register int OFlag = O_RDWR;
   if ((ModemFd = open (PortName, OFlag)) == ERR)
      {
       WriteLog ("OpenModem:", "Can't open", PortName, "");
       return (ERR);
      }
    return (ModemFd);
}
```

EOF

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>PORTS.C

TIME: 1987/02/12 1642:08.4

```
#include "net.h"
int TakePort (Port, Site, Type)
char *Port;
char *Site;
int  Type;
/*
   Get the port list and rewrite the PORTTABLE indicating that the named
   Port is being used to call the named Site with a transmission of the
   given Type.  Assume that the PORTTABLE is already LOCKED.
*/
{
   portlist *PortList;
   register int i, Found=FALSE;
   if ((PortList = GetPorts ()) ==  NULL)
      {
       WriteLog ("TakePort: can't read port list to mark", Port, "in use", "")
       return (ERR);
      }
   for (i=0; (PortList[i] != NULL); i++)
      if (EQUALS(PortList[i]->Port, Port))
          {
           Found = TRUE;
           break;
          }
   if (!Found)
      {
       WriteLog ("TakePort: can't find", Port, "in", PORTTABLE);
       UnLock (PORTTABLE);
       free (PortList);
       return (ERR);
      }
   free (PortList[i]->Site);
   PortList[i]->Site = malloc (strlen (Site)+1);
   strcpy (PortList[i]->Site, Site);
   PortList[i]->State = Type;
   if (PutPorts (PortList) ==  ERR)
      {
       WriteLog ("TakePort: can't write port list to mark", Port, "in use", ""
       free (PortList);
       return (ERR);
      }
   free (PortList);
   return (GOOD);
}
int FreePort (Port)
char *Port;
/*
   Finds the named Port in the PORTTABLE and changes the port entry to
   indicate that the port is no longer in use.
   Assumes the PORTTABLE is UNLOCKED and does its own locking.
```

```
*/
{
   portlist *PortList;
   register int i, Found=FALSE;
   if ((Lock (PORTTABLE)) == ERR)
      {
       WriteLog ("FreePort: Can't lock", PORTTABLE, "", "");
       return (ERR);
      }
   if ((PortList = GetPorts ()) == NULL)
      {
       WriteLog ("FreePort: can't read port list to mark", Port, "in use", "")
       UnLock (PORTTABLE);
       return (ERR);
      }
   for (i=0; (PortList[i] != NULL); i++)
      if (EQUALS(PortList[i]->Port, Port))
         {
          Found = TRUE;
          break;
         }
   if (!Found)
      {
       WriteLog ("FreePort: can't find", Port, "in", PORTTABLE);
       UnLock (PORTTABLE);
       free (PortList);
       return (ERR);
      }
   free (PortList[i]->Site);
   PortList[i]->Site = malloc (strlen ("free")+1);
   strcpy (PortList[i]->Site, "free");
   PortList[i]->State = AVAILABLE;
   if (PutPorts (PortList) == ERR)
      {
       WriteLog ("FreePort: can't write port list to mark", Port, "in use", ""
       UnLock (PORTTABLE);
       free (PortList);
       return (ERR);
      }
   UnLock (PORTTABLE);
   free (PortList);
   return (GOOD);
}
int PutPorts (PortList)
portlist *PortList;
/*
   Writes the contenets of PortList over the current PORTTABLE.
   Assumes the PORTTABLE is LOCKED.
*/
{
```

```
    FILE *PortTableFd;
    register int i;
    if ((PortTableFd = fopen (PORTTABLE, "r+")) == NULL)
       {
        WriteLog ("PutPorts: can't open", PORTTABLE, "for update", "");
        return (ERR);
       }
    for (i=0; PortList[i] != NULL; i++)
        fprintf (PortTableFd, "%c%s %s %c\n", FIELDMARK, PortList[i]->Port,
                 PortList[i]->Site, PortList[i]->State);
    fclose (PortTableFd);
    return (GOOD);
}
portlist *GetPorts ()
/*
  Reads the PORTTABLE and builds an structure of port entries.
  Uses malloc() to get storage for the struct.
  Returns a pointer to the struct in PortList.
  Returns NULL if there is an error, pointer to list if all is well.
  End of array is marked by PortList[n] == NULL.
  Assumes PORTTABLE is LOCKED.
*/
{
#define MASK 0x00ff
   portlist *PortList;
   FILE     *PortTableFd;
   char     TempBuf[LINELEN+1];
   sitename Site;
   pathname Port;
   int      State=0;
   register int i, c, NumPorts;
   char     x;  /* Added - SLG */
   if ((PortTableFd = fopen (PORTTABLE, "r")) == NULL)
      {
       WriteLog ("GetPorts: Can't open", PORTTABLE, "", "");
       return (NULL);                              /* indicate failure */
      }
   /* count the ports so we can malloc() for PortList */
   NumPorts = 0;
   while ((c=getc(PortTableFd)) == ':')
      {
       if (ferror (PortTableFd))
          break;
       NumPorts++;
       SkipEOL (PortTableFd);
      }

   PortList = (portlist *) malloc ((NumPorts+1) * sizeof(portentry *));

   fclose (PortTableFd);
```

```
   PortTableFd = fopen (PORTTABLE, "r");
   /* -->  rewind (PortTableFd); */

   for (i=0; (((c=getc(PortTableFd)) != EOF) && (i < NumPorts)); i++)
      {
       PortList[i] = (portentry *) malloc (sizeof (portentry));
       fscanf (PortTableFd, "%s %s %c", Port, Site, &x);
       PortList[i]->Port = malloc (strlen(Port)+1);
       strcpy (PortList[i]->Port, Port);
       PortList[i]->Site = malloc (strlen(Site)+1);
       strcpy (PortList[i]->Site, Site);
       State = (int) x;
       /* State &= MASK;  -- Bill: this does not work on ONYX */
       PortList[i]->State = State;
       SkipEOL (PortTableFd);       /* goto next entry */
      }
   PortList[i] = NULL;  /* set end of list marker */
   fclose (PortTableFd);
   return (PortList); /* good return code */
}
int ValidPort (PortName)
char *PortName;
/*  Returns TRUE if PortName is defined in PORTTABLE, FALSE otherwise. */
{
   portlist *PortList;
   register int i, Found;
   if ((Lock (PORTTABLE)) == ERR)
      {
       WriteLog ("ValidPort: Can't lock", PORTTABLE, "", "");
       return (ERR);
      }
   if ((PortList = GetPorts ()) == NULL)
      {
       WriteLog ("ValidPort: can't read port list", "", "", "");
       UnLock (PORTTABLE);
       return (ERR);
      }
   for (i=0, Found=FALSE; ((!Found) && (PortList[i] != NULL)); i++)
      if (EQUALS(PortList[i]->Port, PortName))
         Found = TRUE;
   free (PortList);
   UnLock (PORTTABLE);
   return (Found);
}


EOF
```

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>PUTSIT.C

TIME: 1987/02/12 1648:29.2

```
#include "net.h"
extern int DebugLevel;
int  PutSite (SiteEntry)
site     *SiteEntry;
/* Looks up SiteEntry in SITETABLE and replaces its status information */
/* with the info in structure pointed to by SiteEntry.  If the site is not */
/* defined in the SITETABLE, an ERR will be indicated by the return    */
/* code. If the copy succeeds, the storage used for the site entry will be */
/* be released.                                                          */
{
   FILE     *SiteTableFd;
   register int c;
   register int n=0;
   long     offset;
   char     Name [SITENAMELEN];
   if ((Lock (SITETABLE)) == ERR)
      {
       WriteLog ("PutSite: can't lock", SITETABLE, "", "");
       exit (1);
      }
   if ((SiteTableFd = fopen (SITETABLE, "r+")) == NULL)
      {
       WriteLog ("PutSite: Can't open", SITETABLE, "", "");
       UnLock (SITETABLE);
       return (ERR);
      }
   /* scan through file until site found or EOF reached */
   getc (SiteTableFd);  /* skip first colon */
   do {
       fscanf (SiteTableFd, "%s", Name);  /* get a site  */
       if (EQUALS (SiteEntry->SiteName, Name))  /* it is the one we seek */
          {
           /* copy the site status information */
           /* offset = ftell (SiteTableFd); */
           fgetc (SiteTablFd);
           /* fseek (SiteTableFd, ++offset, 0); */ /* reset for output */
           fprintf (SiteTableFd, "%.1d %.1d %.91d", SiteEntry->Status,
                   SiteEntry->NumCalls, SiteEntry->TimeToCall);
           fclose (SiteTableFd);
           if (DebugLevel)
             WriteLog ("PutSite:", SiteEntry->SiteName, "has been modified",
                       "");
           UnLock (SITETABLE);
           free (SiteEntry);     /* release the storage */
           return (GOOD);  /* exit and indicate successful copy */
          }
       else
           /* advance to start of next site entry */
           do {
              SkipEOL (SiteTableFd);       /* skip to next line */
```

```
            c = getc (SiteTableFd);       /* read first character */
          } while ((c != FIELDMARK) && (c != EOF));
      } while (c != EOF);
   /* we wind up here if the site is not defined */
   WriteLog ("PutSite:", SiteEntry->SiteName, "not defined in",
             SITETABLE);
   fclose (SiteTableFd);
   UnLock (SITETABLE);
   free (SiteEntry);      /* release the storage */
   return (ERR);         /* exit and indicate error */
}


EOF
```

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>RDSITE.C

TIME: 1987/02/12 1659:14.0

```
#include "net.h"
site *RdSite (Fd)
FILE *Fd;
/*
       RdSite - read a site entry from the stream Fd into a site
                structure allocated here; return pointer to struct
*/
{
   site *Site;
   int  c;
   int i=0;
   pathname TempBuf;
   Site = (site *) malloc (sizeof (site));
   fscanf (Fd, "%s", TempBuf);
   Site->SiteName = malloc ( strlen(TempBuf)+1 );
   strcpy ( Site->SiteName, TempBuf );
   fscanf (Fd, "%d %d %ld %c %s", &(Site->Status), &(Site->NumCalls),
           &(Site->TimeToCall), &(Site->SysType), TempBuf);
   Site->Password = malloc ( strlen(TempBuf)+1 );
   strcpy ( Site->Password, TempBuf );
   SkipEOL (Fd); /* phone numbers begin on next line */
   c = getc (Fd);  /* read in the array of phone numbers */
   while ((c != NL) && (c != BLANK) && (c != FIELDMARK) &&
            (c != EOF) && (i <= MAXPHONENUMS))
      {
       ungetc (c, Fd);
       fscanf (Fd, "%s", TempBuf);
       Site->PhoneNum [i] = malloc (strlen (TempBuf)+1);
       strcpy (Site->PhoneNum [i], TempBuf);
       ++i;
       SkipEOL (Fd);      /* go to next line              */
       c = getc (Fd);     /* and get the first character */
      }
   Site->PhoneNum [i] = NULL;       /* mark end of list */
   return (Site);
}
```

EOF

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>READDI.C

TIME: 1987/02/12 1702:29.1

```
#include "net.h"
#define NFILES 64

/*char Entries [NFILES][FILENAMELEN];*/

/*
   ReadDir - Return directory structure expected by "dequeue()" containing
             entries in directory specified by parameter Dir.  This version
             is rewritten specifically for the Honeywell DPS/6.
*/

char **ReadDir (Dir)
char *Dir; /* Name of directory to read */

{
   char *star_name(); /* Honeywell directory function */

   char *DirEntry;  /* Pointer to directory entries */
   char *DirPtr;
   register int Num = 0;
   int i;
   char **Entries;  /* Returned vector containing directory entries */

   Entries = (char **) malloc (NFILES * (sizeof (char *)));

   if (Entries == (char **) NULL)
      {
       WriteLog ("ReadDir:", "cannot create storage for queue", "", "");
       return ((char **) NULL);
      }

   /* Call Honeywell-specific routine to get directory listing */

   DirEntry = star_name ("**", Dir);  /* Match all file names for now */
   DirPtr = DirEntry;

   if (DirEntry == (char *) NULL)
      {
       WriteLog ("ReadDir:", "cannot read directory", Dir, "");
       free (Entries);
       return ((char **) NULL);
      }

   while (*(DirEntry++) != '\0')
      {
       if (DirEntry[0] != 'Z')
          {
           Entries [Num] = malloc (strlen (DirEntry) + 1);
           strcpy (Entries [Num], DirEntry);
           Num++;
```

```
            }
         DirEntry += strlen (DirEntry) + 1;    /* Bump pointer past name */
        }

    free (DirPtr);

    if (Num > 0)
        {
/*       Sort (Entries, Num); */
         Entries [Num] = NULL;                 /* Set the last one to NULL */
         return (Entries);
        }
    else
        {
         free (Entries);
         return ((char **) NULL);
        }
}

Sort (Entries, Num)
filename Entries [];
int Num;
{
    /* register */ int Gap, i, j;
    filename Temp;
    for (Gap = Num / 2; Gap > 0; Gap /= 2)
        for (i = Gap; i < Num; i++)
            for (j = i - Gap; j >= 0; j -= Gap)
                {
                 if (strcmp (Entries [j], Entries [j+Gap]) <= 0)
                     break;
                 strcpy (Temp, Entries [j]);
                 strcpy (Entries [j], Entries [j+Gap]);
                 strcpy (Entries [j+Gap], Temp);
                }
}


EOF
```

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>READSI.C

TIME: 1987/02/12 1703:44.7

```
#include "net.h"
site *ReadSite (Fd)
FILE *Fd;
/*
       ReadSite - read a site entry from the stream Fd into a site
                  structure allocated here; return pointer to struct
*/
{
   site *Site;
   int  c;
   int i=0;
   pathname TempBuf;
   Site = (site *) malloc (sizeof (site));
   fscanf (Fd, "%s", TempBuf);
   Site->SiteName = malloc ( strlen(TempBuf)+1 );
   strcpy ( Site->SiteName, TempBuf );
   fscanf (Fd, "%d %d %ld %c %s", &(Site->Status), &(Site->NumCalls),
           &(Site->TimeToCall), &(Site->SysType), TempBuf);
   Site->Password = malloc ( strlen(TempBuf)+1 );
   strcpy ( Site->Password, TempBuf );
   SkipEOL (Fd); /* phone numbers begin on next line */
   c = getc (Fd);  /* read in the array of phone numbers */
   while ((c != NL) && (c != BLANK) && (c != FIELDMARK) &&
            (c != EOF) && (i <= MAXPHONENUMS))
      {
       ungetc (c, Fd);
       fscanf (Fd, "%s", TempBuf);
       Site->PhoneNum [i] = malloc (strlen (TempBuf)+1);
       strcpy (Site->PhoneNum [i], TempBuf);
       ++i;
       SkipEOL (Fd);      /* go to next line              */
       c = getc (Fd);     /* and get the first character */
      }
   Site->PhoneNum [i] = NULL;       /* mark end of list */
   return (Site);
}
```

EOF

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>READST.C

TIME: 1987/02/12 1705:20.2

```
#include "net.h"
#define MAXWAIT 10
char *ReadStr (Str)
char *Str;
/*
   ReadStr() will read a string from the modem into the space pointed to
   by String. The input will be terminated by a CR or NL character. The
   calling routine must insure the presence of sufficient space at String.
   ReadStr() will return a pointer to String if it succeeds or the value
   NULL if it fails.
*/
{
   register int n;
   register char *Ptr = NULL;
   char *GetStr();
   for (n=0; ((n < MAXWAIT) && (Ptr == NULL)); n++)
      Ptr = GetStr(Str);
   return (Ptr);
}
char *GetStr (String)
char *String;
{
   char *Ptr;
   Ptr = String-1;                                  /* point to start of storage
   do {
       Ptr++;
       if ((*Ptr = Receive()) == ERR)               /* get character from modem *
            return (NULL);                          /* bad return - timed out   *
      } while ((*Ptr != CRET) && (*Ptr != NL));
   *Ptr = '\0';     /* replace newline or cret with null */
   return (String);  /* good return - pointer to string */
}
```

EOF

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>REMOVE.C

TIME: 1987/02/12 1711:19.7

```
#include "net.h"
#define FATAL 40
int Remove (FileName, QueueName)
filename FileName;
pathname QueueName;
{
   pathname DirName;

   char *getdir ();

   getdir (DirName, 0);  /* Find out working directory name */

   chdir (QueueName);

   if (unlink (FileName) == ERR)
      {
       chdir ("ZTEMP");  /* Try the temporary directory */
       if (unlink (FileName) == ERR)
          {
           WriteLog ("Remove: FATAL: can't delete", FileName, "", "");
           exit (FATAL);  /* This is fatal.. cannot dequeue! */
          }
      }

   chdir (DirName);
}
```

EOF

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>SETHAY.C

TIME: 1987/02/12 1722:14.1

```
#include "net.h"
extern int ModemFd;
int SetHayes (PortName)
char *PortName;
/*
   Set the command modes and switches on the Smartmodem 1200.
   Test the modem to see if it will accept commands and give proper response.
   Return GOOD if all's well, ERR otherwise.
*/
{
   register int i,n;
    if (write (ModemFd, SETUP, strlen(SETUP)) == 0)
       {
        WriteLog ("SetHayes: can't write setup string to", PortName, "", "");
        return (ERR);
       }
    FlushModemInput (ModemFd);  /* Be sure buffer is empty */
    /* try to talk to the modem */
    for (i=0; i<10; i++)
       {
        write (ModemFd, ATTENTION, strlen(ATTENTION));
        switch (Receive ())
           {
            case ERR : WriteLog ("SetHayes: no response to ATTENTION signal",
                                 "", "", "");
                       break;
            case OK : FlushModemInput (ModemFd);
                      return (GOOD);
            default : WriteLog ("SetHayes: modem answered ATTENTION with",
                                "invalid code", "", "");
                      break;
           }
       }
    FlushModemInput (ModemFd);  /* Be sure buffer is empty */
    WriteLog ("SetHayes: Cannot establish rapport with modem on",
              PortName, "", "");
    return (ERR);
}
```

EOF

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>SETPOR.C

TIME: 1987/02/12 1723:00.0

```
#include "net.h"
extern int ModemFd;
int SetPort (PortName)

char *PortName;

/*
Open the named port for use by IOControl.  Cannot set line parameters, so
don't do that here.  (Honeywell version; other versions set parameters.)
Sets global variable ModemFd to new file pointer, or ERR if modem cannot
be opened.
*/

{
   if ((ModemFd = open (PortName, O_RDWR)) == ERR)
      {
       WriteLog ("SetPort: Can't open", PortName, "", "");
       return (ERR);
      }
    return (ModemFd);
}
```

EOF

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>STATE.C

TIME: 1987/02/12 1723:23.5

```
#include "net.h"
   /*
   This routine will check to see that the site entry pointed to by Site
   is not declared down. If it is, the routine will look at the
   TimeToCall field and see if it is time to try again.
   */
int State (Site) /* returns the value of Site->Status (UP, RETRY, or DOWN) */
site *Site;
{
   if (Site->Status != UP)
      {
       if (NOW >= Site->TimeToCall)
            {
             Site->Status = UP;
             Site->TimeToCall = 0;
            }
      }
   return (Site->Status);
}
```

EOF

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>STRIPM.C

TIME: 1987/02/12 1723:49.4

```
#include "net.h"


char *StripMe (Path, FirstSite)

char *Path, *FirstSite;

/* Removes the first site name from Path and returns a pointer to the */
/* rest of the path. If there is only one site name left in Path, a   */
/* NULL is returned. Path is not altered, and FirstSite points to the */
/* site name that has been stripped off the front of the path.        */

{
   register int i = 0;

   while ((Path [i] != SEPCHAR) && (Path [i] != '\0'))
      {
       FirstSite [i] = Path [i];
       i++;
      }

   FirstSite [i] = '\0';

   return ((Path [i] == '\0') ? '\0' : Path + i + 1);
}
```

EOF

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>UNLOCK.C

TIME: 1987/02/12 1724:55.0

```
#include "net.h"
int UnLock (FileName)
/* returns ERR if the file could not be unlocked, GOOD otherwise */
char *FileName;
{
   pathname LockFile;
   int Result=0;
   sprintf (LockFile, "%s.LCK", FileName);
   Result = unlink (LockFile);
   if (Result == ERR)
      WriteLog ("UnLock: can't unlock", LockFile, "", "");
   return (Result);
}
```

EOF

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>VALIDP.C

TIME: 1987/02/12 1725:29.1

```
#include "net.h"

int ValidPath (Site, Path)

char *Site;
char *Path;

{
   FILE *PathTableFd;
   char SomeSite [128];
   register int c;

   if ((PathTableFd = fopen (PATHTABLE, "r")) == NULL)
      {
       fprintf (stderr, "ValidPath: Can't open %s\n", PATHTABLE);
       exit (1);
      }

   getc (PathTableFd);                                  /* skip first colon */

   do {
       fscanf (PathTableFd, "%s", SomeSite);            /* get a site */
fprintf(stderr, "Path table entry: %s (", SomeSite);
       if (EQUALS (SomeSite, Site))
          {
fprintf(stderr, "matched)\n");
           getc (PathTableFd);                          /* read NL   */
           fscanf (PathTableFd, "%s", Path);
           fclose (PathTableFd);
           return (TRUE);
          }
       else
          {
fprintf(stderr, "not matched)\n");
           c = getc (PathTableFd);
             while ((c != FIELDMARK) && (c != EOF))
               c = getc (PathTableFd);
          }

      } while (c != EOF);

   fclose (PathTableFd);
   return (FALSE);
}
```

EOF

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>VALIDS.C

TIME: 1987/02/12 1726:10.2

```
#include "net.h"
site *ValidSite (Site)
char *Site;
/* Validates the existence of Site in SITETABLE and returns as  */
/* its value a pointer to a site table entry in a struct defined    */
/* by "site" in net.h                                               */
/* If the site is not found in the table, it returns NULL           */
{
   FILE     *SiteTableFd;
   site     *SiteEntry;
   register int c, n=0;
   long     Start;
   pathname TempBuf;
   register int i;
   int      NumRecs;

   if ((Lock (SITETABLE)) == ERR)
      {
       WriteLog ("ValidSite: Can't lock", SITETABLE, "", "");
       exit (1);
      }
   if ((SiteTableFd = fopen (SITETABLE, "r")) == NULL)
      {
       WriteLog ("ValidSite: Can't open", SITETABLE, "", "");
       UnLock (SITETABLE);
       return (NULL);                          /* indicate failure */
      }
   /* skip down to beginning of first site entry */

   NumRecs = 0;

   while ((c = getc (SiteTableFd)) != FIELDMARK) SkipEOL (SiteTableFd);
   do {
       NumRecs++;  /* Remember how many 'records' we have skipped */
/*     Start = ftell (SiteTableFd); /* save pointer to start of entry */
       fscanf (SiteTableFd, "%s", TempBuf);  /* get a site name */
       if (EQUALS (TempBuf, Site))  /* it is the one we seek */
          {
           fclose (SiteTableFd);
           SiteTableFd = fopen (SITETABLE, "r");

           for (i = 0; i < (NumRecs); i++)
               while ((c = getc(SiteTableFd)) != FIELDMARK)
                     ;

           SiteEntry = RdSite (SiteTableFd); /* read in the site entry */
           fclose (SiteTableFd);
           UnLock (SITETABLE);
           return (SiteEntry);  /* return pointer to the site structure */
          }
```

```
        else
            do {
               c = getc (SiteTableFd);       /* read first character */
            } while ((c != FIELDMARK) && (c != EOF));
        } while (c != EOF);
    fclose (SiteTableFd);
    UnLock (SITETABLE);
    return (NULL);  /* indicate site not found */
}


EOF
```

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>WRITEL.C

TIME: 1987/02/12 1732:15.6

```
#include "net.h"
extern pathname LogFile;
int WriteLog (P1, P2, P3, P4)
char *P1, *P2, *P3, *P4;
{
   FILE *LogFd;
   char Date [26];
   register int i=0;
   DateTime (Date);
   if ((LogFd = fopen (LogFile, "a")) == NULL)
           {
            fprintf (stderr, "WriteLog: Can't open %s.\n", LogFile);
            fprintf (stderr, "%s %s %s %s %s\n", Date, P1, P2, P3, P4);
            return (ERR);
         }
/* setbuf (LogFd, (char *) NULL); */
   fprintf (LogFd, "%s %s %s %s %s\n", Date, P1, P2, P3, P4);
   fclose (LogFd);
   return (GOOD);
}
```

EOF

CALLER

This section contains the functions used only by the Caller program (CALLER).

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>CALLER.C

TIME: 1987/02/12 1554:22.7

```
#include "sysdef.h"
#include "net.h"
#include <signal.h>
int ModemFd; /* OpenModem() will return a file descriptor for the modem   */
site *Site;  /* allocated by ValidSite(), released by free() or PutSite() */
pathname LogFile;  /* global LogFile for use by WriteLog()                 */
sitename SysName;  /* name of the system to be called                      */
pathname PortName; /* name of the port to use                              */
pathname IntFileName; /* Name of interrupt file to remove                  */
int IntFlag;  /* Flag indicating if interrupt occurred                     */
int DebugLevel = 0;   /* Runtime debug level (0 = normal)                 */
extern int errno;
#ifdef XENIX
   FILE *File;  /* for resetting O_NDELAY under XENIX */
#endif
main (argc, argv)
int argc;
char ** argv;
/*
   The Caller program will take a system name and a port name as input
   parameters.  It will open the named port for dialout use and check for
   the presence of a modem.  It will get the connection information on the
   desired system and try to establish a connection with that system.  If
   that succeeds, it will call IOControl to send the files located in the
   directory and receive any datafiles from the remote system. Received file
   will be given to the Message Processor for disposition.
   Errors will be logged, and the program will terminate with either a good
   return code (0) or a bad return code (-1).
   If a remote site is called unsuccessfully, a record is kept in the site
   table.  When the number of unsuccessful attempts crosses a threshold
   value, the site is declared down and all traffic for the site is rerouted
   for a period of time.
*/
{
   pathname IOControl;
   int Status;
   register int pid;
   register int Condition;
   register int Connected=FALSE;
   int i;

   int quit();

   signal (SIGQUIT, quit);
   IntFlag = FALSE; /* Do not-so-run-time initialization */
   /* get the arguments and prepare for action */
   if (argc != 3)
      if ((argc == 4) && (argv [3][0] == '-') && (argv [3][1] == 'D'))
         {
          argc--;      /* Remove last argument */
```

```
            sscanf (argv[3]+2, "%d", &DebugLevel);  /* Get debug level (-d#) */
          }
       else
          {
           fprintf (stderr, "usage: %s system port\n", argv[0]);
           exit (ERR);
          }
    strcpy (SysName, argv[1]);
    strcpy (PortName, argv[2]);

    for (i = 0; i < strlen (SysName); i++)
        SysName[i] = tolower (SysName[i]);

    sprintf (LogFile, "log/%s.log", SysName);
    if (DebugLevel)
       WriteLog ("Caller:", "system debug level is", argv[3]+2, "");
    if (!ValidPort(PortName))
       {
        WriteLog ("Caller: port", PortName, "not found in", PORTTABLE);
        exit (ERR);
       }
    /* open the modem port and make sure the modem is awake */
    if ((ModemFd = OpenModem(PortName)) == ERR)
       {
        WriteLog ("Caller: can't open", PortName, "to call", SysName);
        FreePort (PortName);
        exit (ERR);
       }

    /* put the modem in command mode and set the switches */
/*
    if (SetHayes (PortName) == ERR)
       {
        WriteLog ("Caller: can't configure modem on", PortName, "", "");
        FreePort (PortName);
        exit (ERR);
       }
*/

    FlushModemInput (ModemFd);

        if ((Site = ValidSite (SysName)) != NULL)  /* Site is valid */
         {
          if ( (Condition = State (Site)) == UP )
             {
               if (DebugLevel)
                  WriteLog ("Caller: invoked for", SysName, "using", PortName);
              Connected = Dial (Site->PhoneNum);

              if (Connected == TRUE)
```

```
{
close (ModemFd);
if (DebugLevel)
        WriteLog ("Caller: Connected to", SysName,
                  "-", "about to log in");
     if (Login (SysName, Site->Password, Site->SysType,
                PortName) == ERR)
         quit (ERR);
     if ((pid = fork ()) == ERR)
        {
         WriteLog ("Caller: cannot fork - Goodbye!", "", "", "");
         quit (errno);
        }
     if (pid == 0) /* I am the child */
        {
         /* IOCONTROL does the transmitting and receiving */
         sprintf (IOControl, "%s>%s", BIN, IOCONTROL);
         if (DebugLevel)
            execl (IOControl, IOControl, SysName, PortName,
                   argv[3], (char *) 0); /* Pass debug info */
         else
            execl (IOControl, IOControl, SysName, PortName,
                   (char *) 0);
         WriteLog ("Caller: cannot exec", IOControl, "- Goodbye!"
                   , "");
         exit (ERR);
        }
     wait (&Status);            /* wait for IOCONTROL to complete
     switch ((Status&0xff00) >> 8)
        {
         case GOOD :
            if (DebugLevel)
               WriteLog ("Caller: Conversation with", SysName,
                         "complete.", "");
            break;
         case INTERRUPTED :
            if (DebugLevel)
               WriteLog ("Caller: Interrupted during",
                         "conversation with", SysName, "");
        IntFlag = TRUE;  /* Set interrupt flag for later */
            /* Construct interrupt file name to remove */
        sprintf (IntFileName, "%s/%s", SysName, INTFILE);
            break;
         case ERR :
         default  :
            WriteLog ("Caller:", IOCONTROL, "returned error", "")
       if (CheckDown (Site))  /* record unsuccessful call */
           GiveToMP (SysName); /* may need to forward mail */
            quit (ERR);
        }
```

```
                    /* reset site table entry to indicate successful contact */
                Site->Status = UP;       /* declare site up     */
                Site->TimeToCall = NOW; /* reset time to call */
                Site->NumCalls = 0;      /* reset retry count  */
                PutSite (Site);          /* save entry & release storage    */
                quit (GOOD);          /* all is well - hang up phone & exit */
               }
                else          /* Connected=FALSE - remote system didn't answer */
               {
                WriteLog ("Caller: Can't connect to", SysName, "", "");
                if (Connected == FALSE)    /* remote system didn't answer  */
                    {
                   if (CheckDown (Site))   /* if site is declared down     */
                       GiveToMP (SysName);  /* give messages to MSGPROC     */
                       }
                     else         /* Connected=ERR: modem didn't answer commands *
                       {
                        WriteLog ("Caller: we have a problem",
                                  "with the modem at", PortName, "");
                        free (Site);             /* release storage              *
                       }
               }
                }
          else    /* Condition != UP  ( may be DOWN or DELAY) */
               {
                if (Condition == DOWN)  /* may need to forward the messages */
               GiveToMP (SysName);
                free (Site);      /* release storage */
               }
         }
        else  /* ValidSite returned NULL */
         {
          WriteLog ("Caller: No entry for", SysName, "defined in", SITETABLE);
           /* take some action or notify operator of this situation */
         }
   if (DebugLevel)
      WriteLog ("Caller: unlocking", SysName, "", "");
   if (UnLock (SysName) == ERR)
      WriteLog ("Caller: could not unlock", SysName, "queue", "");
   if (DebugLevel)
      WriteLog ("Caller: freeing", PortName, "", "");
   if (FreePort (PortName) == ERR)
      WriteLog ("Caller: could not free", PortName, "after use", "");
   exit (ERR); /* arrive here only if some error occurred above */
}
int quit (retcode)  /* hang up the phone and exit */
int retcode;
{
   if (! HangUp(Site->SysType))
      WriteLog ("Caller: could not hang up the phone", "", "", "");
```

```
/* if (UnLock (SysName) == ERR)
      WriteLog ("Caller: could not unlock", SysName, "queue", ""); */
   if (FreePort (PortName) == ERR)
      WriteLog ("Caller: could not free", PortName, "after use", "");
   if (IntFlag)   /* This means that IOControl returned after interrupt */
   if (unlink (IntFileName) == ERR)  /* We must then acknowledge */
        WriteLog ("Caller:", "could not remove", IntFileName, "");
  exit (retcode);
}


EOF
```

```
Main:  ~ZSYS72>UDD>GOLDBERG>UNIX>CHECKD.C

TIME:  1987/02/12 1559:15.6


#include "net.h"
int CheckDown (RSite)
site *RSite;
{
/* Returns TRUE if site is declared DOWN, FALSE otherwise.                */
/* Sets callback time appropriately and updates number of failed calls. */
/* PutSite sets status and frees site entry storage.                      */
    RSite->NumCalls++;  /* increment the retry count */
    if (RSite->NumCalls >= MAXCALLS)
       {
          RSite->Status = DOWN;  /* declare the site down */
          RSite->NumCalls = MAXCALLS-1;
          RSite->TimeToCall = NOW + DOWNDELAY;
          PutSite (RSite);  /* save entry & release storage */
          WriteLog ("CheckDown:", RSite->SiteName, "is down.", "");
          return (TRUE);
       }
    else   /* RSite->NumCalls < MAXCALLS means try again later */
       {
          RSite->Status = RETRY;  /* declare the site delayed */
          RSite->TimeToCall = NOW + RETRYDELAY;
          WriteLog ("CheckDown:",RSite->SiteName,"temporarily down", "");
          PutSite (RSite);  /* save entry & release storage */
          return (FALSE);
       }
}


EOF
```

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>DIAL.C

TIME: 1987/02/12 1602:57.8

```
#include "net.h"
extern int ModemFd;  /* Modem file descriptor opened elsewhere */

int Dial (TelNums)
char *TelNums[20];  /* Telephone numbers */

{
   char Ctr1N = '\016';  /* InfoMate control character */
   char DialCmd [80];  /* Space for dial command */
   char ch[80];
   int  RetCode;
   int  Result;

/* printf ("about to set modem parameters\n"); */
   sprintf (DialCmd, "   xy\r\n");
   write (ModemFd, DialCmd, strlen (DialCmd));

   sprintf (DialCmd, "%cP 00\r\n", Ctr1N);  /* was P 24 */
   write (ModemFd, DialCmd, strlen (DialCmd));

   sleep (1);
   sprintf (DialCmd, "%cD '%s'\r\n", Ctr1N, TelNums[0]);
   write (ModemFd, DialCmd, strlen (DialCmd));

   Result = read (ModemFd, ch, 79);                    /* Read response */

/*   while (((Result > 1) && (ch[0] == Ctr1N) && (ch[1] != 'A'))
          || (Result <= 1))
*/
     while (ch[1] != 'A')
      {
/*     if ((ch[1] != 'T') && (ch[1] != '\n') && (ch[1] != '\r'))
          printf("got %x from modem (unexpected)\n", (int) ch[1]); */
       read (ModemFd, ch, 79);
      }

/* printf (" Connected to remote system.\n");   */

   return (TRUE);
}
```

EOF

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>GETPRO.C

TIME: 1987/02/12 1619:26.5

```
#include "net.h"
#include "iocontrol.e"
/*
   GetPrompt - read a string from the modem line and return a pointer to it.
               End of string will be indicated by a timeout on read operation.
               There may be embedded newlines or returns, but these will not
               mark the end of the string. If no characters are read, return
               a null string.
*/
char *GetPrompt ()
{
   char *ptr;
   char TempBuf[80];
   int i=0;
   int c;
fprintf(stderr, "About to try to read login prompt.\n");
fflush (stderr);
   while (i < 80)                       /* read until timeout or overflow */
       if ((c = Receive()) == ERR)      /* ERR means timeout, so exit */
           break;
       else
          TempBuf[i++] = c;  /* buffer the character, increment counter */
   TempBuf[i] = '\0';                   /* mark end of string */
fprintf(stderr, "Got a string from the modem: it was '%s'\n", TempBuf);
fflush (stderr);
   ptr = malloc (strlen(TempBuf)+1);  /* allocate storage */
   strcpy (ptr, TempBuf);             /* copy the string */
   if (DebugLevel1)
      {
       if (strlen(TempBuf) > 0)           /* if string is not empty */
          WriteLog ("GetPrompt:", "got", ptr, "from modem");
       else
          WriteLog ("GetPrompt:", "timed out on Receive", "", "");
      }
   return (ptr);                        /* return a pointer to the string */
}
```

EOF

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>GIVETO.C

TIME: 1987/02/12 1620:13.4

```
#include "net.h"
int GiveToMP (SysName)
char *SysName;
{
/* Move all messages that are not courtesy copies over to the msgproc queue. *
/* We have to look out for case where nothing remains in the queue except CC *
/* messages. DeQueue() will keep giving us the names of the CC files forever *
/* if we don't detect this state and exit the loop. We do this by saving the *
/* name of the first CC encountered and checking it against later CC files.  *
/* When it is see for the second time, we will have sent all the messages to *
/* be rerouted except for the remaining CC files, which we must leave in the *
/* queue.                                                                    *
char *MessageFile;
pathname NewPath;
pathname OldPath;
pathname SaveName;
register int Fd;
register int Count = 0;
register int Done = FALSE;
char     Number[10];
/* if ((Fd = OpenDir(SysName)) == ERR)
        {
         WriteLog ("GiveToMP: can't open directory", SysName, "", "");
         return (ERR);
        }
*/
   strcpy (SaveName, "");  /* set strlen(SaveName) to 0 */
   while ( ((MessageFile = DeQueue(SysName)) != NULL) && (!Done) )
      if (MessageFile[0] != CCTYPE)  /* mark non-CC files for forwarding */
         {
          sprintf (OldPath, "%s/%s", SysName, MessageFile);
          sprintf (NewPath, "%s/%c%s", MSGPROCQ, REROUTETYPE, MessageFile+1);
          FRename ( OldPath, NewPath );
          WriteLog ("GiveToMP: moved", OldPath, "to", NewPath);
          ++Count;
         }
      else
         {
          if (strlen(SaveName) == 0) /* remember name of first CC message */
             strcpy (SaveName, MessageFile);
          else /* see if we are back to the first CC message */
             Done = EQUALS (SaveName, MessageFile);
         }
/* CloseDir (Fd);  */
   if (Count > 0)
      {
       sprintf (Number, "%d", Count);
       WriteLog ("GiveToMP: forwarded", Number, "messages", "");
      }
   return (GOOD);
```

```
}

EOF
```

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>HANGUP.C

TIME: 1987/02/12 1621:49.7

```
#include "net.h"
extern int ModemFd;
extern int DebugLevel;
int HangUp (SysType)
char SysType;
/*
   Sends a hangup command to the Cermetek modem
*/
{
   char   HangUpCmd[5];

   FlushModemInput (ModemFd);
   if (SysType == GCOS)
      {
       sleep (5);                /* Ensure that the receiver has ended */
       write (ModemFd, "BYE\r", strlen ("BYE\r"));  /* Transmit logout rqst */
       sleep (4);
      }
   FlushModemInput (ModemFd);

   sleep (1);

   sprintf (HangUpCmd, "%cE\r", '\016');  /* Ctrl-N, E hangs up */
   write (ModemFd, HangUpCmd, strlen (HangUpCmd));
   write (ModemFd, "\n", 1);

   if (DebugLevel)
      WriteLog ("HangUp: hung up the phone ok", "", "", "");
   return (TRUE);  /* hung up the phone OK */
}
```

EOF

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>LOGIN.C

TIME: 1987/02/12 1628:23.6

```
#include "net.h"
#include "iocontrol.h"
extern int ModemFd;
extern int DebugLevel;
int Login (Name, Password, SysType, PortName)
char *Name;
char *Password;
char  SysType;
char *PortName;
{
char System[2];
char Command[40];
char *PromptStr;   /* prompt string from remote site: allocated by GetPrompt *
int Count = 0;
int LoggedIn = FALSE;
char Garbage [101];

if (DebugLevel)
   WriteLog ("Login: Connected, about to log in as", Name, "", "");
switch (SysType)
  {
   case GCOS :       /* must load IOControl by hand */
      {
       if (DebugLevel)
          WriteLog ("Login: system is CGOS", "", "", "");
       write (ModemFd, CR, 1);            /* refresh login prompt */
       sleep (6);
       FlushModemInput (ModemFd);
       sprintf (Command, "%s %s\r", GCOSLOGIN, Name);
       write (ModemFd, Command, strlen (Command));
       sleep (4);                          /* Wait for Password prompt */
       sprintf (Command, "%s\r", Password);
       write (ModemFd, Command, strlen (Command));
       sleep (10);                        /* Wait to log all the way in */
       return (GOOD);
      }
   case UNIX :       /* IOControl is the default shell */
      {
       if (DebugLevel)
          WriteLog ("Login: system is UNIX", "", "", "");
       do {
          FlushModemInput(ModemFd);
          ModemFd = OpenModem (PortName);
          write (ModemFd, CR, 1);            /* refresh login prompt */
          write (ModemFd, "\n", 1);          /* Flush output buffer */
          close (ModemFd);
/*
        PromptStr = GetPrompt();   /* get login prompt */
/*
        if (EndsWith(UNIXLOGIN, PromptStr))
```

```
            {
*/
             sleep (2);  /* Simply wait 2 seconds on Honeywell */
             sprintf (Command, "%s.h\r\n", Name); /* send username */
             ModemFd = OpenModem (PortName);
             write (ModemFd, Command, strlen (Command));
             close (ModemFd);
             if (DebugLevel)
                WriteLog ("Login:", "sent login ID", Name, "to modem");
/*
          PromptStr = GetPrompt();  /* get password prompt */
/*
             if (EndsWith(UNIXPASSWORD, PromptStr))
                {
*/
                 sleep (1);  /* Simply wait 1 second for Password */
                 sprintf (Command, "%s\r\n", Password);
                 ModemFd = OpenModem (PortName);
                 write (ModemFd, Command, strlen (Command));
                 close (ModemFd);
                 if (DebugLevel)
                   WriteLog ("Login:", "sent password", Password, "to modem")

                 LoggedIn = TRUE;  /* we have sent our login sequence */
/*
                }
             else
                {
                 if (DebugLevel)
                    WriteLog("Login:", "didn't get", UNIXPASSWORD, "prompt");
                }

             free (PromptStr);

            }
         else
            {
             if (DebugLevel)
                WriteLog("Login:", "didn't get", UNIXLOGIN, "prompt");
            }
         free (PromptStr); /* release storage */
       } while ((!LoggedIn) && (Count++ < MAXRETRY));
       FlushModemInput(ModemFd);
       if (!LoggedIn)       /* could not log in to remote system */
          {
           WriteLog ("Login:", "could not log in to remote system", "", "");
           return (ERR);
          }
       return (GOOD);
      }
```

```
    default : /* undefined system type */
        {
         sprintf (System, "%c", SysType);
         WriteLog ("Login: undefined system type -", System, "", "");
         return (ERR);
        }
    }
}


EOF
```

IOCONTROL

This section contains the functions used only by the IOControl program (IOCONTROL).

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>IOCONT.C

TIME: 1987/02/12 1622:41.8

```
#include "net.h"
#include "iocontrol.h"

/*
   IOControl - perform both SEND and RECEIVE functions for bottom layer of
               network system.  Use character-oriented protocol with check-
               sums and stop-and-wait retransmission scheme. (With a
               single-packet transmission window.)  If invoked
               by local system, assume role of SENDER at start,
               otherwise become a RECEIVER for the first transactions.
*/
/* Global variables for I/O Control system */
int ModemFd;          /* Modem file descriptor */
int SeqNo = 0;        /* Current packet-sequence number */
pathname LogFile;     /* Global LogFile for IOControl routines */
pathname QueueName; /* Name of queue from which we're reading */
int DebugLevel = 0; /* Runtime debug level (0 = normal) */
/* */
main (argc, argv)
int   argc;
char *argv[];
{
   int mode;     /* Current operating mode (0,1=init; 2=send; 3=receive) */
   int memory = FALSE;    /* Used to determine hangup status */
   pathname NextPath; /* Complete path of any given file */
   char *NextName; /* File name for each file read from queue */
   char *TermName; /* Terminal name returned by ttyname() -- Honeywell */
   sitename RemoteName; /* Name of calling remote site */
   int Result;
   int QueueFd;
   extern int CurFilePtr;

   char *ttyname(); /* Need to know terminal's name on Honeywell */

   /* Parse arguments to see if this is invoked on local system */
   sprintf (LogFile, "log/%s.log", QMS);
   if ((argc != 3) && (argc != 1))
      if ((argc == 4) && (argv [3][0] == '-') && (argv [3][1] == 'D'))
         {
          argc--;      /* Remove last argument */
          sscanf (argv[3]+2, "%d", &DebugLevel);  /* Get debug level (-d#) */
          WriteLog ("IOControl:", "system debug level is", argv[3]+2, "");
         }
      else
         {
          fprintf (stderr, "usage: %s [ sysname portname ]\n", argv[0]);
          exit (FATAL);
         }
   if (argc == 3)      /* master mode */
      {
```

```
    mode = MASTERINIT;  /* Enter INITIALIZATION as MASTER */
    strcpy (QueueName, argv[1]);  /* Place queue name in local var */
    strcpy (RemoteName, argv[1]); /* Remember name of remote */
    ModemFd = SetPort (argv [2]);  /* Open modem as instructed */
    if (ModemFd == ERR)  /* Could not do it */
       {
        WriteLog ("IOControl:", "FAILED opening", argv[2], "(modem)");
        exit (FATAL);  /* The modem should have been available; abort */
       }
   }
else     /* slave mode */
   {
    TermName = ttyname (0);
    close (0);
    close (1);
    ModemFd = open (TermName, O_RDWR);  /* Treat terminal as remote */
    mode = SLAVEINIT;            /* Enter INITIALIZATION as SLAVE */
   }

CurFilePtr = 0;  /* Initialize index for DEQUEUE routine */

/* Main loop - perform tasks depending on current mode */
while (mode != HANGUP)
   {
    switch (mode) {
       case MASTERINIT :
                sprintf (LogFile, "log/%s.log", RemoteName);
                if (DebugLevel)
                   WriteLog ("IOControl: assuming MASTER mode.", "", "", ""
                WaitEnq ();             /* Wait for enquire, send ACK */
                if (DebugLevel)
                   WriteLog ("IOControl:", "enquire received and",
                             "acknowledged", "");
                SendName ();            /* Send system name, wait for ACK */
                WriteLog ("IOControl:", "connection established (",
                          RemoteName, ")");
                /* QueueFd = OpenDir (QueueName); */
                mode = SENDMODE;        /* Enter send mode */
                break;
       case SLAVEINIT :
                SendEnq ();              /* Send ENQ signal, wait for ACK */
                WaitName (RemoteName); /* Wait for remote id, send ACK */
                sprintf (LogFile, "log/%s.log", RemoteName);
                if (DebugLevel)
                   WriteLog ("IOControl: assuming SLAVE mode.", "", "", "")
                WriteLog ("IOControl:", "connection established (",
                          RemoteName, ")");
                strcpy (QueueName, RemoteName);
                mode = RECEIVEMODE;    /* Enter receive mode */
                break;
```

```
        case SENDMODE : /* Send mode */
                /* Check to see if line needed by higher priority task */
                /* ...or if we're simply finished.  Hang up if so.     */
                if ((memory) || Preemption (QueueName))
                    {
                     mode = HANGUP;
                     /* CloseDir (QueueFd); */
                     SendByte (EOT);
                     break;
                    }
                NextName = DeQueue (QueueName); /* Read sysX queue */
                if (NextName != NULL)
                    {
                     sprintf (NextPath, "%s/%s", QueueName, NextName);
                     if (SendFile (NextPath, NextName) != ERR)  /* Send it
! */
                         {
                          Archive (NextName, QueueName);  /* Copy file */
                          Remove (NextName, QueueName);   /* Delete file */
                         }
                     else  /* Something went wrong during transmission */
                         {
                          WriteLog ("IOControl:", QueueName, "FAILED", "");
                          exit (LOSTCONTACT);  /* Hang up and all that */
                         }
                    }
                else
                    {
                     SendByte (EOT);  /* Send an end-of-transmission */
                     memory = TRUE;         /* Remember that we sent EOT */
                     /* CloseDir (QueueFd); */
                     mode = RECEIVEMODE;  /* Enter receive mode */
                    }
                break;
        case RECEIVEMODE : /* Receive mode */
                Result = GetFile (); /* Get next receive file */
                if (Result == EOT)
                     {
                      /* QueueFd = OpenDir (QueueName); */
                      mode = SENDMODE;
                     }
                else if ((Result == INTERRUPTED) || (Result == ABORTED))
                     {
                      WriteLog ("IOControl:", RemoteName, "aborted", "");
                      WriteLog ("IOControl:", "relinquishing line", "", "");
                      mode = HANGUP;  /* Force a graceful hangup */
                     }
                else if (Result == ERR)
                     {
                      WriteLog ("IOControl:", RemoteName, "FAILED", "");
```

```
                        exit (LOSTCONTACT);
                      }
                  else
                      memory = FALSE;       /* Forget about hanging up */
                  break;
          default : /* Illegal (undefined) mode */
                  WriteLog ("IOControl:", "internal error:", "bad mode", "");
                  exit (INTERNALERROR); /* Ungraceful exit to O/S */
          }
      }  /* End of WHILE */
   WriteLog ("IOControl:", "conversation COMPLETE (", RemoteName, ")");
   if (Preemption (QueueName))  /* Give different return code if preempted */
      exit (INTERRUPTED);
   exit (GOOD);  /* Exit with 'normal completion' code, 0 */
}

EOF
```

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>ARCHIV.C

TIME: 1987/02/12 1553:44.3

```
#include "net.h"
/*      Archive - copy file in given queue into archive directory  */
int Archive (FileName, Queue)
char *FileName;
char *Queue;
{
   int c;
   FILE *OldFd;
   FILE *NewFd;
   pathname Path;
   sprintf (Path, "%s/%s", Queue, FileName);     /* name of input file  */
   if ((OldFd = fopen (Path, "r")) == NULL)      /* read from this file */
      {
       WriteLog ("Archive: can't open", Path, "-", "file not archived");
       return (FALSE);
      }
   sprintf (Path, "%s/%s", ARCHIVEQ, FileName); /* name of archive file  */
   if ((NewFd = fopen (Path, "w")) == NULL)      /* write to archive file */
      {
       WriteLog ("Archive: can't open", Path, "-", "file not archived");
       return (FALSE);
      }
   while ((c = getc(OldFd)) != EOF)              /* copy the file */
      putc (c, NewFd);
   fclose (OldFd);                               /* close the files */
   fclose (NewFd);
   return (TRUE);
}
```

EOF

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>CHECKS.C

TIME: 1987/02/12 1559:55.2

```
#include <stdio.h>
/*
  CheckSum - compute a 16-bit checksum of 'length' bytes starting at 'data'.
*/
int CheckSum (data, max)
unsigned char *data;
int  max;
{
   int i;
   long Result; /* Storage for temporary result */

   Result = 0;

   for (i = 0; i <= max; i++)
        if (data[i] != (unsigned char) '\n')
           Result += (int) data[i];

   Result &= 0xffff; /* Restrict to 16-bit quantity */

   return ((int) Result);
}
```

EOF

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>CREATE.C

TIME: 1987/02/12 1600:44.3

```
#include "net.h"
/*
   CreateFile - build a file name in the directory given by Queue, putting
                in in the ZTEMP subdirectory to make it invisible, using
                the name of the file in FileName. Open the file and
                return a file descriptor.
*/
FILE *CreateFile (FileName, Queue)
char *FileName;      /* pointer to new file name */
char *Queue;         /* directory in which to put the file */
{
   pathname DirName;      /* storage for current working directory name */
   pathname PathName;     /* pointer to modified filename (invisible) */
   FILE *Result;          /* resultant file pointer from fopen (3) */

   char *getdir();

   getdir (DirName, 0);  /* Save working directory name */

   sprintf (PathName, "%s>%s", Queue, "ZTEMP"); /* Put files in temp queue */

   if (chdir (PathName) != 0)
      {
       WriteLog ("CreateFile:", "cannot change directory to", Queue, "");
       return ((FILE *) NULL);
      }

   Result = fopen (FileName, "w");          /* Create the file */

   chdir (DirName);  /* Change back to parent directory before returning */

   return (Result);  /* Return file pointer or NULL if error */
}
```

EOF

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>GETBLO.C

TIME: 1987/02/12 1612:22.1

```
#include "net.h"
#include "iocontrol.h"
#include "iocontrol.e"

/*
   GetBlock - read the next block from remote system, taking
              care of retransmission, etc.  Either return a pointer
              to good packet data along with a length, or return an error
              code (NULL) with the length set to indicate reason
              for leaving (EOT = end of transmission received,
              ERR = error). Return parameters "length" and "endflag"
              indicate the length of data (in bytes) and the
              value of the end-of-text flag, respectively.
              Recognize a packet containing the CAN control code
              as indicating remote system shutdown.
*/

#define ENDFLAG (PackLen - ETXOFFSET)  /* End-of-text pointer is here */

#define EatModemInput {char tmp[100]; read(ModemFd,tmp,99);}

unsigned char *GetBlock (length, endflag)

int *length;  /* Return parameter for length of block */
int *endflag; /* Return parameter for end-of-text vs. end-text-block */

{
   int TempValue;      /* Result to return to caller */
   unsigned char *TempBuf;      /* Pointer to buffer returned from 'getpacket' */
   unsigned char *BlockBuffer; /* Buffer to return to caller with good data */
   int PackLen;        /* Length of incoming packet */
   int HisCheckSum;    /* Checksum sent by remote system */
   int MyCheckSum;     /* Checksum computed here */
   int HisSeqNo;       /* Sequence number decoded from received packet */
   int RetryCount;     /* Retry count before giving up on a packet */
   unsigned char c;
StartOver:
   RetryCount = 0;
   *length = ERR;  /* Default return parameter */
   while (RetryCount <= MAXRETRY)
      {
       if ((TempBuf = GetPacket (&PackLen)) == NULL)  /* We have an error */
          {
           sleep (2);       /* Wait 2 seconds - attempted error recovery */
           FlushModemInput (ModemFd);
           RetryCount++;    /* Consider TIMEOUT to be very bad */
           WriteLog("GetBlock: timed out on GetPacket", "", "", "");
           continue;        /* Go back to main loop and try again */
          }
       if (PackLen == 3)
```

```
    {
     if (TempBuf [0] == EOT)  /* Empty transmission - switch modes */
        {
         *length = EOT;  /* Set error condition to "EOT received" */
         free (TempBuf);
         return (NULL);  /* Return error condition */
        }
     else if (TempBuf [0] == CAN)  /* Remote had to abort cleanly */
        {
         *length = CAN;   /* Set error condition to "CAN received" */
         free (TempBuf);
         return (NULL);
        }
     else /* Phase error, etc. */
        {
         FlushModemInput (ModemFd);
         RetryCount++;     /* Note the failure */
         SendByte (NAK);  /* Improper control information */
         WriteLog("GetBlock: improper control info", "-", "sent NAK", ""
         EatModemInput;   /* Error recovery -- special to Honeywell */
         free (TempBuf);
        }
    }
else if (PackLen < 3)
    {
     sleep (2);  /* wait while things stabilize - this is adjustable */
     FlushModemInput (ModemFd); /* Phase error requires good cleanup */
     RetryCount++;
     SendByte (NAK);   /* Send negative acknowledgement */
     WriteLog("GetBlock: phase error - sent NAK", "", "", "");
     EatModemInput;
    }
else if (PackLen > 3)
    {
     c = TempBuf [3];  /* Save */
     TempBuf [3] = '\0';
     sscanf (TempBuf, "%x", &HisSeqNo);  /* Decode packet number */
     TempBuf [3] = c;  /* Restore */
     if ((HisSeqNo != SeqNo) && (HisSeqNo != SeqNo-1))
        {
         RetryCount++;     /* Bad data - don't go forever */
         SendByte (NAK);  /* Send negative acknowledgement */
         WriteLog ("GetBlock: Sequence number is bad","-","sent NAK","")
         EatModemInput;
         continue;         /* Get back to main loop */
        }
     c = TempBuf [PackLen-2];  /* Save */
     TempBuf [PackLen-2] = '\0'; /* Set up EOS for a second */
     sscanf (&(TempBuf[PackLen - 6]), "%x", &HisCheckSum);
     TempBuf [PackLen-2] = c;  /* Restore EOS - don't worry */
```

```
            MyCheckSum = CheckSum (TempBuf+3, PackLen-10); /* Compute CkSum */
            if (MyCheckSum != HisCheckSum)  /* Packet in error */
               {
                FlushModemInput (ModemFd);  /* Make sure buffer is clean */
                SendByte (NAK);  /* Send negative-acknowledgement control */
                WriteLog ("GetBlock: checksum's bad", "-", "sent NAK", "");
                EatModemInput;
               }
            else  /* All is A.O.K. */
               {
                SendByte (ACK);  /* Send positive acknowledgement */
                break;  /* Leave this loop, we have a good packet */
               }
         }
       RetryCount++;  /* See how long it takes */
      }
   /* Why did we leave the loop? */
   if (RetryCount > MAXRETRY)  /* Because of error */
      return (NULL);  /* We had an error */
   else
      {
       if (SeqNo == HisSeqNo)
          SeqNo = (SeqNo + 1) % 0x100;   /* Set up for next packet if good */
       else
          goto StartOver; /* Start over if old packet retransmitted */
       /* Set up return information - create block buffer */
       *endflag = TempBuf [ENDFLAG];  /* Set return parameter */
       *length = ENDFLAG - 4;  /* Set block length for caller (no EOS) */
       BlockBuffer = (unsigned char *) malloc (*length + 1);/* Allocate block
       if (BlockBuffer == NULL)  /* Don't keep going if problem exists */
          {
           WriteLog ("GetBlock:", "OUT OF MEMORY", "allocating block", "");
           exit (INTERNALERROR);
          }
       movmem (TempBuf + 3, BlockBuffer, *length);  /* Move good data */
       /* Get rid of storage allocated in lower level for packet */
       free (TempBuf);
       return (BlockBuffer);   /* Return pointer to good block of data */
      }
}


EOF
```

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>GETFIL.C

TIME: 1987/02/12 1615:19.4

```
#include "net.h"
#include "iocontrol.h"
#include "iocontrol.e"
/*
   GetFile - read the next file (if available) from remote system, taking
             care of retransmission, etc.  Either write a new file or
             return an error value.
*/
int GetFile ()
{
   int BlockLen;            /* Length of each new block read */
   int ETXFlag;                 /* Indicates EOT received */
   int Result;              /* Result from 'GetHeader' call */
   unsigned char *TempBuf; /* Block pointer for each read-in block */
   int MsgLength = 0;       /* Length of message (index to Message) */
   FILE *FileFp;            /* File pointer for temporary file */
   filename FileName;       /* Name of current incoming message */
   int i;


   /* For each file received, get header with message name */

   Result = GetHeader (FileName); /* Get header from remote system & audit */

   if (Result == EOT)
      return (EOT);            /* No new header arrived; end of transmission */

   if (Result == REJECT)       /* The message has been rejected */
      return ((int) NULL);     /* Exit gracefully, as if nothing happened */

   /* Create and receive new message */

   for (i = 0; i < strlen (FileName); i++)
       FileName[i] = toupper (FileName[i]);  /* Conver to uppercase */

   FileFp = CreateFile (FileName, MSGPROCQ);
   if (FileFp == NULL)
      {
       WriteLog ("GetFile:", "can't open file for message:", FileName, "");
       exit (RECOVERABLE);
      }

   SendByte (ACK);  /* Send message acknowledge only when completely ready */

   while ((TempBuf = GetBlock (&BlockLen, &ETXFlag)) !=
                                  (unsigned char *) NULL) /* Loop */
      {
       for (i = 0; i < BlockLen; i++)
           {
            if (TempBuf [i] == DLE)
```

```
            {
             i++;                          /* Skip the first DLE of any pair */
             if (i >= BlockLen)
                {
                 WriteLog ("GetFile:", "INTERNAL ERROR:", "unmatched DLE",
                           "at end of packet data");
                 exit (INTERNALERROR);
                }
            }

          if (TempBuf [i] == '\r')
             TempBuf [i] = '\n';

          fputc (TempBuf [i], FileFp); /* Write the next character */
         }
     MsgLength += BlockLen;  /* Keep track of length of entire message */
     if (ETXFlag == ETX)
          break;  /* Leave the loop if end-of-text received */
     free (TempBuf);  /* Get rid of storage allocated at top of loop */
     fflush (FileFp);
    }
fclose (FileFp);  /* Write the complete file */
/* Find out why we left while loop */
if (TempBuf != (unsigned char *) NULL)
    {
     if (ETXFlag == ETX)
        {
         free (TempBuf);  /* Let it go */
         WriteLog ("GetFile:", "file", FileName, "received OK");
         FileNQ (FileName, MSGPROCQ);  /* Give the file to msg-proc */
         return ((int) NULL);
        }
    }
else if (BlockLen == EOT)  /* An end-of-transmission was received */
    {
     WriteLog ("GetFile:", "EOT received", "but not expected", "");
     WriteLog ("GetFile:", FileName, "not saved", "");
     Remove (FileName, MSGPROCQ); /* Get rid of it */
     return (INTERRUPTED);  /* Indicate we were interrupted */
    }
else if (BlockLen == CAN)  /* The remote system shut down suddenly */
    {
     WriteLog ("GetFile:", "CAN received", "(remote aborted)", "");
     WriteLog ("GetFile:", FileName, "not saved", "");
     Remove (FileName, MSGPROCQ);
     return (ABORTED);
    }
else  /* Process errors */
    {
     WriteLog ("Receiver:", "timed out -", "FAILED", "");
```

```
        Remove (FileName, MSGPROCQ);   /* Throw away incomplete file */
        return (ERR);
      }
}


EOF
```

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>GETPAC.C

TIME: 1987/02/12 1617:18.0

```
#include "net.h"
#include "iocontrol.h"
#include "iocontrol.e"


#define ReadBurst(a,b) read(ModemFd,a,b)

/*
   GetPacket - retrieve a packet (control information or data) from the
               remote system, allocating storage for it, and return a
               pointer to the caller, who will release storage when done.
*/

unsigned char *GetPacket (length)

int *length;  /* Return parameter is length of packet in bytes */

{
   unsigned char TempPacket [MAXPACKET];  /* Allocate static storage */
   unsigned char *Ptr;    /* Pointer for returned storage */
   int CurMax;  /* Current maximum index of packet */
   int Result;  /* Result from ReadBurst, length of data if positive */
   int RetryCount = 0;

Restart:

   *length = 0;

   Result = ReadBurst (TempPacket, MAXPACKET);  /* Read until '\r' */
   CurMax = (Result > 0 ? Result - 1 : 2);

   while ((Result > 0) && !((TempPacket[CurMax-1] == EM) &&
                            (TempPacket[CurMax-2] == DLE)))
         {
          Result = ReadBurst (&TempPacket[CurMax+1], MAXPACKET-CurMax-1);
          CurMax += (Result > 0 ? Result : 0);
         }

   if (Result <= 0)
      {
       WriteLog ("GetPacket:", "received poorly-formed packet", "", "");
       SendByte (NAK);
       if (++RetryCount > MAXRETRY)
          {
           WriteLog ("GetPacket:", "retry limit reached", "", "");
           return (NULL);
          }
       goto Restart;
      }
```

```
   /* A valid packet was received, in that it terminated in DLE/EM */

   *length = CurMax;  /* Set return value to number of characters read */

   Ptr = (unsigned char *) malloc (*length + 1);  /* Allocate storage */
   if (Ptr == (unsigned char *) NULL)
      {
       WriteLog ("GetPacket:", "OUT OF MEMORY", "allocating for packet", "");
       exit (INTERNALERROR);
      }

   movmem (TempPacket, Ptr, *length);  /* Move data to new buffer */
   return (Ptr);
}


EOF
```

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>GETHEA.C

TIME: 1987/02/12 1618:25.2

```
#include "net.h"
#include "iocontrol.h"
#include "iocontrol.e"
/*
   GetHeader - read header for next incoming message.  The message header
               is merely a packet containing the name of the message file.
               Verify that the message is 'new' and return good if so,
               send NAK if message rejected for being a 'repeat'.
               Return EOT if remote has no more files to send.
               Calling routine will send corresponding ACK when all is done.
*/
int GetHeader (messagename)
char *messagename;
{
   unsigned char *ResponseBlock; /* Block from 'getblock', allocated therein *
   int  EndFlag;           /* End of text flag - needed for 'getblock' */
   int  BlockLength;       /* Length of block (message-name) returned */
   if ((ResponseBlock = GetBlock (&BlockLength, &EndFlag)) == NULL)
      if (BlockLength != EOT)
         {
          /* Unexpected condition - no error-handling here */
          WriteLog ("GetHeader:", "message header received improperly", "", ""
          exit (BADCONNECTION);  /* Synchronization bad between messages */
         }
      else
         return (EOT);      /* Remote had no more files to transmit */
   if (BlockLength > FILENAMELEN)  /* Don't permit overindexing */
      {
       /* Remote message name is not of proper form */
       WriteLog ("GetHeader:", "message name is invalid", "", "");
       free (ResponseBlock);
       exit (BADCONNECTION);
      }
   /* Check message name against records -- not implemented now */
   strncpy (messagename, ResponseBlock, BlockLength);  /* set return text */
   messagename [BlockLength] = '\0';
/* SendByte (ACK); -- no longer send ACK here -- send it in GETFIL */
   FlushModemInput (ModemFd);
   free (ResponseBlock); /* release storage */
   return ((int) NULL);   /* Good result */
}
```

EOF

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>PREEMP.C

TIME: 1987/02/12 1647:53.7

```
#include "net.h"
#include "iocontrol.h"
/*
   Preemption - return boolean value indicating whether or not the current
                queue has been interrupted so that a higher-priority message
                can be sent to another system via the line in use.  Check
                for the file INTFILE in the given queue (directory).
*/
int Preemption (queuename)
pathname *queuename;  /* Name of directory to check for INTFILE file */
{
   FILE *fp;  /* Temporary file pointer to be released if successful */
   pathname IntFileName;  /* Storage for complete file path */
   sprintf (IntFileName, "%s/%s", queuename, INTFILE);  /* Construct path */
   if ((fp = fopen (IntFileName, "r")) == NULL)
      {
       /* The file was not found, or else we have a bad problem. Say 'no' */
       return (FALSE);  /* Indicate no need to relinquish line */
      }
   fclose (fp);  /* Make sure we close the file */
   return (TRUE);  /* Indicate that we have agreed to terminate */
}
```

EOF

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>READBU.C

TIME: 1987/02/12 1702:06.7

```
#include "iocontrol.e"

/*
   ReadBurst - read a burst of information from the modem port.  Given
               Asynchronous configuration of Honeywell ports, assume that
               a burst will end in '\r', which will be in the buffer,
               and that a read will not exceed given character limit.
*/

int ReadBurst (buffer, maxsize)

unsigned char *buffer;
int maxsize;

{
   return (read (ModemFd, buffer, maxsize));
}
```

EOF

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>RECEIV.C

TIME: 1987/02/12 1705:59.7

```
#include "net.h"
#include <signal.h>
#include "iocontrol.h"
#include "iocontrol.e"
/*
   Receive - get a byte from the remote system (via the modem), waiting
             a maximum of MAXDELAY seconds for the byte.  If a character
             arrives, return it to the caller.  If a byte is not
             received within the threshold, return the error condition.
*/
int Receive ()
{
   int alrmint();
   int i = 0;          /* Initialize wait to 0 */
   char *cptr;         /* Character to read from port */
   unsigned char c;    /* Character to return */

   cptr = malloc (1); /* Just allocate a byte */
/* signal (SIGALRM, alrmint);
   alarm (MAXDELAY);         /* Set timeout interval */
   if (read (ModemFd, cptr, 1) > 0)
      {
/*     alarm (0);           /* reset timer */
       c = *cptr;           /* Get that unsigned 8-bit character */
       free (cptr);         /* Return storage */
       return ((int) c);    /* We got a character */
      }
/* alarm (0); */
   free (cptr);
   return (ERR);       /* Return ERR if timeout or error on read */
}
/*int alrmint ()
{
   fprintf (stderr, "---ALARM!!---\n");
   fflush (stderr);
   fclose (ModemFp);
   ModemFp = fopen ("!DIAL03", "rb+");
   return (ERR);
}*/
```

EOF

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>RECEIVE.C

TIME: 1987/02/12 1710:38.0

```
#include "net.h"
#include <signal.h>
#include "iocontrol.h"
#include "iocontrol.e"
/*
   Receive - get a byte from the remote system (via the modem), waiting
             a maximum of MAXDELAY seconds for the byte.  If a character
             arrives, return it to the caller.  If a byte is not
             received within the threshold, return the error condition.
*/
int Receive ()
{
   int alrmint();
   int i = 0;          /* Initialize wait to 0 */
   char *cptr;         /* Character to read from port */
   unsigned char c;    /* Character to return */

   alarm (0);
   cptr = malloc (1); /* Just allocate a byte */
   signal (SIGALRM, alrmint);
   alarm ((unsigned) MAXDELAY);        /* Set timeout interval */
   if (read (ModemFd, cptr, 1) > 0)
      {
fprintf(stderr, ".");
fflush(stderr);
       alarm (0);          /* reset timer */
       c = *cptr;          /* Get that unsigned 8-bit character */
       free (cptr);        /* Return storage */
       return ((int) c);   /* We got a character */
      }
   alarm (0);
   free (cptr);
   return (ERR);       /* Return ERR if timeout or error on read */
}
int alrmint ()
{
   return (ERR);
}
```

EOF

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>SEND.C

TIME: 1987/02/12 1712:47.4

```
#include "net.h"
#include "iocontrol.h"
#include "iocontrol.e"
/*
   Send - output a 'packet' to the output device (ModemFd).  Do not make
            any assumptions about format (permit binary if needed).
            Place an EM (End-of-Message) followed by an ASCII carriage-
          return (CR) after the packet as part of the low-level protocol.
*/
int Send (string, length)
unsigned char *string; /* String of characters to send (may be binary) */
int  length;  /* Length of string to send in bytes */
{
   unsigned char Tail [3];

   write (ModemFd, string, length);  /* Output the string */
   Tail [0] = DLE;  Tail [1] = EM;  Tail [2] = CRET;
   write (ModemFd, Tail, 3);       /* Always terminate with a carriage-return *
   write (ModemFd, "\n", 1);  /* Force Honeywell to flush output */
}
```

EOF

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>SENDBL.C

TIME: 1987/02/12 1713:19.4

```
#include "net.h"
#include "iocontrol.h"
#include "iocontrol.e"
/*
   SendBlock - send a block to the remote system until received correctly
               or until retry limit is reached.
*/
int SendBlock (data, length, endflag)
unsigned char *data;     /* Data to be transmitted, null-terminated */
int  length;    /* Length of block in bytes (can't use string fns) */
int  endflag;  /* Flag indicating this is the last block of a message */
{
   int RetryCount = 0;  /* Count of number of negative acknowledgements */
   while (RetryCount <= MAXRETRY)  /* Do not try forever */
      {
       FlushModemInput (ModemFd);    /* Dump the garbage */
       SendPacket (data, length, endflag);  /* Try to send the packet */
       if (WaitAck () == (int) NULL)
          {
           SeqNo = (SeqNo + 1) % 0x100; /* Give us next 2-byte sequence no. */
             break;                     /* Exit loop acknowledge received */
          }
       RetryCount++;
      }
   return (RetryCount <= MAXRETRY);  /* Return error condition */
}
```

EOF

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>SENDBY.C

TIME: 1987/02/12 1713:57.5

```
#include "net.h"
#include "iocontrol.h"
#include "iocontrol.e"
/*
   SendByte - send a single control byte followed by End-of-Message
              (EM) and Carriage-Return (CR) as dictated by the low-
              level protocol.
*/
int SendByte (byte)
unsigned char byte;
{
   unsigned char Tail [3];
   write (ModemFd, &byte, 1);  /* See send.c */
   Tail [0] = DLE;
   Tail [1] = EM;
   Tail [2] = CRET;
   write (ModemFd, Tail, 3);
   write (ModemFd, "\n", 1);     /* Forces Honeywell to flush output */
}
```

EOF

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>SENDEN.C

TIME: 1987/02/12 1714:22.5

```
#include "net.h"
#include "iocontrol.h"
#include "iocontrol.e"
/*
   SendEnq - send an enquire signal (ENQ), and await response with an
             acknowledge signal from master.  Time out after specified
             retry limit is exceeded.
*/
int SendEnq ()
{
   unsigned char *Buffer;  /* Response buffer allocated by GetPacket */
   int RetryCount = 0;     /* Counter for number of retries */
   int Length;             /* Response buffer length (not used) */
   FlushModemInput (ModemFd);  /* Clear input buffer before sending ENQ */
   while (RetryCount++ < MAXRETRY)
      {
       SendByte (ENQ);  /* Transmit enquire signal */
       if ((Buffer = GetPacket (&Length)) == NULL)
         {
          if (DebugLevel)
             WriteLog ("WaitEnq:", "Timed out waiting for ENQ", "", "");
         }
       else
          if (Buffer [0] == ACK)
             break;  /* Response was good, systems are in sync */
          else
             free (Buffer); /* release storage */
      }
   FlushModemInput (ModemFd);  /* Get rid of possible extra characters */
   if (RetryCount >= MAXRETRY)
      {
       WriteLog ("SendEnq:", "timed out awaiting", "contact with master.", "")
       exit (LOSTCONTACT);
      }
   free (Buffer); /* release storage */
}
```

EOF

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>SENDFI.C

TIME: 1987/02/12 1715:07.8

```
/* #define PREEMPTION  /* Set mode to check between blocks for interrupt */
#include "net.h"
#include "iocontrol.h"
#include "iocontrol.e"
extern pathname QueueName;  /* Reference semi-global variable */
/*
   SendFile - send a file to the remote system, breaking it into certain
                sized 'blocks' for better error detection.  Also preserve
              original file name.
*/
int SendFile (filepathname, messagename)
pathname *filepathname;
char *messagename;    /* Message file name - unique throughout network */
{
   FILE *FileFd;
   int BlockSize;  /* Size of each block transmitted */
   unsigned char *BlockPtr;
   int i;
   int EndOfText;
   unsigned int c;

   for (i = 0; i < strlen (messagename); i++)
       messagename[i] = tolower (messagename[i]);  /* Convert to lowercase */

   if ((FileFd = fopen (filepathname, "r")) == NULL)
      {
       WriteLog ("SendFile:", "cannot open message file:", filepathname, "");
       return (ERR);
      }

   /* Begin message introduction session */

   if (SendHeader (messagename) != ACK)  /* Send header for message name */
      {
       WriteLog ("SendFile:", filepathname, "rejected by remote", "");
       fclose (FileFd);
       return ((int) NULL);          /* Normal return - just don't send file *
      }

   /* Remote will accept message; begin transmission */

   BlockPtr = (unsigned char *) malloc (BLOCKLENGTH + 1); /* Allocate storage
   EndOfText = FALSE;   /* Set to transmit ETB after each block */

   while (!EndOfText)   /* Loop until all blocks are sent */
      {
#ifdef PREEMPTION
       /* First of all, check to see if a higher entity requests the line */
       if (Preemption (QueueName))
          {
```

```
            /* We must relinquish this line */
            SendByte (EOT);  /* Other system will understand */
            WriteLog ("SendFile:", "INTERRUPTED while sending", filepathname,
                      "");
            exit (INTERRUPTED);
           }
#endif
        BlockSize = 0;
        while (BlockSize < BLOCKLENGTH)  /* Blocks may overrun by at most 1 */
          {
           if ((c = getc (FileFd)) == EOF)
              break;       /* Exit the loop if end of file within a block */
            c = c & (unsigned int) 0xff;      /* Just keep the bottom 8 bits */
            if (c == DLE)  /* Check for data-link escape in file */
               BlockPtr [BlockSize++] = c;  /* Go ahead */
            if (c == '\n')
               BlockPtr [BlockSize++] = '\r'; /* Insert carriage-return */
            BlockPtr [BlockSize++] = c;      /* Place character in block */
          }
        /* We have a block, now transmit it */
        EndOfText = (c == EOF);  /* We read an end-of-file, end with ETX */
        if (!SendBlock (BlockPtr, BlockSize, EndOfText)) /* Send & Wait */
          {
           WriteLog ("SendFile:", "Timeout awaiting acknowledgement.", "", "");
           WriteLog ("SendFile:", "Could not transmit", filepathname, "");
           fclose (FileFd);
           return (ERR);  /* Probably lost contact */
          }
      }
   WriteLog ("SendFile:", "message", filepathname, "sent OK.");
   free (BlockPtr);  /* Free storage now */
   fclose (FileFd);
   return ((int) NULL);
}


EOF
```

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>SENDHE.C

TIME: 1987/02/12 1716:48.5

```
#include "net.h"
#include "iocontrol.h"
#include "iocontrol.e"
/*
   SendHeader - send current message identification header and await
                remote acceptance.  (Remote may deny a message by header
                alone, indicating it has already received the message.)
                Remote must respond with two ACK's.  The first indicates
                that the message i.d. header packet was received correctly,
                and the second is the remote's acceptance of the message.
*/
SendHeader (messagename)
char *messagename;          /* Name of message being sent */
{
   int NameLength;          /* Length of message name to transmit */
   int GoodResult;          /* Result from 'sendblock' call */
   int EndFlag = TRUE;      /* End of text flag for 'sendblock' */
   int Length;              /* Length of response buffer - space holder */
   int Result;              /* Result from 'waitack' */
   NameLength = strlen (messagename);  /* Compute length of string */
   FlushModemInput (ModemFd);  /* Clear input buffer before sending header */
   /* Send the block and wait for an acknowledgement */
   GoodResult = SendBlock (messagename, NameLength, EndFlag);
   if (!GoodResult)
      {
       WriteLog ("SendHeader:", "can't send message header:", messagename, "")
       exit (LOSTCONTACT);
      }
   /* Now acknowledge message acceptance - must get double acknowledge */
   Result = WaitAck ();  /* See if remote sends an acknowledgement */
   if (Result == ERR)
      {
       WriteLog ("SendHeader:", "TIMED OUT waiting for acceptance:",
                 messagename, "");
       exit (LOSTCONTACT);
      }
   FlushModemInput (ModemFd);  /* Make sure no garbage remains */
   return (Result == (int) NULL ? ACK : NAK);  /* Return acceptance flag */
}
```

EOF

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>SENDNA.C

TIME: 1987/02/12 1717:47.1

```
#include "net.h"
#include "iocontrol.h"
#include "iocontrol.e"
/*
   SendName - send this system's node name to remote system for secondary
              login and identification.  (Remote system needs to know who
              called so it can check for outgoing messages for that system.)
*/
SendName ()
{
   char *ThisSiteName;      /* Name of current site */
   int SiteNameLength;      /* Length of site name to transmit */
   int GoodResult;          /* Result from 'sendblock' call */
   int EndFlag = TRUE;      /* End of text flag for 'sendblock' */
   int Length;              /* Length of response buffer - not used */
   int Result;              /* Result from 'waitack' */
   ThisSiteName = MyName ();  /* Get the name of this node */
   SiteNameLength = strlen (ThisSiteName);
   FlushModemInput (ModemFd);  /* Clear input buffer before sending name */
   /* Send the block and wait for an acknowledge */
   GoodResult = SendBlock (ThisSiteName, SiteNameLength, EndFlag);
   FlushModemInput (ModemFd);  /* Get rid of possible extra information */
   if (!GoodResult)
      {
       WriteLog ("SendName:", "could not achieve","validation from slave", "")
       exit (LOSTCONTACT);
      }
   /* Now acknowledge login - must get double acknowledge */
   Result = WaitAck ();  /* See if remote sends an acknowledgement */
   if (Result == NAK)
      {
       WriteLog ("SendName:", "BAD SITE NAME", "(remote does not know me)", ""
       exit (BADCONNECTION);
      }
   else if (Result == ERR)
      {
       WriteLog ("SendName:", "TIMED OUT waiting for second ACK", "", "");
       exit (LOSTCONTACT);
      }
   /* Simply returning indicates success.  Errors are fatal, causing exit. */
}
```

EOF

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>SENDPA.C

TIME: 1987/02/12 1720:25.2

```
#include "net.h"
#include "iocontrol.h"
#include "iocontrol.e"
/*
   SendPacket - construct a 'packet' with checksum
                    using DLE as a data-link escape symbol (precedes ETX/ETB).
                    The parameter "end" indicates if this is the end of a mess
\ce
                    or if there is more to come.
*/
int SendPacket (data, length, end)
unsigned char *data;    /* Pointer to character data (binary permitted) */
int  length;  /* Length of packet to transmit */
int  end;     /* End of text flag */
{
   unsigned char *Packet;  /* Must allocate storage in this routine */
   int  Ptr;       /* Pointer into packet */
   int  Code;      /* Checksum code returned */
   int  i;
#ifdef DEBUG
   FILE *tmp=fopen("/usr/taccnet/log/proto.log", "a");
#endif
   Packet = (unsigned char *) malloc (length + PKTOVERHEAD);  /* Reserve data
   if (Packet == NULL)
      {
       WriteLog ("SendPacket:", "OUT OF MEMORY", "allocating for packet", "");
       exit (INTERNALERROR);
      }
   sprintf (Packet, "%.2X", SeqNo);  /* Encode packet number - ASCII hex */
   Packet [2] = STX;
   for (i = 0; i < length; i++)    /* Copy data section into packet */
         Packet [i+3] = data [i];  /* (Transparency already taken care of) */
   Ptr = i + 3;
   if (Ptr > (length + PKTOVERHEAD))
      {
       WriteLog ("SendPacket:", "memory violation", "building packet", "");
       exit (INTERNALERROR);
      }
   Packet [Ptr++] = DLE;  /* Place Data-link Escape before end-text marker */
   Packet [Ptr++] = (end ? ETX : ETB);  /* End-of-text or -text-block */
   Code = CheckSum (Packet+3, Ptr-4);    /* C      e checksum on data & ETX */
   sprintf (Packet + Ptr, "%
   /* Now that the packe                              to low-level I/O */
#ifdef DEBUG
   fprintf(tmp,"---------                             -------\n");
   for (i=0;i<Ptr+4;i++)                              t[i]);
   fprintf(tmp,"\n-------                             --------\n");
   fclose(tmp);
#endif
   Send (Packet, Ptr + 4);                             n characters */
```

```
    free (Packet);   /* Free storage */
}

EOF
```

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>WAITAC.C

TIME: 1987/02/12 1727:15.4

```
#include "net.h"
#include "iocontrol.h"
#include "iocontrol.e"
/*
   WaitAck - wait for an acknowledgement signal from remote system and
     return error status if "NAK" received or if timeout occurs.
*/
int WaitAck ()
{
   unsigned char *ResponsePacket;  /* From "getpacket", allocated therein */
   int  ResultStatus;      /* Status to pass to caller */
   int  PacketLength;      /* Needed for call to "getpacket" (unused) */
   if ((ResponsePacket = GetPacket (&PacketLength)) == NULL)
      ResultStatus = ERR;  /* Timeout before packet was received */
   else
      if (*ResponsePacket == ACK)
           ResultStatus = (int) NULL;
      else if (*ResponsePacket == EOT)
          {
           WriteLog ("WaitAck:", "remote system weas preempted;",
                     "connection terminated", "");
           exit (GOOD);
          }
      else if (*ResponsePacket == CAN)
          {
           WriteLog ("WaitAck:", "remote system shut down suddenly;",
                     "connection terminated", "");
           exit (GOOD);
          }
      else
           ResultStatus = NAK; /* Differentiate from ERROR and error */

   if (ResponsePacket != NULL)
      free (ResponsePacket); /* Get rid of unneeded storage */
   return (ResultStatus);
}
```

EOF

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>WAITEN.C

TIME: 1987/02/12 1728:27.3

```
#include "net.h"
#include "iocontrol.h"
#include "iocontrol.e"
/*
   WaitEnq - wait for an enquire signal from remote system and
               return error status if timeout occurs.  Send acknowledgement
             signal if enquiry received.  This establishes synchronization
             with remote after local system is activated.
*/
int WaitEnq ()
{
   unsigned char *ResponsePacket; /* From "getpacket", allocated therein */
   int  PacketLength;             /* Needed for call to "getpacket" (unused) *
   register int Count = 0;        /* retry counter */
   while (Count++ < MAXRETRY)
      if ((ResponsePacket = GetPacket (&PacketLength)) == NULL)
         {
          if (DebugLevel)
             WriteLog ("WaitEnq:", "Timed out waiting for ENQ", "", "");
         }
      else
         if (*ResponsePacket == ENQ)
            break;
         else
            free (ResponsePacket); /* release storage */
   FlushModemInput (ModemFd);
   if (Count >= MAXRETRY)
      {
       WriteLog ("WaitEnq:", "Did not receive ENQ packet", "", "");
       SendByte (NAK);  /* Make sure remote understands */
       WriteLog ("WaitEnq:", "Bad connection - Goodbye!", "", "");
       exit (BADCONNECTION);  /* Could not synchronize */
      }
   free (ResponsePacket); /* release storage */
   SendByte (ACK);  /* Transmit acknowledge signal */
}
```

EOF

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>WAITNM.C

TIME: 1987/02/12 1730:21.5

```
#include "net.h"
#include "iocontrol.h"
#include "iocontrol.e"
/*
   WaitName - wait for the remote system to identify itself and
              return error status if timeout occurs.  Send acknowledgement
              signal if valid name received.  This ensures security
              and facilitates bidirectionality of system.
*/
int WaitName (RemoteName)
sitename RemoteName;  /* Name of remote system; storage must be pre-allocated
{
   unsigned char *ResponseBlock; /* Block from "getblock", allocated therein *
   int  EndFlag;           /* End of text flag - needed for 'getblock' */
   int  BlockLength;       /* Length of block (remote-name) returned */
   if ((ResponseBlock = GetBlock (&BlockLength, &EndFlag)) == NULL)
      {
       /* Unexpected condition - no error-handling here */
       WriteLog ("WaitName:", "did not receive remote name properly.", "", "")
       SendByte (NAK);  /* Make sure remote knows there's a problem */
       exit (BADCONNECTION);  /* Synchronization bad at initial connection */
      }
   if (BlockLength > SITENAMELEN)  /* Don't permit overindexing */
      {
       /* Remote system name is not of proper form */
       WriteLog ("WaitName:", "remote name is not of proper form.", "", "");
       SendByte (NAK);  /* Tell remote there's a problem */
       free (ResponseBlock);
       exit (BADCONNECTION);
      }
   /* Move site name to caller's storage;  NOTE: may still be bad data */
   strncpy (RemoteName, ResponseBlock, BlockLength);
   /* We allow unknown sites to log in, since sites may come & go using */
   /* administrative messages to  change our tables. They still have to */
   /* get in through password security at the login level.              */
   if (ValidSite (RemoteName) == NULL)  /* Validate the site before going on *
       WriteLog ("WaitName:", "UNKNOWN SITE (", RemoteName, ")");
   FlushModemInput (ModemFd);
   free (ResponseBlock); /* Get rid of unneeded storage (smart) */
   SendByte (ACK);  /* Transmit acknowledge signal */
}
```

EOF

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>WAITNQ.C

TIME: 1987/02/12 1731:28.7

```
#include "net.h"
#include "iocontrol.h"
#include "iocontrol.e"
/*
   WaitEnq - wait for an enquire signal from remote system and
               return error status if timeout occurs.  Send acknowledgement
             signal if enquiry received.  This establishes synchronization
             with remote after local system is activated.
*/
int WaitEnq ()
{
   unsigned char *ResponsePacket; /* From "getpacket", allocated therein */
   int  PacketLength;             /* Needed for call to "getpacket" (unused) *
   register int Count = 0;        /* retry counter */
   while (Count++ < MAXRETRY)
      if ((ResponsePacket = GetPacket (&PacketLength)) == NULL)
         {
          if (DebugLevel)
             WriteLog ("WaitEnq:", "Timed out waiting for ENQ", "", "");
         }
      else
         if (*ResponsePacket == ENQ)
            break;
         else
            free (ResponsePacket); /* release storage */
   FlushModemInput (ModemFd);
   if (Count >= MAXRETRY)
      {
       WriteLog ("WaitEnq:", "Did not receive ENQ packet", "", "");
       SendByte (NAK);  /* Make sure remote understands */
       WriteLog ("WaitEnq:", "Bad connection - Goodbye!", "", "");
       exit (BADCONNECTION);  /* Could not synchronize */
      }
   free (ResponsePacket); /* release storage */
   SendByte (ACK);  /* Transmit acknowledge signal */
}
```

EOF

GENMSG

This section contains the functions used only by the Message Generator program (GENMSG).

PATH: ^ZSYS72>UDD>GOLDBERG>UNIX>GENMSG.C

TIME: 1987/02/12 1608:52.2

```
#include "net.h"

#define  TEMPFILEHEAD "ZG"
#define  MAXSITES 20

pathname LogFile; /* global LogFile for genmsg routines */

/* GenMsg is a program to generate messages for transmission to a remote syste
   It expects a priority as parameter 1 followed by a list of destinations.
   There must be at least 1 destination given.

   It will read the message body from stdin and build a message file
   or files which will be placed in the appropriate system message queue.
   If the option "-in filename" is given, input will be read from that file.
   Input may only be text data (not binary).

   Multiple destination paths or sites may be specified on the command
   line. They should be separated by one or more spaces. All messages will
   be given the same priority.

   The destination may be given as a path alias, an absolute path, or a
   path alias with an absolute path appended. The last token in the path
   may be a user id at the target site. If so, the message will be mailed
   to that user upon arrival at the site.

   destination ::= <alias>[!<path>] | <site>[!<path>]
   path ::= <site>[!<path>] | <site>!<user>
   site ::= a network node name defined in the site table
   user ::= a valid user id on the target site
*/

main (argc, argv)
int argc;
char **argv;

{
   FILE     *TermFd;
   FILE     *TmpFileFd;
   filename TmpFileName;
   pathname Path;
   char     *SiteList [MAXSITES+1];
   char     *PathList [MAXSITES+1];
   char     Priority [2+1];
   register int Ch;
   register int j,k;
   char     *InFileName;

   int      MakeMessage();
   char     *BuildPath ();
   FILE     *freopen();
```

```
/* set log file for WriteLog() */
sprintf (LogFile, "log/%s.log", GENMSG);

/* validate argument count */
if (--argc < 2)     /* there are at least two arguments */
   usage (argv[0]);

/* Get possible alternate input file name */
if (strcmp (argv[argc-1], "-IN") == 0)
   {
    InFileName = argv[argc];
    freopen (InFileName, "r", stdin);
    argc -= 2;
   }

/* get and validate message priority */
strcpy (Priority, argv[1]);
if ((strlen(Priority) != 1) || (atoi(Priority) < 0) || (atoi(Priority) > 9)
   {
    fprintf (stderr, "invalid priority - must be in range 0-9\n");
    usage (argv[0]);
   }

/* get list of destination sites */
for (j=2 ; j <= argc ; j++)
   {
    SiteList[j-2] = malloc ( strlen(argv[j])+1 );
    for (k = 0; k < strlen (argv[j]); k++)
       SiteList[j-2][k] = tolower (argv[j][k]); /* Add & convert to lower *

    SiteList[j-2][strlen (argv[j])] = '\0';
   }
SiteList[j-2] = NULL;  /* mark end of list */

for (k = 0, j = 0; SiteList[j] != NULL; j++)
    PathList[k++] = BuildPath (SiteList[j]);

PathList[k] = NULL; /* mark end of list */

if (PathList[0] == NULL)
   {
    fprintf (stderr, "no valid paths specified\n");
    usage (argv[0]);
   }

/* read the message into a temporary file */

sprintf (TmpFileName, "%s%d", TEMPFILEHEAD, NOW);
if ((TmpFileFd = fopen(TmpFileName, "w")) != NULL)
```

```
        {
         if ((Ch = getc (stdin)) == EOF)  /* empty message not allowed */
             {
              fprintf (stderr, "Sorry, empty messages are not allowed.\n");
              fclose (TmpFileFd);
              if (unlink (TmpFileName) == ERR)  /* delete the temporary file */
                 WriteLog ("GenMsg: can't unlink", TmpFileName, "", "");
              exit (ERR);
             }
         while (Ch != EOF)  /* copy text till end of file */
             {
              putc (Ch, TmpFileFd);
              Ch = getc (stdin);
             }
         fclose (TmpFileFd);
        }
    else
        {
         fprintf (stderr, "Can't allocate tmpfile for message input.");
         exit (ERR);
        }

    /* for each path in PathList, generate a message and enqueue it */

    for ( j = 0; PathList[j] != NULL; j++)
         if (!MakeMessage (Priority, PathList[j], TmpFileName))
           fprintf (stderr, "can't generate message\n");

    if (unlink (TmpFileName) == ERR)  /* delete the temporary file */
       WriteLog ("GenMsg: can't unlink", TmpFileName, "", "");

}

/* This function will generate a message for the site given in SiteName
   with the body of text contained in the stream Fd.
   Return value is TRUE if operation succeeds, FALSE otherwise.
*/

int MakeMessage (Priority, Path, Name)

char *Priority;
char *Path;
char *Name;

{
    FILE     *MsgFileFd;
    FILE     *Fd;
    filename MsgFileName;
    sitename FirstSite;
    register int Ch;
```

```
    StripMe (Path, FirstSite);  /* Get the first site in the list */

    /* create a message file to hold the message */

    if ((MsgFileFd = NewFile (MsgFileName, MESSAGETYPE, FirstSite)) == NULL)
        {
         fprintf (stderr, "GenMsg: Can't create %s\n", MsgFileName);
         return (FALSE);
        }

    /* write the message header into the message file */

    fprintf (MsgFileFd, ">%s\n>%s\n>%s\n", Priority, Path, MyName ());

    Fd = fopen (Name, "r");              /* reopen the message text */

    while ((Ch = getc (Fd)) != EOF)      /* copy the message text */
        putc (Ch, MsgFileFd);

    fclose (MsgFileFd);

    FileNQ (MsgFileName, FirstSite);    /* put message in proper system queue */

    return (TRUE);
}


/* This function will build a network path to the site given in SiteName.
   The SiteName may be a path alias, an absolute path, or an alias
   with an absolute path appended.
   Return value will be a pointer to the constructed path, or NULL if the
   path was invalid.
*/

char *BuildPath (SiteName)

char *SiteName;

{
    char     *RetVal;
    pathname DestPath;
    pathname Path;
    sitename FirstSite;

    char *strchr ();

    /* validate or expand the first site on the path */

     if (strchr (SiteName, '!') == NULL)
```

```
        strcpy (FirstSite, SiteName);
    else
        sscanf (SiteName, "%[^!\0]", FirstSite);  /* strip first site */

    if (ValidSite(FirstSite) == NULL)        /* undefined site */
      if  (!ValidPath(FirstSite,Path))       /* no path to the site */
        {
         fprintf (stderr, "Sorry, no path to %s.\n", FirstSite);
         return (NULL);
        }
      else  /* construct the desired path with info from ValidPath */
        sprintf (DestPath, "%s%s", Path, SiteName+(strlen(FirstSite)));
    else
      strcpy (DestPath, SiteName);  /* first site is defined */

   RetVal = malloc ( strlen(DestPath) + 1 );
   strcpy (RetVal, DestPath);
   return (RetVal);
}

int usage (name)
char *name;
{
   fprintf (stderr, "usage: %s priority dest [dest ...]\n", name);
   exit (ERR);
}


EOF
```

Final Technical Report, 1985 - 86

Combat Service Support Computer System Advanced Experimental Demonstrations

by

Principal Investigator:
Alton P. Jensen, Professor

Project Manager:
William O. Putnam, Research Scientist II

Project Staff:
Steven L. Goldberg, Research Assistant
Hong S. Shinn, Graduate Research Assistant

School Of Information and Computer Science
Georgia Institute Of Technology
Atlanta, Georgia 30332

Presented to:

U.S. Army Institute for Research In Management
Information, Communications, and Computer Science
(AIRMICS)

April 28, 1987

Contract No. DAHC0-85-C00012
Research Project No. G36-633

Table of Contents

1. Introduction

This document has been prepared as a technical report on work done under project G36-633 at the Georgia Institute of Technology surrounding the development and implementation of the Combat Service Support Control System (CSSCS) and the Command and Control (C^2) Database. In order to fully explore these issues, prototype CSSCS and C^2 Database systems were designed and implemented. Descriptions of these prototypes along with instructions for installation and operation of the prototype systems may be found in a companion report, *Considerations in the Design and Development of a Combat Service Support Computer System*, which was written as a technical specification for the prototype systems developed.

In the year ending in September 1985 the project group made good progress in investigating the issues surrounding the development of the Combat Service Support Control System. In these endeavors, fundamental issues such as dynamic network configuration, automatic routing, failure management, database backup and recovery, JINTACCS message processing, and network control were addressed. The year was concluded with a successful demonstration of a network incorporating the above features and a presentation of results to AIRMICS and assembled AIRMICS contractors.

The demonstration network consisted of a Honeywell DPS/6 running GCOS Mod 400, two Burroughs B26 TACCS systems running DISTRIX and XENIX, two IBM PC/AT systems running XENIX, and two ONYX computers running UNIX† System V.

The Honeywell DPS/6 is designated as the DAS3 in the ACCS network plan. The Burroughs B26 machines are beta test versions of the TACCS. The PC/AT and ONYX systems were used to simulate more TACCS systems and to provide a wider variety of hardware and software systems for testing.

The project began with the results of previous work undertaken to study the issues of communications and message passing in the environment of the CSSCS. These results were expanded upon and incorporated into the development of the prototype CSSCS system. In addition to message passing and failure detection, the prototype system addresses the issues of automatic message routing, dynamic network reconfiguration, remote node identification, network security, message generation and processing, C^2 database interactions, and database backup/recovery.

† UNIX is a trademark of AT&T Bell Laboratories.

The CSSCS environment is described as a loosely coupled *occasionally connected* network of independently operating computers. By *occasionally connected,* we mean that nodes in the network may come and go at will under normal operating conditions. In addition, the CSSCS environment is conducive to catastrophic node failures of network nodes. The necessity of mobility and risk of destruction are unavoidable features of the battlefield environment in which the CSSCS will operate. Node failures must be detected and handled by the system with a minimum of human intervention.

Development of the prototype system was done under the UNIX operating system. UNIX was chosen because of its wide availability in the class of small machines being considered for use by the Army as the TACCS. UNIX provides a full featured multi-tasking, multi-user environment conducive to the development of software. It is well supported in a variety of hardware environments including personal computers, minicomputers, and mainframes. The C programming language, standard on all UNIX systems, is a powerful, high level language incorporating structured programming features, flexible data structures of almost any desired level of complexity, and systems programming features for simple access to UNIX or to other processes.

Because the CSSCS environment requires real-time response to changing conditions and due to the unpredictable nature of communications in that environment, a robust, flexible, timesharing system is desirable. The UNIX operating system provides these qualities and offers a greater degree of portability among different types and sizes of machines than any other system currently available.

During the course of the project a series of Advanced Experimental Demonstrations (AED) were conducted. These are described fully in section 6. AED 1 demonstrated backup and recovery of the C^2 database via TACCNET. AED 2 demonstrated the C language TACCNET implementation on the Honeywell DSP/6 under GCOS. A copy of the scenario set used for the demonstrations is provided in Appendix 2.

Copies of all presentation materials developed during the project are provided in Appendix 3.

A set of data flow diagrams for the main components of the TACCNET system is provided in Appendix 4.

2. Overview

The CSSCS is intended to perform a prescribed set of functions in a designated environment. The details of these functions and the nature of this environment impose certain constraints on the system. In this section we will define the functions to be performed by the CSSCS and describe the CSSCS environment.

The CSSCS is intended to support the CSS commander on the battlefield at the corps, division, and brigade levels. The term "CSS commander" is refers to the officer responsible for managing the CSS function at any particular site. CSS is defined as the functional areas of supply, maintenance, field services, transportation, personnel, and health services found in the divisions, corps, and theater Army.

2.1. CSSCS Functions

The CSSCS is an information system for the Combat Service Support (CSS) node of the Army Command and Control System (ACCS). As such, the CSSCS will interface will the other four nodes of the ACCS as well as with other functional systems within CSS (e.g. STAMMIS). The interface to external nodes will be via JINTACCS messages, both sent and received. The interface to STAMMIS will be limited to the extraction of data which will be posted to the Command and Control (C^2) database which will be an integral part of the CSSCS.

The basic functions of the CSSCS are:

- to provide a transport and communications network for information exchange among CSS units, primarily in the form of JINTACCS messages;
- to provide a database of information for use by CSS commanders and personnel in the performance of CSS functions;
- provide decision support functions to the CSS commanders on the battlefield.

2.2. CSSCS Environment

The CSSCS will operate in the battle field environment of the modern Army. This requires mobility and portability of all systems as well as transparency with respect to communications media. CSS units will appear as nodes in a loosely connected network capable of frequently changing topology. Nodes may join and leave the network at will as they change locations in the battlefield environment. Nodes are also subject to catastrophic failure due to enemy activity.

These elements of the environment require a network which is able to detect arrivals, departures, and failures and adjust operations accordingly.

The CSSCS must be able to detect errors in the routing or delivery of a message and reroute the message as necessary to ensure timely delivery to an appropriate CSS unit. In the event that a CSS node is rendered inoperative, it will be necessary to recover its C^2 database from a backup (at another node) and reconstitute the database by collecting all messages sent to the node after the backup and before the failure. It may be necessary for one CSS node to perform the function of a down node, taking its place in the network and carrying out the function of the down node until that node can be replaced.

The modern battlefield will offer a variety of communications media including existing telephone networks, microwave links, optical links, packet radio, and other more traditional media. Since it may not be possible to determine in advance the media available for CSSCS data transfer, it is desirable to have a system which is independent of communications medium. Limited bandwidth for digital communications encourages reduction of data redundancy in message formats and message redundancy in reporting systems.

The CSSCS environment places the following constraints on the CSSCS:

- restricted bandwidth for communications
- media transparent communications
- nodes join and depart the network at will
- nodes subject to catastrophic failure
- must provide distributed backup and recovery of C^2 databases
- must automatically route messages for timely delivery
- must detect failures and reroute messages accordingly
- messages in JINTACCS format

The system must observe these constraints and carry out its functions with a minimum of operator intervention.

3. Design Issues

The design issues involved in the development of the TACCNET system are discussed fully in the technical specification and will not be repeated here. We will, however, highlight the major design decisions made in the course of the system development. The design issues to be discussed in this section are those which impact the system at the top level. These decisions had a major impact on the nature of the software developed and on its functionality.

3.1. Network Topology

The Army is organized in a hierarchical fashion with responsibilities distributed among various internal organizations. These organizations are related through the chain of command in a formally defined hierarchical manner. The organizations within CSS which use the CSSCS will have a defined hierarchy, and it is reasonable to assume that their reporting and communications will observe this hierarchy. For our purposes, communications will be in the form of JINTACCS messages transmitted among CSSCS nodes.

3.1.1. Connectivity

Two types of networks are possible: a fully connected network in which any node can contact any other node as needed, or a polled network in which some nodes are only contacted by other nodes at specified intervals. The TACCNET prototype is a fully connected system. This choice was made to insure that important messages of high priority would not have to wait for a higher-echelon node to call in order to be transmitted and processed. In the prototype, nodes call one another whenever they have messages to deliver. They first check to be sure that a conversation with the desired system is not already in progress. Priority messages get fastest possible service, preempting routine messages or conversations if necessary. In a polled system, messages would have to wait (half the polling interval, on the average) for the next call from the master system regardless of priority.

The cost of a fully connected system is redundancy of capability, increased system complexity, greater overhead in communications, and increased usage of communications bandwidth. A polled system would tend to cut down the number of calls and increase the size of the transmissions as larger batches of messages would accumulate between contacts. The cost is mainly in the area of increased delay in propagating information through the system. In the battle field environment, timeliness of information can often be critical. There is also the matter of deciding who polls whom, how often to call, and how to handle high priority messages. It is possible to envision a compromise, where some nodes are polled and others communicate at will. This type of system could take advantage of the inherent hierarchical relationships between superior and

subordinate elements within CSS.

Store and forward message passing could be used to allow nodes which do not know how to contact each other directly to communicate through intermediate nodes. A modified "post office" scheme can be used to handle messages for unknown destinations - they are passed on to a designated node which may know how to deliver them (or may pass them on to another designated node, and so on). This would allow nodes to restrict their database of network contacts to those needed for routine operations.

3.1.2. Priority Message Handling

During the development of TACCNET we were unable to find information detailing the exact messages used within CSS and the frequencies of transmission of those messages. We did encounter some documentation which indicated that there would be response time criteria associated with some JINTACCS messages. The expected response times ranged from less than one minute to over a week. This implies a need for different classes of service for messages with different response time requirements. It is also reasonable to expect that there will be messages of different priorities (flash, immediate, secret, etc.) requiring different levels of service.

The question then arises: what types of special services are required for these types of messages? We may expect, at least, that there will be a requirement for immediate or fastest possible delivery and processing. There may also be a need for immediate reply while the sending node remains on the line waiting. Processing of database queries could be stratified to provide immediate service for queries at top priority levels. Ports and connections which are busy with routine transmissions may be preempted for priority transmissions. The incorporation of these capabilities complicates the system and can introduce the possibility of deadlocks or race conditions as processes of different priority compete for resources.

There is also the matter of node failures. What is to be done when there is a priority message for a site that cannot be contacted? Should the message be immediately rerouted to a designated backup node for prompt action? Does the type of message affect the action to be taken? If so, a knowledge base for the messages will be required for the system to determine the correct action for each type of message and each priority level. Obviously, this can become quite a serious consideration in the design of the system. The bottom line is that the system should give fastest possible service to important messages; preferably without degrading the performance of the network with respect to routine traffic.

The TACCNET prototype provides preemptive service for priority messages, inserting them into active conversation streams where possible, preempting routine transmission if necessary, and scheduling calls and resources to give fastest possible service to priority messages. Only two levels of messages are recognized: priority messages and routine messages. Priority messages get fastest possible service while routine messages are served on a first-come-first-served basis. In a fielded system there will probably be a need for more different levels of priority, but we do not know at this time how many or what types of service they would require.

3.1.3. Alternate Sites

In the event that a node is down and cannot be reached, what is to be done with messages for that node? Some messages are routine reports and can wait until the site returns to action and calls for them, but others (especially priority messages) may require fast or even immediate action. With this in mind, the TACCNET prototype uses a table of designated *alternate sites* for each node in the network to re-route messages targeted for inactive sites. The method used is to keep a list of sites which can take over the processing of important messages during the node's absence. The list is traversed in order; if the first alternate site is also unavailable, the next one on the list is tried. Courtesy copies of all message sent to alternate sites should be kept and delivered to the original destination site once it has returned to operation.

This method implies a hierarchy of alternate sites in case of failure. This is consistent with the organization of CSS and the military chain of command. Without more information on the internal operation of CSS and the usage of JINTACCS messages for CSS functions we cannot determine what the hierarchy (and therefore the ordering of alternate sites) should be. There is also the question of how many alternate sites to put on the list. Should the chain stop at some point or should it continue all the way up the CSS chain of command. What is to be done with a message when no alternate sites are available?

3.2. Routing

We have already discussed the issues surrounding the topology of a CSSCS network. Regardless of the type of network proposed, it must be able to route message efficiently and automatically through the network. The user should be relieved of the details of selecting a path and should only need to know the desired destination of his message. In the case of routine or automated reports and queries, the system should keep track of message origins and destinations, leaving the operator responsible only for message composition or auditing.

Due to the dynamic nature of the CSSCS network environment, there are advantages to implementing a store and forward message passing system.

When a message cannot be delivered directly to the desired site, it can be routed through and intermediate site which may have an active link to the desired site. Messages labeled for unknown destinations could be passed on to a designated "post office" site responsible for maintaining a complete database of network nodes and addresses. New nodes or nodes returning to action could call in to the post office to register and pick up waiting mail. The post office site could then distribute the new node's address to the rest of the network.

A side effect of the store and forward capability is the ability to route a message through a chain of intermediate sites to a final destination. Message paths would be composed of a sequence of node names. This could be used to broadcast messages of general importance to related groups of nodes or to use a specific set of links so as to avoid down or unreliable links. Ordinarily the user would supply only the final destination node name and the system would choose the shortest available path to that system, probably a direct connection via dialup. The user would, however, have the ability to override the system choice and specify a particular path. Aliases could be maintained by the system for complex or lengthy paths allowing the user to send a message to a designated group of nodes without remembering all of the nodes and their order or connectivity.

The TACCNET prototype provides store and forward message passing, automated path selection with optional user override, and path aliasing. It does not provide the post office method of dealing with undeliverable messages but does provide mail holding for departed or inactive sites.

3.2.1. Message Forwarding

We have already discussed alternate sites and message passing. In a network where nodes are expected to be mobile and to enter and leave the network at random we must provide a means for forwarding messages to appropriate nodes in the event that the designated recipient cannot be reached. This means that the system must have a set of criteria for use in evaluating the state of a node in the network. These criteria will be used to decide whether a node is down, temporarily unavailable, active, or destroyed. They system must be able to automatically decide whether to hold messages and keep trying to contact the remote site or to forward the messages to another site for delivery or processing.

The system must monitor the state of each node and take appropriate actions to maintain connectivity and continued operability. This may require the automatic rerouting of messages to insure prompt processing. It may require generation and maintenance of courtesy copies of messages for bypassed nodes so that they may be brought up to date when they return to action. It may require special handling of priority messages for down nodes when it is not

acceptable to wait through the retry process.

The TACCNET prototype provides the capability to determine the state of a site and to automatically route messages around a down site while keeping courtesy copies of all messages for bypassed sites. The courtesy copies are delivered when the site is successfully contacted and the site is restored to active status. The system keeps track of the current state of each node and keeps a record of the last successful contact as well as the number of failed attempts to contact a down site. Sites are declared to be down when the number of failed contact attempts exceeds a user determined threshold.

3.3. Failure Management

The discussion of message rerouting brings up the topic of link failure. There are different classes of failures which the system must be able to recognize and handle. The system may be limited in its ability to recognize some types of failures by the limitations of the communications equipment.† In any event, the system must be smart enough to distinguish between local failures (eg. can't dial out) and remote failures (eg. no answer or no login at remote modem).

3.3.1. Classes of Failures

The first class of failure is the local failure. This includes conditions such as no available ports, no response from local modem, modem unable to dial out, and inactive phone line. These conditions indicate local hardware or system problems and should not count against the remote site's connection history. They should not be considered when trying to determine whether a remote site is up or down. The proper response to these conditions will usually be to notify the operator and wait for correction of the situation. The incident should be logged automatically so that patterns of performance may be analyzed.

The second class of failure is the remote failure. This includes no answer from remote modem, busy signal, no carrier, no login prompt from remote system, and login or startup failure. These conditions span a range of problems from malfunctioning hardware to invalid login id or password. When the remote system answers the phone but does not allow login and synchronization we know that the site is operational and not destroyed. The correct action may be to keep calling or to change the login id or password for that system. When there is no answer to the call, the site may be down or destroyed and the messages may need to be rerouted. If there was a busy signal, it may be sufficient to wait

† For example: the D. C. Hayes Smartmodem employed in the TACCNET prototype does not distinguish between the "busy signal" and the "no answer" condition. The Cermetek Infomate does.

a while and call back. If the call is answered but there is no carrier, there may be a problem with the modems (possible hardware incompatibility). In each case, a note must be kept in a log file describing the result of the attempted call and the possible cause of failure.

The third class of failure is the transmission interruption. This includes link failure during transmission, cancellation of transmission, and preemption for priority messages. These types of failures do not usually indicate that the site has gone down. It will usually be sufficient to retry the connection after a short delay and continue transmission at the point of interruption. If the remote site has in fact gone down, the failure will be detected and handled as a class 2 failure as described above.

The TACCNET prototype detects and handle each of these classes of failures. Because of certain hardware limitations, it does not have the desired degree of resolution for class 2 failure diagnosis. It does recognize login and startup failures as well as transmission interruptions. Log files are maintained for each site showing the history of contacts and messages transferred. A system table is kept for the sites to monitor site status and contact times. Another table is used to monitor local port activity.

3.4. C^2 Database Backup and Recovery

Since the CSS environment can be volatile and nodes may be destroyed, it is desirable to build into the network the capability to back up and restore important information. Examples of such information include network configuration tables and the unit commander's C^2 database. This requires the designation of backup sites for each node. These need not necessarily be the same as the alternate processing sites for the node, but that would be a logical choice.

To increase the possibility of being able to recover a lost site's information, the database should be backed up on more than one remote site, each at a different location. The system designer must decide how many backup sites to allow or provide and how to insure that they are kept current. Another issue is frequency of backups: how often do we take a snapshot of the data for backup?

When the failure occurs and recovery is desired, how will it be initiated? A message may be sent to one of the backup sites to request an upload of the last database backup. It will be important to know and validate the time of that backup. It will probably be desirable to request retransmission of any messages sent to the destroyed node after the date of the last backup. The method for requesting these retransmissions must not flood the network with redundant messages, but must make sure that all relevant information is obtained. It may be possible to avoid retransmission of old messages by restoring a snapshot

backup and updating it from a higher level node.

3.5. Node Emulation

The TACCNET prototype system provides the capability to run more than one copy of the system on a single physical machine. This allows a node to perform the functions of a down or departed node in addition to its own work. Other nodes in the network do not need to know that the down node is being emulated.

All that is required to set up an emulated node is to create a root directory for TACCNET to use to handle the emulated node's work, disseminate the new phone number for the emulated node, adjust the site tables of the local and emulated node to show that they are in fact resident on a single machine, and invoke TACCNET in the new root directory.

This capability can be used to maintain a logical configuration even when network nodes are destroyed.

4. TACCNET Prototype Features

The TACCNET system is fully described in the technical specification referenced earlier and we will not attempt to repeat that description here. In this section we will highlight the principal features of the prototype CSSCS and provide a general overview of the system and its capabilities.

4.1. Heterogeneous Communications

The system provides media transparent communications between machines of different types and sizes, requiring only that the TACCNET software be present on each machine. The TACCNET system will run on any machine which can run the UNIX operating system. This encompasses a broad range of hardware classes including personal computers, microcomputers, minicomputers, and mainframes. The TACCNET system has run on the following machines: ONYX, IBM PC/XT, IBM PC/AT, Burroughs B26 (TACCS), ATT 3B2, DEC VAX 11/780, Honeywell DPS/6. The system normally operates at 1200 bps over voice grade phone lines, but will operate over other media at any speed. The system is completely automated and uses auto-dial, auto-answer modems.

4.2. Binary data transfer

The TACCNET system transfers a file exactly as it is stored. Full 8-bit binary data transfer is assured by the protocol. In tests the system has been used to transfer executable programs between machines for remote execution.

4.3. Error detection and recovery

The system uses a 16 bit checksum to detect errors in transmission. Packets containing errors are retransmitted. Repeated errors are logged and notification is sent to the system operator.

4.4. Transaction logging and archiving

All messages passing through each node are logged and copied to an archive. Administrative operations and database transactions are also logged. If necessary, the archives can be searched and saved messages can be retransmitted. This is done when it is necessary to reconstitute the C^2 database for a unit whose database has been lost.

4.5. Bidirectional on-demand links

All communication links between nodes in the TACCNET network are bidirectional. Any machine can call any other machine. There are no delays due to

polling. It is possible to configure the system so that a site is polled and does not call other sites, but the normal configuration allows full connectivity. Sites call one another whenever they have messages to transmit. If two sites call each other at the same time, the collision is detected and one of the sites will back off. A site can converse with many sites simultaneously (depending on the number of available communications lines) but only one conversation at a time is allowed between any two given sites.

4.6. Tunable parameters

The performance of the TACCNET system on a given machine depends on a number of factors, including available memory, disk space, work load, and processor type. The TACCNET system has a number of adjustable parameters which can be used to tailor a configuration to the needs or limits of a particular site. These include: message forwarding, retry delays, error detection thresholds, archiving, courtesy copies, and scanning intervals.

4.7. Broadcast messages

The system allows one site to broadcast a message to all the sites it knows about from its own site table. Such a message is the propagated throughout the network until every node has received a copy. The system has the ability to check for and reject messages which have already been received so that unnecessary transmissions are minimized. It is also possible to send copies of a message to other designated sites or to route messages through specific sites or paths in the network.

4.8. Failure detection and management

The TACCNET system is designed to operate in a failure prone environment. In fact, node failures are considered to be routine, expected events and the system will detect such failures automatically and route messages around failed nodes. The system will automatically bypass a failed node and will keep a copy of the any messages for that node to be delivered when the node returns to service. Also, the system will monitor down nodes and call them at regular intervals to attempt to regain contact. Nodes may leave the system in a more controlled manner by notifying other nodes of their departure. In that case, mail for the departed nodes will be held until they return to service. Nodes can add, change, delete, and query information in the site tables of other nodes without operator intervention.

4.9. File transfer

The basic function of the system is file transfer between machines. The system transfers JINTACCS messages in text files and can also be used to transfer any

other text, data, or programs files between machines. Files are transferred through the system by the store-and-forward method. Multi-node paths through the system and multiple recipient transfers are possible.

4.10. Electronic mail

The TACCNET system has a gateway into the UNIX electronic mail system. It can therefore be used to send mail to users or processes at other sites in the network. The use of TACCNET is transparent to the user.

4.11. Priority message scheduling

The TACCNET system is designed to allow different classes of service for messages of different priorities. For demonstration purposes, messages were assumed to belong to one of two classes: priority messages requiring fastest possible delivery, or routine messages to be delivered on a first come first served basis. The system processes all messages as received and tries to deliver them as soon as possible. Priority messages, however, can cause the delay or interruption of routine messages in order to receive fastest possible service. The system detects the arrival of priority messages and will insert them into the communications stream ahead of remaining routine messages if possible. If necessary, the system will automatically preempt a communications port from a routine transmission in order to send off a priority message.

4.12. On-line JINTACCS message dictionary

The TACCNET prototype offers a JINTACCS message composition aid based on an on-line JINTACCS message dictionary. The dictionary is contained in a set of database relations stored under the UNIFY relational database management system. The system can generate messages directly from the C^2 database on command or can be used to prompt the user in sequence for the data to be formatted into a JINTACCS message. This frees the user from the task of remembering all the rules of JINTACCS.

4.13. Password security

The TACCNET features three levels of password security. First, users must supply a password at login time to gain access to a node in the system. Second, each node must supply a password whenever it logs in to another node to initiate communications. Third, each node can enable or disable access to the network administrative functions and to the C^2 database server by defining the appropriate userid in the system password file.

4.14. Portability

The TACCNET software is written in the C programming language under the UNIX operating system. Great care has been taken to avoid hardware dependencies and operating system version dependent features. The system has been compiled and run under the following different operating systems: IBM PC/IX, IBM XENIX 1.0, IBM XENIX 2.0, ATT UNIX System III, ATT UNIX System Vr2, GCOS, ONYX.

4.15. JINTACCS messages to and from C^2 database

The system provides the capability to generate JINTACCS messages automatically from data contained in the prototype C^2 database. The system can also receive JINATCCS messages and used them to automatically update information the the C^2 database. The C^2 database is a collection of relations in a UNIFY relational database schema.

4.16. Automated JINTACCS message composition interface

The system provides a JINTACCS message composition interface to assist users in composing JINTACCS messages for transmission by the system. The tool uses the on-line JINTACCS message dictionary to prompt the user for the necessary data and format the data into a valid JINTACCS message. The user may view the message at any stage of the composition. Upon completion the message may be edited, transmitted, or saved in a file.

4.17. Distributed C^2 database backup and recovery

The TACCNET system provides backup and recovery functions for the C^2 database. A set of command files may be used to send a copy of the database files to other nodes in the network for storage. The stored copy may be retrieved later in the event of a system failure. Transactions occurring after the saving of the database may be recovered by broadcasting a message requesting retransmission of all messages sent to the failed site after the database backup was made. This message is propagated through the network and all relevant messages are recovered from node archives and retransmitted. These transactions are then run against the backup copy of the database and an up to date database results.

4.18. Single machine emulation of multiple nodes

A feature is provided to allow a single machine to emulate multiple nodes in the network. This is useful in the event that a node is taken down for some period of time. The network functionality can be maintained without extensive

reconfiguration until the down node returns to service. It may also be used for testing purposes. All that is requires is that the operator create a set of directories for use by the emulated system and then start a copy of the TACCNET software running in those directories.

4.19. Automated network management via messages

Network modifications are made easy through the use of network administrative messages. These special messages allow an operator at one node to obtain or modify the network configuration information at other nodes. Messages are provided to add, delete, change, and query information at remote nodes. Operators may enable or disable the processing of these messages for security reasons.

4.20. Screen oriented menu interface for user

The TACCNET system is controlled through a screen oriented menu interface on the system console. The interface provides command menus and graphic displays to show the state of the system. Windows into system log files may be opened and closed as needed. Special commands for system initiation, termination, monitoring, and maintenance are provided.

4.21. Path definitions

Operators can define special paths through the network and can assign names to these paths. These path aliases can be used for commonly used paths and destinations to save time and typing.

5. Summary of Activities

In this section the efforts and activities of the project are summarized. The term of the project was from September 1895 to May 1986, but the project was a continuation of work from a previous project, G36-610. Furthermore, work from this project was continued into project G36-655 which began in May 1986 and extends through September 1987.

5.1. September 1985

In September the project group worked to maintain and expand the TACCNET prototype system. Improved login handling and broadcast message handling were developed. Error logging and message tracking systems were improved. Other planned improvements to TACCNET included: screen editing capability for JINTACCS messages, console monitoring facilities for network observation, message display utilities for demonstrations, robust login procedures, and administrative utilities.

The Honeywell GCOS C compiler was ordered and was expected to arrive in October. Preparations were made to transfer all TACCNET source code from the IBM PC/AT to the Honeywell DPS/6. Substantial changes were expected to be required in the I/O routines due to differences between UNIX and GCOS.

Design work began for the database backup and recovery system. It was expected to involve a combination of shell scripts and C functions. Modifications to the Message Processing system were required.

5.2. October 1985

A task plan for the project was prepared and presented to AIRMICS. Work areas included UNIX/GCOS conversion of TACCNET, JINTACCS message preparation facilities, C^2 database backup and recovery, and TACCNET user interface development.

Design work continued on the database backup and recovery system. The system was to use a prototype C^2 database implemented using the UNIFY relational database management system. The system was intended to allow a CSSCS node to transmit a copy of its C^2 database to another node for storage and then retrieve that copy on demand without operator intervention on the remote system. All messages processed after the date of the backup would then be retransmitted upon receipt of a special broadcast message.

A TACCS computer was received by AIRMICS. It was installed and brought up under a version of UNIX called DISTRIX 1.0. Plans were made to port TACCNET to the new system as soon as possible.

An advance copy of the Honeywell GCOS C Compiler arrived but was not accompanied by any documentation. AIRMICS personnel worked on getting manuals and on expediting our order.

5.3. November 1985

Three work areas were identified: TACCNET improvements, Honeywell C conversion, and database backup and recovery. The conversion of Honeywell DPS/6 code from PASCAL to C were to begin as soon as the Honeywell DPS/6 C compiler was received and installed. The other work was begun immediately.

The planned improvements to TACCNET included: screen editing capability for JINTACCS messages, console monitoring facilities for network observation, message display utilities for demonstrations, robust login procedures, and administrative utilities. These improvements are primarily of a cosmetic nature and will improve the clarity and impact of TACCNET demonstrations.

While the DAS3 described in the CSSCS environment was not required to receive JINTACCS messages from the TACCS at that time, it was expected that that capability could be provided during the C code conversion. Some groundwork was required so that existing C code from the UNIX based TACCS environment could be transferred to the Honeywell DPS/6 and retrofitted to run under GCOS. Numerous changes were required due to the disparate nature of the GCOS and UNIX operating systems, but the interfaces and operations of the programs were designed to be the same under both systems.

An examination of DISTRIX 1.0 uncovered or confirmed numerous deficiencies which were reported. DISTRIX 2.0 was reported to fix some of the bugs. Jim Kearns worked on getting a pre-release copy for the project.

A presentation based on the final report of the previous contract (g-36-610) was given. In addition, a briefing on TACCNET was scheduled for December 18.

5.4. December 1985

Documentation for the Honeywell C compiler was received in December. Work began on transferring the UNIX-based C code for TACCNET from the PC/AT to the DPS/6. The transfer was completed without incident using

simple terminal emulation and file transfer programs. Initial attempts to compile the C code were moderately successful with most routines compiled on the first try. Some compiler differences were encountered and corrections made, resulting in the successful compilation of approximately 95% of the C code.

A problem emerged when we attempted to link the compiled code into an executable program. The linker for the C compiler contains a bug which prevents programs larger than a few Kbytes from linking. We called in to Honeywell and spoke with them about a fix. In the mean time, we began making necessary changes that we already knew about in the device dependent parts of the system.

Work proceeded on the database backup and recovery system for TACCNET. The system was designed to allow the C^2 database files (used by UNIFY) to be transferred via TACCNET to a designated backup site for safekeeping. The files were to be stored in a special "backup" directory on the remote machine and retrieved automatically by sending a special network administrative message.

The user was to be insulated from the actual administrative messages required to back up and recover the files by a set of shell scripts which would be invoked as command files.

The briefing scheduled for Dec. 18 was canceled and rescheduled for a later date. Bill Putnam prepared a high level presentation on TACCNET to supplement the existing technical briefing.

5.5. January 1986

The C functions and shell scripts for the database backup function were installed and tested. Code to allow the distribution and handling of TACCNET broadcast messages was in development.

Broadcast messages are those which are sent from one site to all other sites in the network. This type of message was to be used to send out requests for database recovery from archived messages. It could also be used to distribute network wide routing information.

Shinn reported progress in the development of the screen editor interface for the JINTACCS message entry system. The system used the curses screen handling package. This package is supported on almost all UNIX systems with only minor differences among the different UNIX versions.

The interface as planned would prompt the user in a screen oriented (as opposed to line oriented) manner and use the responses to build a JINTACCS format message. The user would be able to examine the message at any point in the composition process.

The GCOS system conversion proceeded slowly. An apparent bug in the GCOS device handler caused the transmission of the DEL character as a filler in spite of configuration commands that should have disabled this "feature". The low level I/O routines of the IOCONTROL module were rewritten for GCOS while the DEL problem and the C Linker problems were being investigated.

A hard disk failure on the "xenair" PC/AT XENIX system rendered that system inoperative.

The TACCS DISTRIX 2.0 system was obtained and installed. The TACCNET software was ported over to the new system. The port was accomplished without incident, and the system functioned as a regular TACCNET node. We were unable to run the UNIFY dbms system on the TACCS (different cpu). This prevented the TACCS node from running the C^2 database and participating fully in the test scenarios involving database backup and recovery. We were able to use the TACCS node as a backup site for the other nodes, however.

A meeting was held to discuss the desired content of the TACCNET Technical Specification required for Task 3. It was decided that the spec should contain a high level design description, user manual, and an installation guide for the TACCNET system.

A briefing and demonstration were scheduled for February 19.

5.6. February 1986

The PC/AT with the hard disk problem was repaired and reinstalled. The latest version of TACCNET was installed on the two PC/AT systems (xenics and xenair), one ONYX system (sysa), and the TACCS (taccs) DISTRIX system. The new version contained a robust login procedure, database backup system, broadcast messages, and improved error logging.

The JINTACCS message composition tool was nearing completion. It allowed the user two operate in command or editing mode and to toggle between these modes at will. Input syntax checking and a help facility were under development.

A demonstration was presented on Feb. 26. Test scenario 3 was executed successfully. This is described in the next section, Demonstrations and Presentations. A request was made for better console monitoring utilities.

We had some trouble getting customer support for the Honeywell C compiler. The questions were eventually resolved.

5.7. March 1986

Work continued on the Honeywell PASCAL to C conversion. Modification of low level I/O routines was complete and testing began. The next phase involved the conversion of directory and file handling routines. Problems with the port configuration and the modems hampered progress, but were resolved.

The C program linker for GCOS still refused to link moderate to large programs. Help from Honeywell produced a "workaround" solution which required the user to interrupt the compilation and manually edit the linker control file to set certain link parameters before restarting the link process. This allowed us to link the programs, but was a severe inconvenience and was regarded as a temporary fix.

The database backup and recovery functions were completed and tested. Coding began for the incremental database recovery from archives. A scenario was devised to demonstrate the system.

Coding for the JINTACCS message composition tool was completed. The program was debugged and some cosmetic changes were made. The program was incorporated into the TACCNET system as the primary user interface for message creation.

5.8. April 1986

Development proceeded on the message processor routines to recover JINTACCS messages from site archives. These messages were to be re-queued for transmission to a specified site for use in reconstituting that site's C^2 database. Functions for the regular backup and recovery of the C^2 database were installed and validated by testing.

A draft outline for the TACCNET technical specification was prepared.

We decided to obtain a copy of the JINTACCS handbook, which would allow us to put more messages into the system.

We acquired a Kurzweil Voice System machine which capable of voice recognition with a 1000 word vocabulary. We considered ways to incorporate this machine into the research using voice recognition software to build JINTACCS messages.

We discovered that the Honeywell DPS/6 does not allow timeouts during read operations. Some rewriting of the code was required to allow for this. Other problems related to the configuration of ports were recorded. The system required some modification of UNIX based code to handle special timeout, DEL, and BEL situations arising from GCOS limitations on asynchronous reads and writes. These changes were bracketed by "#ifdef" statements in the C code so that they are compiled only as needed.

We became aware of a software product developed by Consultant's Choice, Inc. which purported to be able to read and parse JINTACCS messages. Arrangements were made to investigate the product and evaluate its suitability for use in the CSSCS environment. A presentation by CCI representatives was held at the AIRMICS office. A decision was made to examine the system further and to solicit sources for JINTACCS message processing systems for open evaluation.

5.9. Conclusion

The term of the project was concluded with successful demonstrations of the results of Tasks 1 and 2 and with a presentation at the August 1986 Joint IPR hosted by AIRMICS. These presentations are described below.

6. Demonstrations and Presentations

Two Advanced Experimental Demonstrations (AED) and one In Process Review (IPR) were conducted during the term of the project. In this section we will describe the activities and results presented.

6.1. Advanced Experimental Demonstration 1

Advanced Experimental Demonstration 1 was the culmination of Task 1 of the work statement. This demonstration presented the results of an examination of the issues of maintaining, updating, and restoring a distributed database in the presence of communications failures and node failures in the CSSCS network.

The previously defined TACCNET prototype system was expanded to allow interaction between a prototype Command and Control (C^2) database and the JINTACCS messages being passed by the system. The prototype C^2 database was derived from the set of messages to be used in the demonstration scenarios. The database consisted of a set of relations in a schema defined under the UNIFY relational database management system. The relations were developed by studying the JINTACCS messages and normalizing the data elements from these messages into coherent relations. As there was no information available on the content or usage of the C^2 database it was necessary to define the relations in an ad hoc manner. The resulting prototype database should not be considered as a fieldable system; it is merely a tool for demonstration of certain capabilities. A complete description of the C^2 database is given in the TACCNET technical specification document referenced above.

The TACCNET system was expanded to allow the automated generation of JINTACCS messages from the prototype C^2 database and the automated posting of messages to the database. Incoming JINTACCS messages were scanned and their component data elements extracted and inserted into database relations. Existing database relations were updated with the new information.

The primary focus of this task was to demonstrate the ability to maintain copies of a unit C^2 database at several other sites in the network and to be able to access those copies and any relevant archived messages to reconstruct the C^2 database in the event of a node failure. TACCNET was used to transfer and retrieve the database backup files. Broadcast messages and archive scanning were implemented to allow a node to request retransmission of all messages sent since the last backup was made. A set of shell command files were generated to allow the user to easily make backups and retrieve them from other nodes.

In the AED, a unit database was successfully backed up on two other nodes in the testbed network. The original database was then deleted and recovery was effected from a remote node. Broadcast messages were sent to other nodes resulting in retransmission of JINTACCS messages sent to the failed node after the backup date. The desired messages were automatically recovered from remote system archives and retransmitted to the local system where they were posted to the C^2 database. The resulting database was then shown to be the same as the original database. The backup and recovery were accomplished by the use of three commands on the local system.

Finally, the TACCNET system was expanded to allow local emulation of remote units which have failed. This allowed a unit to replace a failed unit by starting up an emulated node using the database backup from the failed unit. Broadcast messages could be sent to get any subsequent messages for the failed unit and to inform other units that the substitution had taken place. In this manner a remote node could take over the function of a failed local node and continue to answer queries and provide updates during the local node downtime. Upon reactivation, the local node could retrieve the current database from the remote node and resume activity.

6.2. Advanced Experimental Demonstration 2

Task 2 entailed the conversion of TACCNET code on the Honeywell DPS/6 from PASCAL to C. This was completed and the resulting code was shown to have greater functionality then the original. The implementation in C was taken straight from the UNIX systems. The C source code was transferred from the PC/AT running XENIX to the Honeywell GCOS system using the original TACCNET system for file transfer. The C code was then modified to allow for GCOS file system dependencies and the resulting source was compiled. The complete source for the system compiled without error almost immediately, but some bugs required additional work.

A major handicap to progress on this task was presented by the Honeywell C compiler. The compiler was very green and contained a number of bugs and design deficiencies. The problems are described in the Summary of Activities below. The deficiencies were eventually overcome by means of workaround solutions and the problems were reported to Honeywell. It is the opinion of the project staff that the Honeywell C compiler needs some serious work before it is a viable program development tool.

In the end, the TACCNET implementation on the Honeywell was completed and the system was used as a node in the testbed network. Files and message were exchanged between the Honeywell and the other systems in the network. The IOCONTROL, CALLER, and GENMSG programs were implemented on the Honeywell system. This allowed the Honeywell to answer the calls from

other systems and to call other systems and initiate transfers when so commanded by the operator. The QMS program was not implemented on the Honeywell since it is not expected that the DAS3 will require its capabilities.

6.3. In Process Review

In August 1986 a joint IPR for all AIRMICS contractors was hosted by AIRMICS. The TACCNET project group presented four hours of briefings and demonstrations on the project and on the TACCNET system. As a result, TACCNET was installed on several other machines and is being used by some other AIRMICS contractors on their projects.

The visual aids used in the IPR are provided in appendix 3 of this report.

7. TACCNET System Requirements

In this section the hardware and software requirements for the TACCNET software are defined. The system was designed and developed with portability as a major concern and should be easily ported to any machine using the UNIX operating system or one of its derivatives.

7.1. Hardware

The TACCNET software has been installed and tested on ONYX and IBM PC/AT computers. These machines were used to simulate the TACCS system. The system requires at least one 360K floppy disk and a minimum of 10 megabytes of hard disk storage. The minimum memory required to run the system is 512K bytes. The GCOS part of the system is installed on a Honeywell DPS Level 6 minicomputer.

7.2. Operating Systems

All software for the TACCS has been written in the C programming language and developed under the ONYX and XENIX operating systems. Each of these systems is a derivative of the UNIX operating system. The software will run without modification on any UNIX System III machine.

Software for the DAS3 has been written in C and developed under the GCOS Mod 400 operating system using the M4_CC compiler. While there are differences in implementation details between the GCOS and UNIX versions of TACCNET, the functionality at the communications and protocol level is the same. The message processing and network management levels have not been implemented in the GCOS version.

7.3. Communications Equipment

The modems employed in the development and demonstration of the prototype system are D. C. Hayes Smartmodem 1200 standalone modems. These modems are auto-dial, auto-answer, programmable asynchronous 1200 bps devices intended for use with voice grade telephone lines. The TACCNET software uses Smartmodem commands to program the modem and dial the phone, thus requiring the use of the Hayes Smartmodem on all *dialout* lines. Any auto-answer modem could be used for the *dialin* lines.

Each TACCS system must have at least one telephone line for *dialin* use and one line for *dialout* use. More lines may be allocated for each mode if available. Section 4.3 explains the configuration of phone lines for the system. The DAS3 must have one phone line for *dialout* use. The modem on this line must

be a Cermetek Infomate 1200 bps modem. The proper configuration of the Honeywell port is explained in section 4.3.

7.4. Software

The TACCNET prototype system provides automated database operations for a sample set of three JINTACCS messages. These messages may be generated from information stored in the prototype C^2 database or they may be used to update information in the database. These functions are provided by the database *server* program and are dependent on the presence of the UNIFY relational database management system. The dbms must be configured to include the relations defined for the C^2 database prototype which is described in section 4.2.4.

8. Recommendations and Conclusions

In this report we have summarized the issues encountered and work done by the project team during the development of the prototype TACCNET system for the CSSCS. Some of these considerations are general and will apply to any information and communications systems developed for use in the CSSCS environment. Others are dependent on the specific functional requirements of the proposed system. It is critical to define and describe the complete set of functions to be performed by the system so that such design considerations may be discussed and resolved before the system is built.

We also believe that the CSSCS information requirements are not yet sufficiently well defined for development of the CSSCS communications systems to begin. There seems to be little or no information available on the content and intended usage of the Command and Control Database in the CSS units. It is difficult to project communications requirements and to develop system interfaces without a good analysis of the nature of the C2 database and its role in the CSSCS.

One theme which has been present throughout our work and in this report is the importance of portability. At this time it appears likely that the CSSCS will run on a small computer using UNIX or one of its derivatives. The specific hardware and UNIX version are not yet defined. Since one of the prime features of UNIX and the C language is portability, it would be a serious mistake to write hardware or version dependent code in the development of CSSCS software. With a small amount of extra work one can develop code which is easily ported to any UNIX-derived system, regardless of the hardware chosen. The TACCNET system has been run successfully on seven different versions† of UNIX and on five different machines‡.

A number of issues remain for further investigation in this area. These are summarized in Appendix 1, "Areas for Further Investigation".

A separate document, " Considerations in the Design and Development of a Combat Service Support Computer System", has been provided as a description of the TACCNET prototype developed as part of the CSSCS AED program. This system is described to illustrate the main issues in CSSCS communications and is not to be considered as a fieldable system. It is a starting point for further development. It is not expected that the reader of this report will be

---

† The versions are: AT&T System Vr2; AT&T System III; IBM XENIX 1.0; SCO XENIX V; ONYX Onix V; PC/IX; DISTRIX 2.0 (and 1.0).

‡ The machines are: IBM PC/XT; IBM PC/AT; Burroughs B26; ONYX; AT&T 3b2

fully able to understand and operate the TACCNET system. It will probably be necessary to study the system source code in order to fully understand the system. The TACCNET development team at Georgia Tech will be happy to answer any questions and provide any assistance necessary.

Appendix 2 to this report contains the demonstration scenarios used in the Advanced Experimental Demonstrations presented during the project. Copies and descriptions of the JINTACCS messages used in the AEDs are provided.

Appendix 3 contains copies of viewgraphs used in a presentation about TACCNET. These will be helpful in understanding the system.

Appendix 4 contains high level data flow diagrams for the major TACCNET system components. These will aid in understanding the interactions of the TACCNET subsystems.

9. References

[1] ACCS-A3-400-004 (Interface Specification For) Maneuver Control Element Interface With Combat Service Support Control Element, August 1984.

[2] ACCS-A3-400-005 (Interface Specification For) Air Defense Control Element Interface With Combat Service Support Control Element, 9 November 1984.

[3] ACCS-A3-400-008 (Interface Specification For) Combat Service Support Control Element Interface With Intelligence/Electronic Warfare Control Element, 9 November 1984.

[4] ACCS-A3-400-009 (Interface Specification For) Combat Service Support Control Element Interface With Fire Support Control Element, 9 November 1984.

[5] ACCS-A3-500-003 Army Command and Control Systems Character Oriented Message Format Standards, June 84.

[6] ACCS-A3-500-003 Army Command and Control Systems Character Oriented Message Format Standards, Supplement 1, June 84.

[7] JINTACCS Technical Interface Design Plan, Volume VIII, Combat Service Support, October 1984.

[8] JINTACCS Technical Interface Design Plan, Volume VIII, Combat Service Support, Appendix E, COM Text Formatting Rules, June 1984.

[9] "Human Factors In The Display Of JINTACCS Messages", Tech. Rep. #USAISEC-RARA-85-2, D. Sharpe and A. Badre, 9 October 1985.

[10] "An Analysis of the Data Processing Requirements of CSSCS", Tech. Rep. #USAISEC-ASBG-85-1, M. Graham, 20 September 1985.

[11] CSSCS C^2 Information Requirements, 4 October 1982.

[12] "Of JINTACCS and JABBERWOCKS", Maj. J. Morrison and Maj. R. Case, *Signal*, vol. 38, no. 3, pp 55-58, November, 1983.

[13] "JINTACCS: getting the message across", Maj. A. Schenk, *Army Communicator*, Winter, 1986, pp 12-20.

[14] "How Secure is Secure?", G. Grossman, *UNIX Review*, August, 1986, pp 50-63.

[15] CSSCS Advanced Experimental Demonstrations, Final Technical Report for 1983-84, A. Jensen, W. Putnam, S. Goldberg, Georgia Institute of Technology, July, 1984.

[16] CSSCS Advanced Experimental Demonstrations, Final Technical Report for 1984-85, A. Jensen, W. Putnam, S. Goldberg, Georgia Institute of Technology, December, 1985.

[17] Considerations in the Design and Development of a Combat Service Support Computer System, A. Jensen, W. Putnam, S. Goldberg, Georgia Institute of Technology, December, 1986. Technical specification and user's manual for the TACCNET system.

Appendix 1 - Areas for Further Investigation

Development of a JINTACCS Message Parser

Examine and develop a working version of a JINTACCS message grammar using a well defined subset of the ACCS COMS; Apply UNIX tools (eg yacc, lex) to develop a parser for messages; incorporate parser into a system which would use knowledge about the messages and their users (expert system) to aid in construction, routing, and processing of JINTACCS messages.

Analysis of DSS requirements and Applications

Identify users, sources, applications of information available in C^2 database in the context of a DSS based on that database. Describe DSS<->C^2DB interface.

C^2 Database Analysis

Interview potential users and operators of TACCS systems (as well as other affected individuals), digest available documentation on C^2DB, examine JNITACCS message set, and develop a basic description of the C^2 database: its users, applications, contents, and interfaces.

TACCNET Security Analysis

Study and define security considerations in the TACCNET environment. Areas include: software protection; data security; access levels (software, hardware, and information); data transfer; encryption (information storage and transfer). Apply findings to current TACCNET software system.

Network Tracking

Interactive monitoring of network operations; compilation of statistics on message flow & usage, node interactions. Usage of such data in automatic configuration of network and individual systems. Examination of sources/destinations of messages, development of tools for network analysis and management.

Analysis of ACCS COM Usage

Identify stimulus/response patterns in usage of JINTACCS messages. Analyze and describe processing requirements and operations for automated handling of messages according to stimulus/response patterns. Develop scenarios for testing of JINTACCS message systems.

Technology Transfer and Integration

Integrate multiple related tasks and projects in the Command & Control and/or JINTACCS message processing areas into coherent prototype and demonstration systems. Develop and evaluate hardware, software, and documentation necessary for interfaces between existing and proposed systems. Support AIRMICS internal development teams in transfer of prototype systems to target environment and in development of operational systems. Application of Expert Systems, Voice Recognition, Human Factors Engineering, Intelligent Data Flow Analysis, and other advanced technologies to environment and operations of C^2 and JINTACCS messages.

STAMMIS extraction and the CSSCS interface

The extraction of information from STAMMIS for use in the CSSCS should be examined. Interfaces for information exchange should be developed and standardized. Transfer of information from a STAMMIS into the CSSCS prototype system via TACCNET could be demonstrated.

These scenarios are to be used in conjunction with Task 3 of Delivery Order 0018 for Contract DAAK70-79-D-0087 with the Georgia Institute of Technology. These scenarios were developed jointly by AIRMICS and the Georgia Institute of Technology. Attending the meeting were the following:

Professor A. P. Jensen, GIT
Mr. Bill Putnam, GIT
Maj(P) Thomas Rogers
Maj David R. Forinash
Maj Terry Hilderbrand
Cpt(P) Larry Frank
Dr. Jerry McCoyd

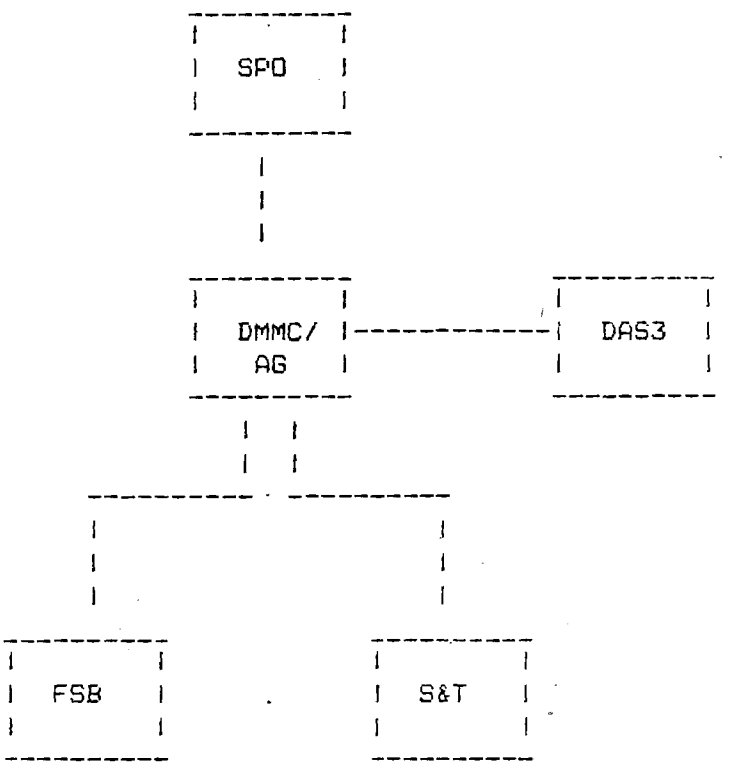
Description of task: The extensive data requirements of a command and control system will require some type of data base to store the information until it is needed for decision making or message creation. This data base must be capable of being posted automatically, answering automated queries, and responding to on-line queries. This task will require the analysis of the requirements for a C2 data base, establishment of the data base on the two TACCS computers of the CSS test-bed, and use of the data base to generate the data elements required for the messages passed in tasks 1 and 2. Messages must be received from the DAS3 system and both received from and passed to the other TACCS system. Any courtesy copies needing posting due to the TACCS being bypassed while it was non-operational shall be automatically posted to the data base. Data elements should be stored independent of message format allowing update of individual items. The demonstration performed as part of task 2 shall be rerun using these data bases as the source and destination of the messages' data elements.

Modification to task: Due to the delivery schedule of the TACCS computer systems, the two TACCS computers will be unavailable. Two Onyx C8002 computer systems will be substituted for the specified TACCS systems. The specific equipment for the demonstration will be as follows:

1 Honeywell DPS6 Model 54 using GCOS400 to simulate the DAS3
3 Onyx C8002 using UNIX System III to simulate the TACCS
1 IBM PC/AT using Xenix to simulate the TACCS

Equipment Configuration: The following diagram shows the normal network routing that will be used in all scenarios. Communications links between systems will be over dial-up telephone lines without manual intervention. All connections will be computer initiated and terminated.

Test-Bed Configuration



Messages: The messages to be used in this demonstration are from the ACCS COM Message Standards. They are in JINTACCS format. The chosen messages are required to be passed at the force level (between combat service support and the other four nodes) as well as internally (as identified by the information requirements document supplied by LOGC). The force level interface will be simulated since the interrim system has a remote interface. This simulation will consist of a screen display of the message in JINTACCS format (complete with slashes) that may be displayed upon demand as well as an on demand "operator readable" version that displays both the message data and identifying information (such as labels).

Message Features: (see incl 1 for message formats) The examples given are illustrative of the different types of features in the messages and are in no way exhaustive of all the features in the messages, i.e. all features are not listed in all messages, rather, new features are listed that differ from previous examples.

S006 Casualty Information Report

The set KUNITCAS demonstrates the use of field descriptors. These descriptors are not required for machine interpretation of the data since the data is position dependent, however the descriptors do make the data human readable.

The sets KDTGRPTN, KUNITCAS, and 5KCASLTY are mandatory.

The set 5KMOSCAS is conditional. The condition that indicates whether the set is present is determined by the user and therefore the option must be given for the user to include the data or not.

The sets 5KCASLTY and 5KMOSCAS are columnar sets. This means that the fields in the set are repeatable even though the code does not indicate them as such. The first line will be the column headings. Subsequent lines will be the actual data as given by the fields in the set.

The set 5KCASLTY demonstrates a set where an arithmetic computation must be made to complete the set. The field "COUNT OF PERSONNEL" is the total of the preceding four fields. This same type of behavior is exhibited by sets 5KMOSCAS and KUNITCAS.

This message has some problems with machine interpretation. First, the set 5KMOSCAS has alternative fields without field descriptors. The fields have the same formats (5 AN) so there is no way to determine the actual MOS or SSI usage of the field. The interpretation we are giving to this is that MOS pertains only to enlisted while SSI pertains to warrant and commissioned officers. In this interpretation, the actual usage is immaterial and a unique identifier results and the information may be parsed without problem. Actual usage of these fields may be different and problems could be present that are not addressed by this demonstration. Another problem is the fact that columnar sets 5KCASLTY and 5KMOSCAS are related but there is no line-by-line key provided. A usage convention is needed to be able to cross check data values for validity. In this demonstration the data will be merely displayed and no attempt to validate the values in sets will be attempted.

S026 POL Locations

The set 3KPOLLOC demonstrates alternative fields without field descriptors. This presents a problem in that field location has 5 alternative representations. Four of these have unique lengths that identify the format that is used. The fifth, UNIT LOCATION NAME, has a specification of 1-20 ANBS and therefore could be confused with any of the other four formats. In this demonstration, the only valid location that will be used will be the LOCATION, UTM 100-METER.

Sets 3KPOLLOC and 3KCLTHRE show columnar sets where the data entry fields in one set key the data back to the data in the previous set.

The AMPN set (page 2) in this message is conditional on the judgement of the operator that the set must be included to complete the

data.

S034 Supply Shortages

The set AMPN is conditional upon data entered in the previous set and therefore should be prompted for in cases where it is necessary to include the set.

Data Bases: Each TACCS device (look-alike) will incorporate a command and control data base. At this time there is no mandatory requirement for replication of data bases. Each data base will contain the information required by that node to conduct business (to include contingency plans such as rerouting messages).

Scenarios: There are three scenarios. Two scenarios are logistics oriented while the third is personnel oriented. The three scenarios could in fact be performed as one integrated scenario, however, breaking them into individual scenarios allows greater insight into the processing taking place.

REFERENCES:

ACCS-A3-400-004	(Interface Specification For) Manuever Control Element Interface with Combat Service Support Control Element	AUG 84
ACCS-A3-400-005 (first draft)	(Interface Specification For) Air Defense Control Element Interface with Combat Service Support Control Element	09 NOV 84
ACCS-A3-400-008 (first draft)	(Interface Specification For) Combat Service Support Control Element Interface with Intelligence/Electronic Warfare Control Element	09 NOV 84
ACCS-A3-400-009 (first draft)	(Interface Specification For) Combat Service Support Control Element Interface with Fire Support Control Element	09 NOV 84
ACCS-A3-500-003	COM Message Format Standards	JUNE 84
ACCS-A3-500-003	COM Message Format Standards, Supplement 1	JUNE 84

JINTACCS Technical Interface Design (Final Edition), Appendix E, Character Oriented Message Text Rules — JUNE 84

JINTACCS Message Standards, Catalog of Keyword Data Sets, Parts 1,2, Reissue 5 — DEC 84

JINTACCS Message Element Dictionary (MED) — DEC 84

Parts 1, 2, 3, 4 - Reissue 5

Appendix A, Data Field Identifier Standards, Reissue 1

Appendix E - Character-Oriented Message Text Formatting Rules, Reissue 1 — DEC 84

SCENARIO I

Scenario I uses the ACCS message S006 Casualty Information Report (CSS C2 Information Unit 130A1 PERSITREP). The force level interface documentation shows this message being transmitted to CSS by ADA, IEW, and FS. There is no requirement from MCS. The CSS C2 Information Requirements shows this message as being sent to G1, AG, and P&A by the units. These two usages are compatible.

1. The messages will originate at the DISCOM SPO. This manual origination is consistent with the interim system that uses a remote interface. Data entry is initiated through the keyboard of the TACCS. Data entry need not be in JINTACCS format.

2. The SPO places the data in its internal data base.

3. The SPO sends the message, in JINTACCS format, to the AG.

4. The AG places the data in its internal data base.

5. An on-line manual query verifies the arrival and storage of the data.

SCENARIO II

Scenario II uses the message S034 Supply Shortages. This message may be used in two ways. One way is to report status of selected supplies and the actions being taken to procure them. The other way is to request information on the status of selected supplies. The interface specifications show that this message is used across the interfaces to FS and IEW in a bidirectional mode and transmitted only to ADA. MCS has no requirement for this message. The internal usage of this message (081A3) shows this message flowing from functional organization to FSB, DMMC, CORPS SPT GP, and COSCOM. In this scenario, the information on supply status is resident on the DAS3. The DAS3 periodically would update the command and control data base on the status of previously identified supply items. The command and control data base is resident on the DMMC TACCS. A message requesting supply status will be received by the SPO. This request for information must then be serviced.

1. Manually initiate on the DAS3 a message giving the status of selected supplies for various units.

2. Send the message to the DMMC.

3. DMMC places the information in the C2 data base.

4. SPO receives S034 Supply Shortage Request from Fire Support. This is simulated through the use of a screen display and data entry.

5. SPO forwards request to DMMC.

6. DMMC receives S034 Supply Shortage from SPO.

7. Human operator reviews S034 and makes appropriate queries of C2 data base.

8. Operator uses screen and keyboard to create return message with the desired status.

9. DMMC sends S034 message to SPO.

10. SPO, on manual request, displays message to simulate remote interface. Message should be displayed as both JINTACCS format (with all the slashes) and as human readable (a friendlier format than JINTACCS).

SCENARIO III

This scenario uses the force level message S026 POL location (CSS C2 Information Requirement 061A1). The force level interface specification shows this as a message transmitted by CSS to all four nodes. The CSS C2 Information Requirements Document shows this as a message sent every four hours by supply companies and the POL battalion to the FSB , DMMC, and CMMC. Note: at this time it does not appear that there is a way to delete a POL location by use of a message. The scenario will assume that all POL locations, once active, will remain active. Quantities transmitted are current on-hand quantities and not changes to previous messages. Messages may be sent by 1) "pushing a button" that causes the message to be automatically formatted and sent, 2) the clock says "time to send a message" and the message is formatted and sent without operator intervention, 3) manually entering message data into a screen template, or 4) automatic response to an automated query (again without operator intervention).

1. Initially the network consists of the SPO, the DMMC, and the S&T Battalion. The FSB reaches its location and sends its message to the network to enter it into the appropriate routing tables and distribution lists.

2. S&T Battalion uses the "push a button" method to send the POL Location message to the DMMC. Data has been entered manually through on-line data base update. Location sent is for the division main fuel distribution point.

3. FSB uses screen template to format and send location of POL point to DMMC. Data is also stored in data base. Location sent is for forward fuel distribution point in the brigade area.

4. Messages are received by DMMC.

5. DMMC posts data to its data base.

6. DMMC uses clock method to formulate a consolidated list of POL locations and send to SPO and FSB.

7. SPO receives message from DMMC and posts it to its data base.

8. SPO simulates forwarding of message to MCS, IEW, ADA, and FS by displaying message on screen upon demand by operator. Message is displayed in JINTACCS format as one option and as human readable as a second option.

9. FSB receives the message from DMMC and, upon demand, displays it upon the screen.

10. FSB enters the POL location data in its data base.

11. Change the quantities of POL on hand at the division main fuel distribution point in the S&T Battalion data base using on-line update.

12. DMMC formulate on screen a request for update of POL locations using the POL Location message in a request mode.

13. Send the request message to the S&T Battalion.

14. S&T Battalion responds to DMMC automated query by automatically sending message with new locations.

15. DMMC receives the message from the S&T Battalion and changes its data base.

16. Manual query at DMMC verifies the change.

17. DMMC node goes down.

18. Change manually the POL location data at the S&T Battalion and the FSB.

19. S&T and FSB formulate their messages using "push a button" method and attempt to send to DMMC.

20. After determining that DMMC is down, FSB and S&T send their messages to SPO as an automatic reroute.

21. SPO receives messages.

22. SPO updates its data base.

23. SPO displays data on screen in response to on-line query by operator.

24. SPO determines that DMMC was bypassed and sends consolidated POL Location message to FSB.

25. FSB posts data to data base upon receipt.

26. FSB displays data on screen upon demand (on-line query).

27. DMMC returns to operational status.

28. FSB and S&T send courtesy copy of message to DMMC.

29. DMMC posts its data base automatically using new data.

30. On-line query verifies new data at DMMC.

INCLOSURE 1

JINTACCS MESSAGE FORMATS

MESSAGE NUMBER: S006

TITLE: Casualty Information Report (CASSTATS)

GENERAL INSTRUCTIONS

This message reports casualty information in four categories broken down by unit total, military personnel class, and specialty skill or MOS. The four categories are KIA, WIA, MIA, and non-battle casualties.

The sets, EXER through NARR, are prepared in accordance with the message instructions for the initial main text sets.

SPECIAL INSTRUCTIONS

Set Identifier: KDTGRPTN [M].

Field 1, As of Date-Time [M]. Enter the as of date-time (day, hour, and minute) for the time of the report (DFI #C914, DUI A25).

Field 2, Report Serial Number [M]. Enter the appropriate report serial number (DFI #E487, DUI 048).

Set Identifier: KUNITCAS [M]. This set is used to report a unit's total number of casualties in four categories.

Field 1, Unit Identification [M]. Enter the unit identification or the transliterated unit name by unit number, organization type, and echelon level for the unit reporting casualties (DFI #E987, DUI 005) or (DFI #C095, DUI 001).

Field 2, Actual KIA [M]. Enter the field descriptor followed by the total number of unit personnel killed in action (DFI #E959, DUI 006).

Field 3, Actual WIA [M]. Enter the field descriptor followed by the total number of unit personnel wounded in action (DFI #E959, DUI 013).

Field 4, Actual MIA [M]. Enter the field descriptor followed by the total number of unit personnel missing in action (DFI #E959, DUI 014).

Field 5, Actual Non-Battle Casualties [M]. Enter the field descriptor followed by the total number of non-battle casualties for the unit (DFI #E959, DUI A04).

Field 6, Count of Personnel [M]. Enter the field descriptor followed by the total number of casualties from the entries in fields 2, 3, 4, and 5 for the unit entered in field 1 (DFI #E959, DUI 001).

MESSAGE NUMBER: S006

TITLE: Casualty Information Report (CASSTATS)

SPECIAL INSTRUCTIONS (Continued)

Set Identifier: 5KCASLTY [M]. This set is used to report unit casualties in four categories by military personnel class (i.e., officer, warrant officer, and enlisted) as well as the total number of casualties for each class.

Field 1, Military Personnel Class [M]. Enter the character code for the military personnel class to be reported (DFI #E168, DUI 001).

Field 2, Actual KIA [M]. Enter the number of personnel killed in action for the personnel class entered in field 1 (DFI #E959, DUI 006).

Field 3, Actual WIA [M]. Enter the number of personnel wounded in action for the personnel class entered in field 1 (DFI #E959, DUI 013).

Field 4, Actual MIA [M]. Enter the number of personnel missing in action for the personnel class entered in field 1 (DFI #E959, DUI 014).

Field 5, Actual Non-Battle Casualties [M]. Enter the number of non-battle casualties for the personnel class entered in field 1 (DFI #E959, DUI A01).

Field 6, Count of Personnel [M]. Enter the total number of casualties from the entries in fields 2, 3, 4, and 5 for the personnel class entered in field 1 (DFI #E959, DUI 001).

Set Identifier: 5KMOSCAS [C]. This set is mandatory if the casualties reported are to be further classified by specialty skill identifier/military occupational skill code.

Field 1, Specialty Skill Identifier: [M]. Enter the appropriate specialty skill identifier (DFI #E4027, DUI A02).

OR

Military Occupational Specialty. Enter the desired military occupational specialty code (DFI #E4027, DUI A01).

Field 2, Actual KIA [M]. Enter the number of personnel killed in action for the code entered in field 1 (DFI #E959, DUI 006).

MESSAGE NUMBER: S006

TITLE: Casualty Information Report (CASSTATS)

SPECIAL INSTRUCTIONS (Continued)

Set Identifier: 5KMOSCAS [C]. (Continued)

Field 3, Actual WIA [M]. Enter the number of personnel wounded in action for the code entered in field 1 (DFI #E959, DUI 013).

Field 4, Actual MIA [M]. Enter the number of personnel missing in action for the code entered in field 1 (DFI #E959, DUI 014).

Field 5, Actual Non-Battle Casualties [M]. Enter the number of non-battle casualties for the code entered in field 1 (DFI #E959, DUI A01).

Field 6, Count of Personnel [M]. Enter the total number of casualties from the entries in fields 2, 3, 4, and 5 for the code entered in field 1 (DFI #E959, DUI 001).

Set Identifier: RMKS [O]. This set is used for any additional required information.

Set Identifier: DWNGRADE [C]. This set is mandatory when the message is classified.

Field 1, Downgrading and Declassification Markings [M]. Enter the appropriate downgrading and declassification markings if the message is classified (DFI #E679, DUI 001).

MESSAGE NUMBER: S006

TITLE: CASUALTY INFORMATION REPORT (CASSTATS)

PURPOSE: TO REPORT CASUALTY INFORMATION IN FOUR CATEGORIES BROKEN DOWN BY UNIT TOTAL, MILITARY PERSONNEL CLASS, AND SPECIALTY SKILL OR MOS. THE FOUR CATEGORIES ARE KIA, WIA, MIA, AND NON-BATTLE CASUALTIES.

SET IDENT	CAT	FIELD NO	MANDATORY ENTRY/ FLD DESC/COL HEADER	FIELD NAME	START COL J	NO/TYPE	DFI NO	DUI NO
EXER	C							
	M	1		EXERCISE NICKNAME		1- 56 ANBS	E 335	001
	O	2		EXERCISE MESSAGE ADDITIONAL IDENTIFIER		1- 16 AB	E 335	002
OPER	C							
	M	1		OPERATION CODEWORD		1- 32 AB	E 336	001
	O	2		PLAN ORIGINATOR AND NUMBER		3- 23 ANS	E 925	001
	O	3		OPTION NICKNAME		1- 23 ANBS	E 585	001
	O	4		SECONDARY OPTION NICKNAME		1- 23 ANBS	E 585	002
MSGID	M							
	M	1	CASSTATS	MESSAGE TYPE		1- 20 ANBS	E 050	001
	M	2		ORIGINATOR		1- 20 ANBS	E 146	001
	O	3		MESSAGE SERIAL NUMBER		7 N	E 147	005
				REPORT SERIAL		4- 5 ANS	E 147	006
	O	4		MONTH		3 A	E 580	001
	O	5		QUALIFIER		3 A	E 568	001
	O	6		SERIAL NUMBER OF QUALIFIER		1- 3 N	E 467	017
REF	O,R							
	M	1		SERIAL LETTER		1 A	E 636	002
	M	2		MESSAGE TYPE		1- 20 ANBS	E 050	001
				COMMUNICATION TYPE		3 A	E 646	001
	M	3		ORIGINATOR		1- 20 ANBS	E 146	001
	M	4		DATE OF REFERENCE, YEAR-MONTH-DAY		6 N	C 075	010
				DAY-TIME OF REFERENCE		7 AN	C 143	005
				DATE-TIME GROUP		13 ANB	C 647	001
				DATE OF REFERENCE, DAY-MONTH-YEAR		6 N	C 648	002
				DATE OF REFERENCE, DAY-ALPHAMONTH-YEAR		7 AN	C 649	002
	O	5		MESSAGE SERIAL NUMBER		7 N	E 147	005
				REPORT SERIAL		4- 5 ANS	E 147	006
	O	6		SPECIAL NOTATION		5 A	E 1042	001
	O,R	7		NASIS CODE		3 A	E 332	001
AMPN	C							
NARR	C							

MESSAGE NUMBER: S006

TITLE: CASUALTY INFORMATION REPORT (CASSTATS)

PURPOSE: TO REPORT CASUALTY INFORMATION IN FOUR CATEGORIES BROKEN DOWN BY UNIT TOTAL, MILITARY PERSONNEL CLASS, AND SPECIALTY SKILL OR MOS. THE FOUR CATEGORIES ARE KIA, WIA, MIA, AND NON-BATTLE CASUALTIES.

SET IDENT	CAT	FIELD NO	MANDATORY ENTRY/ FLD DESC/COL HEADER	FIELD NAME	START COL J	NO/TYPE	DFI NO	DUI NO
EXER	C							
	M	1		EXERCISE NICKNAME		1- 56 ANBS	E 335	001
	O	2		EXERCISE MESSAGE ADDITIONAL IDENTIFIER		1- 16 AB	E 335	002
OPER	C							
	M	1		OPERATION CODEWORD		1- 32 AB	E 336	001
	O	2		PLAN ORIGINATOR AND NUMBER		3- 23 ANS	E 925	001
	O	3		OPTION NICKNAME		1- 23 ANBS	E 585	001
	O	4		SECONDARY OPTION NICKNAME		1- 23 ANBS	E 585	002
MSGID	M							
	M	1	CASSTATS	MESSAGE TYPE		1- 20 ANBS	E 050	001
	M	2		ORIGINATOR		1- 20 ANBS	E 146	001
	O	3		MESSAGE SERIAL NUMBER		7 N	E 147	005
				REPORT SERIAL		4- 5 ANS	E 147	006
	O	4		MONTH		3 A	E 580	001
	O	5		QUALIFIER		3 A	E 568	001
	O	6		SERIAL NUMBER OF QUALIFIER		1- 3 N	E 467	017
REF	O,R							
	M	1		SERIAL LETTER		1 A	E 636	002
	M	2		MESSAGE TYPE		1- 20 ANBS	E 050	001
				COMMUNICATION TYPE		3 A	E 646	001
	M	3		ORIGINATOR		1- 20 ANBS	E 146	001
	M	4		DATE OF REFERENCE, YEAR-MONTH-DAY		6 N	C 075	010
				DAY-TIME OF REFERENCE		7 AN	C 143	005
				DATE-TIME GROUP		13 ANB	C 647	001
				DATE OF REFERENCE, DAY-MONTH-YEAR		6 N	C 648	002
				DATE OF REFERENCE, DAY-ALPHAMONTH-YEAR		7 AN	C 649	002
	O	5		MESSAGE SERIAL NUMBER		7 N	E 147	005
				REPORT SERIAL		4- 5 ANS	E 147	006
	O	6		SPECIAL NOTATION		5 A	E 1042	001
	O,R	7		NASIS CODE		3 A	E 332	001
AMPN	C							
NARR	C							

MESSAGE MAP

MESSAGE NUMBER: S006

TITLE: CASUALTY INFORMATION REPORT (CASSTATS)

```
1234567890123456789012345678901234567890123456789012345678901234567890123456789

EXER/XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
     1
/XXXXXXXXXXXXXXXA//
 2

OPER/XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/XXXXXXXXXXXXXXXXXXXXXX
     1                                  2
/XXXXXXXXXXXXXXXXXXXXXXXX/XXXXXXXXXXXXXXXXXXXXXXXXXX//
 3                        4

MSGID/CASSTATS/XXXXXXXXXXXXXXXXXXX/NNNNNNN/AAA/AAA/NNN//
      1        2                   3       4   5   6
                                   NNNNNNN
                                   AXNXX

REF/A/XXXXXXXXXXXXXXXXXXX/XXXXXXXXXXXXXXXXXXX/NNNNNNABAAANN/NNNNNNN
    1 2                   3                    4              5
      XXXXXXXXXXXXXXXXXXX                      NNNNNN         NNNNNNN
      AAA                                      NNNNNNA        AXNXX
                                               NNNNNNABAAANN
                                               NNNNNN
                                               NNAAANN

/AAAAA/AAA//
 6     7

AMPN/

NARR/
```

MESSAGE CONTENT

MESSAGE NUMBER: S006

TITLE: CASUALTY INFORMATION REPORT (CASSTATS)

SET IDENT	CAT	FIELD NO	MANDATORY ENTRY/ FLD DESC/COL HEADER	FIELD NAME	START COL J	NO/TYPE	OFI NO	OUI NO
KDTGRPTN	M							
	M	1		AS OF DATE-TIME		6 N	C 914	A25
	M	2		REPORT SERIAL NUMBER		6 AN	E 487	048
KUNITCAS	M							
	M	1		UNIT IDENTIFICATION		1- 24 ANBS	E 987	005
				UNIT IDENTIFICATION		6- 21 ANS	C 095	001
	M	2	ACTKIA:	ACTUAL KIA		1- 4 AN	E 959	006
	M	3	ACTWIA:	ACTUAL WIA		1- 4 AN	E 959	013
	M	4	ACTMIA:	ACTUAL MIA		1- 4 AN	E 959	014
	M	5	ACTNBC:	ACTUAL NON-BATTLE CASUALTIES		1- 4 AN	E 959	A01
	M	6	CTPERS:	COUNT OF PERSONNEL		1- 4 AN	E 959	001
SKCASLTY	M							
	M	1	MILPERCL	MILITARY PERSONNEL CLASS	2L	1 A	E 168	001
	M	2	ACTKIA	ACTUAL KIA	11R	1- 4 AN	E 959	006
	M	3	ACTWIA	ACTUAL WIA	18R	1- 4 AN	E 959	013
	M	4	ACTMIA	ACTUAL MIA	25R	1- 4 AN	E 959	014
	M	5	ACTNBC	ACTUAL NON-BATTLE CASUALTIES	32R	1- 4 AN	E 959	A01
	M	6	CTPERS	COUNT OF PERSONNEL	39R	1- 4 AN	E 959	001
SKMOSCAS	C							
	M	1	SC-MOS	SPECIALTY SKILL IDENTIFIER	2L	5 AN	E 4027	A02
			SC-MOS	MILITARY OCCUPATIONAL SPECIALTY	2L	5 AN	E 4027	A01
	M	2	ACTKIA	ACTUAL KIA	11R	1- 4 AN	E 959	006
	M	3	ACTWIA	ACTUAL WIA	18R	1- 4 AN	E 959	013
	M	4	ACTMIA	ACTUAL MIA	25R	1- 4 AN	E 959	014
	M	5	ACTNBC	ACTUAL NON-BATTLE CASUALTIES	32R	1- 4 AN	E 959	A04
	M	6	CTPERS	COUNT OF PERSONNEL	39R	1- 4 AN	E 959	001
RMKS	O							
DWNGRADE	C							
	M	1		DOWNGRADING AND DECLASSIFICATION MARKINGS		1- 25 ANBS	E 679	DD1

MESSAGE MAP

MESSAGE NUMBER: S006

TITLE: CASUALTY INFORMATION REPORT (CASSTATS)

```
1234567890123456789012345678901234567890123456789012345678901234567890123456789

EXER/XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
     1
/XXXXXXXXXXXXXXXA//
 2

OPER/XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/XXXXXXXXXXXXXXXXXXXXXX
     1                                  2
/XXXXXXXXXXXXXXXXXXXXXXXX/XXXXXXXXXXXXXXXXXXXXXXXXXX//
 3                        4

MSGID/CASSTATS/XXXXXXXXXXXXXXXXXXX/NNNNNNN/AAA/AAA/NNN//
      1        2                   3       4   5   6
                                   NNNNNNN
                                   AXNXX

REF/A/XXXXXXXXXXXXXXXXXXX/XXXXXXXXXXXXXXXXXXX/NNNNNNABAAANN/NNNNNNN
    1 2                   3                    4              5
      XXXXXXXXXXXXXXXXXXX                      NNNNNN         NNNNNNN
      AAA                                      NNNNNNA        AXNXX
                                               NNNNNNABAAANN
                                               NNNNNN
                                               NNAAANN

/AAAAA/AAA//
 6     7

AMPN/

NARR/
```

MESSAGE CONTENT

MESSAGE NUMBER: S006

TITLE: CASUALTY INFORMATION REPORT (CASSTATS)

SET IDENT	CAT	FIELD NO	MANDATORY ENTRY/ FLD DESC/COL HEADER	FIELD NAME	START COL J	NO/TYPE	OFI NO	OUI NO
KDTGRPTN	M							
	M	1		AS OF DATE-TIME		6 N	C 914	A25
	M	2		REPORT SERIAL NUMBER		6 AN	E 487	048
KUNITCAS	M							
	M	1		UNIT IDENTIFICATION		1- 24 ANBS	E 987	005
				UNIT IDENTIFICATION		6- 21 ANS	C 095	001
	M	2	ACTKIA:	ACTUAL KIA		1- 4 AN	E 959	006
	M	3	ACTWIA:	ACTUAL WIA		1- 4 AN	E 959	013
	M	4	ACTMIA:	ACTUAL MIA		1- 4 AN	E 959	014
	M	5	ACTNBC:	ACTUAL NON-BATTLE CASUALTIES		1- 4 AN	E 959	A01
	M	6	CTPERS:	COUNT OF PERSONNEL		1- 4 AN	E 959	001
SKCASLTY	M							
	M	1	MILPERCL	MILITARY PERSONNEL CLASS	2L	1 A	E 168	001
	M	2	ACTKIA	ACTUAL KIA	11R	1- 4 AN	E 959	006
	M	3	ACTWIA	ACTUAL WIA	18R	1- 4 AN	E 959	013
	M	4	ACTMIA	ACTUAL MIA	25R	1- 4 AN	E 959	014
	M	5	ACTNBC	ACTUAL NON-BATTLE CASUALTIES	32R	1- 4 AN	E 959	A01
	M	6	CTPERS	COUNT OF PERSONNEL	39R	1- 4 AN	E 959	001
SKMOSCAS	C							
	M	1	SC-MOS	SPECIALTY SKILL IDENTIFIER	2L	5 AN	E 4027	A02
			SC-MOS	MILITARY OCCUPATIONAL SPECIALTY	2L	5 AN	E 4027	A01
	M	2	ACTKIA	ACTUAL KIA	11R	1- 4 AN	E 959	006
	M	3	ACTWIA	ACTUAL WIA	18R	1- 4 AN	E 959	013
	M	4	ACTMIA	ACTUAL MIA	25R	1- 4 AN	E 959	014
	M	5	ACTNBC	ACTUAL NON-BATTLE CASUALTIES	32R	1- 4 AN	E 959	A04
	M	6	CTPERS	COUNT OF PERSONNEL	39R	1- 4 AN	E 959	001
RMKS	O							
DWNGRADE	C							
	M	1		DOWNGRADING AND DECLASSIFICATION MARKINGS		1- 25 ANBS	E 679	DD1

MESSAGE MAP

MESSAGE NUMBER: S006

TITLE: CASUALTY INFORMATION REPORT

```
1234567890123456789012345678901234567890123456789012345678901234567890123456789

KDTGRPTN/NNNNNN/AAXNNN//
         1      2

KUNITCAS/XXXXXXXXXXXXXXXXXXXXXXXX/ACTKIA:NNNX/ACTWIA:NNNX/ACTMIA:NNNX
         1                          2           3           4
         XXXXXXXXXXXXXXXXXXXXXXX
         XXXXXXXXXXXXXXXXXXXX
/ACTNBC:NNNX/CTPERS:NNNX//
 5           6

5KCASLTY
/MILPERCL ACTKIA ACTWIA ACTMIA ACTNBC CTPERS
/A          NNNX   NNNX   NNNX   NNNX   NNNX//
 1        2      3      4      5      6

5KMOSCAS
/SSI-MOS  ACTKIA ACTWIA ACTMIA ACTNBC CTPERS
/NNANX      NNNX   NNNX   NNNX   NNNX   NNNX//
 1        2      3      4      5      6
 NNANX
 NNAXX

RMKS/

DWNGRADE/AAAXXXXXXXXXXXXXXXXXXXXX//
         1
```

MESSAGE EXAMPLE

MESSAGE NUMBER: S006

TITLE: Casualty Information Report (CASSTATS)

UNCLAS
EXER/BALD EAGLE 88//
MSGID/CASSTATS/8 INF BN//
KDTGRPTN/180700/N01808//
KUNITCAS/8 INF BN/ACTKIA:12/ACTWIA:18/ACTMIA:1/ACTNBC:1/CTPER:32//
5KCASLTY

/MILPERCL	ACTKIA	ACTWIA	ACTMIA	ACTNBC	CTPERS
/O	1	4	-	-	5
/W	-	-	-	1	1
/E	11	14	1	-	26//

5KMOSCAS

/SSI-MOS	ACTKIA	ACTWIA	ACTMIA	ACTNBC	CTPERS
/11B41	1	4	-	-	5
/630A0	-	-	-	1	1
/11B10	8	10	-	-	18
/11B320	2	3	1	-	6
/11B30	1	1	-	-	2//

MESSAGE NUMBER: S034

TITLE: Supply Shortages (SHORTSUP)

GENERAL INSTRUCTIONS

This message is used to report the identification of supplies which because of their shortage could affect the effectiveness of a unit. It can be used by both using units and logistics activities. Using units would report supply shortages to their supporting logistics activity. Logistics activities could also use this format to report critical shortages to command and higher level logistics activities.

The AMPN set can be related to individual items in the 6KSHTSUP set through correlation with the CMNT field. Special comments related to individual items can be conveyed in the AMPN set.

The sets, EXER through NARR, are prepared in accordance with the message instructions for the initial main text sets.

SPECIAL INSTRUCTIONS

Set Identifier: DTGM [M].

Field 1, As of Date-Time [M]. Enter the date-time (date, hour, and minute) of the effective time of the report (DFI #C914, DUI A25).

Set Identifier: UNITIDM [M].

Field 1, Unit Designator: [M]. Enter the unit designator of the unit making the report using either (DFI #C095, DUI 001) or (DFI #E987, DUI 005).

Set Identifier: 6KSHTSUP [M].

Field 1, Logistical Support Item [M]. Enter the logistical support item in short supply (DFI #C460, DUI 001).

Field 2, Report Comment [O]. This field is used to designate if the specific item is on requisition, to provide the requisition document number, and to identify a specific item for reference in the following AMPN set. Enter "A2" if the item is not on requisition. Enter "A1" if the item is on requisition. The requisition document number may be added immediately following the A1 entry (DFI #E150, DUI 022).

MESSAGE NUMBER: S034

TITLE: Supply Shortages (SHORTSUP)

SPECIAL INSTRUCTIONS (Continued)

Set Identifier: AMPN [C]. This set is mandatory if entries are made in the report comment field of set 6KSHTSUP. Enter free text explanation of entry made in report comment field of set 6KSHTSUP.

Set Identifier: RMKS [O]. This set is used for any additional required information.

Set Identifier: DWNGRADE [C]. This set is mandatory when the message is classified.

Field 1, Downgrading and Declassification Markings [M]. Enter the appropriate downgrading and declassification markings if the message is classified (DFI #E679, DUI 001).

MESSAGE CONTENT

MESSAGE NUMBER: S034

TITLE: SUPPLY SHORTAGE (SHORTSUP)

PURPOSE: TO IDENTIFU SUPPLIES WHICH BECAUSE OF THEIR SHORTAGE COULD AFFECT THE COMBAT EFFECTIVENESS OF A UNIT.

SET IDENT	CAT	FIELD NO	MANDATORY ENTRY/ FLD DESC/COL HEADER	FIELD NAME	START COL J	NO/TYPE	DFI NO	DUI NO
EXER	C							
	M	1		EXERCISE NICKNAME		1- 56 ANBS	E 335	001
	O	2		EXERCISE MESSAGE ADDITIONAL IDENTIFIER		1- 16 AB	E 335	002
OPER	C							
	M	1		OPERATION CODEWORD		1- 32 AB	E 336	001
	O	2		PLAN ORIGINATOR AND NUMBER		3- 23 ANS	E 925	001
	O	3		OPTION NICKNAME		1- 23 ANBS	E 585	001
	O	4		SECONDARY OPTION NICKNAME		1- 23 ANBS	E 585	002
MSGID	M							
	M	1	SHORTSUP	MESSAGE TYPE		1- 20 ANBS	E 050	001
	M	2		ORIGINATOR		1- 20 ANBS	E 146	001
	O	3		MESSAGE SERIAL NUMBER		1- 7 ANBS	E 147	005
	O	4		MONTH		3 A	E 580	001
	O	5		QUALIFIER		3 A	E 568	001
	O	6		SERIAL NUMBER OF QUALIFIER		1- 3 N	E 487	017
DTGM	M							
	M	1		AS OF DATE-TIME		6 N	C 914	A25
UNITIDM	M							
	M	1		UNIT DESIGNATOR		6- 21 ANS	C 095	001
				UNIT DESIGNATOR		1- 24 ANBS	E 987	005
6KSHTSUP	M							
	M	1	MAT-EQUIP-VEH	LOGISTICAL SUPPORT ITEM	2L	11- 22 ANBS	C 460	001
	O	2	CMNT	REPORT COMMENT	25L	1- 18 ANBS	E 150	022
AMPN	C							
RMKS	O							
DWNGRADE	C							
	M	1		DOWNGRADING AND DECLASSIFICATION MARKINGS		1- 25 ANBS	E 679	001

MESSAGE MAP

MESSAGE NUMBER: S034

TITLE: SUPPLY SHORTAGES (SHORTSUP)

```
1234567890123456789012345678901234567890123456789012345678901234567890123456789

EXER/XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
     1
/XXXXXXXXXXXXXXXA//
 2

OPER/XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/XXXXXXXXXXXXXXXXXXXXXX
     1                                 2
/XXXXXXXXXXXXXXXXXXXXXXX/XXXXXXXXXXXXXXXXXXXXXX//
 3                       4

MSGID/SHORTSUP/XXXXXXXXXXXXXXXXXXX/XXXXXXX/AAA/AAA/NNN//
      1        2                    3       4   5   6

DTGM/NNNNNN//
     1

UNITIDM/XXXXXXXXXXXXXXXXXXXXXXX//
        1
        XXXXXXXXXXXXXXXXXXXXX
        XXXXXXXXXXXXXXXXXXXXXXX

6KSHTSUP
/MAT-EQUIP-VEH          CMNT
/NXXXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXX//
 1                       2

AMPN/

RMKS/

DWNGRADE/AAAXXXXXXXXXXXXXXXXXXXXXXX//
         1
```

MESSAGE EXAMPLE

MESSAGE NUMBER: S034

TITLE: Supply Shortages (SHORTSUP)

UNCLAS
EXER/LOG RED 88//
MSGID/SHORTSUP/2BDE19ARDIV/2007004//
DTGM/200600//
UNITIDM/2BDE19ARDIV//
6KSHTSUP

/MAT-EQUIP-VEH	CMNT
/20 ECH LWHEL M35A2	A18152-006
/12 ECH TANK M1	A2
/190 CSE MISC RATIONS	A2A
/5K GAL FUEL DSL	A2
/54 ECH LTMG M60	A2
/30 ECH MISC LITTER	A18157-003//

AMPN/A1 DENOTES ITEM ON REQUISITION, NUMBER FOLLOWING A1 IS REQUISITION DOCUMENT NUMBER. A2 DENOTES ITEM NOT ON REQUISITION. ITEM 8152-006 URGENTLY REQUIRED FOR RESUPPLY VEHICLES TO FORWARD DEPLOYED UNITS. ITEM A2A IS MCI//

MESSAGE NUMBER: SO26

TITLE: POL Locations (POLLOC)

GENERAL INSTRUCTIONS

The POL Locations message is used to announce the location, capabilities, and availability of Class III items. Actual quantities of fuel, oil, and lubricants are reported in set 3KCLTHRE. The servicing capability and other type of Class III items available at the specific CL III point are reported in the AMPN set.

The sets, EXER through NARR, are prepared in accordance with the message instructions for the initial main text sets.

SPECIAL INSTRUCTIONS

Set Identifier: DTGM [M].

Field 1, As of Date-Time [M]. Enter the as of time (date, hour, and minute) of the report (DFI #C914, DUI A25).

Set Identifier: 3KPOLLOC [M].

Field 1, Data Entry [M]. Enter the data entry number (DFI #E082, DUI 001).

Field 2, Point Name [M]. Enter the code name for the specific CL III point (DFI #E468, DUI 004).

Field 3, Location [M]. Enter the location of the specific CL III point using one of the following:

Location, Seconds (DFI #C011, DUI 043)
Location, UTM 10-Meter (DFI #C012, DUI 005)
Location, Minutes (DFI #C469, DUI 011)
Unit Location, Name (DFI #E500, DUI 043)
Location, UTM 100-Meter (DFI #C542, DUI 012).

Set Identifier: 3KCLTHRE [M].

Field 1, Data Entry [M]. Enter the data entry number from set 3KPOLLOC which identifies the appropriate Class III point (DFI #E082, DUI 001).

Field 2, Fuel Quantity and Type [M]. Enter the quantity, unit of measurement, and fuel type on hand at the time of the report (DFI #C4007, DUI A01).

MESSAGE NUMBER: S026

TITLE: POL Locations (POLLOC)

SPECIAL INSTRUCTIONS (Continued)

Set Identifier: 3KCLTHRE [M]. (Continued)

Field 3, Oil Quantity and Type [M]. Enter the quantity, unit of measurement, and oil type on hand at the time of the report (DFI #C4032, DUI A01).

Field 4, Lubricant Quantity and Type [M]. Enter the quantity, unit of measurement, and lubricant type on hand at the time of the report (DFI #C4033, DUI A01).

Field 5, 3KCLTHRE Comments [O]. Enter any pertinent comments about the POL point in the space provided, or if more space is required, include a reference note or number and expand in the AMPN set below (DFI #E150, DUI A01).

Set Identifier: AMPN [C]. This set is mandatory if reporting additional information pertaining to set 3KCLTHRE. Information on additional type of Class III items available and POL servicing capability should be reported in this set.

Set Identifier: RMKS [O]. This set is used for any additional required information.

Set Identifier: DWNGRADE [C]. This set is mandatory when the message is classified.

Field 1, Downgrading and Declassification Markings [M]. Enter the appropriate downgrading and declassification markings if the message is classified (DFI #E679, DUI 001).

MESSAGE NUMBER: S026

TITLE: POL LOCATIONS (POLLOC)

PURPOSE: TO ANNOUNCE THE LOCATION AND CAPABILITIES OF POL POINTS TO PROVIDE RESUPPLY.

SET IDENT	CAT	FIELD NO	MANDATORY ENTRY/ FLD DESC/COL HEADER	FIELD NAME	START COL J	NO/TYPE	DFI NO	DUI NO
EXER	C							
	M	1		EXERCISE NICKNAME		1- 56 ANBS	E 335	001
	O	2		EXERCISE MESSAGE ADDITIONAL IDENTIFIER		1- 16 AB	E 335	002
OPER	C							
	M	1		OPERATION CODEWORD		1- 32 AB	E 336	001
	O	2		PLAN ORIGINATOR AND NUMBER		3- 23 ANS	E 925	001
	O	3		OPTION NICKNAME		1- 23 ANBS	E 585	001
	O	4		SECONDARY OPTION NICKNAME		1- 23 ANBS	E 585	002
MSGID	M							
	M	1	POLLOC	MESSAGE TYPE		1- 20 ANBS	E 050	001
	M	2		ORIGINATOR		1- 20 ANBS	E 146	001
	O	3		MESSAGE SERIAL NUMBER		7 N	E 147	005
				REPORT SERIAL		4- 5 ANS	E 147	006
	O	4		MONTH		3 A	E 580	001
	O	5		QUALIFIER		3 A	E 568	001
	O	6		SERIAL NUMBER OF QUALIFIER		1- 3 N	E 487	017
REF	O,R							
	M	1		SERIAL LETTER		1 A	E 636	002
	M	2		MESSAGE TYPE		1- 20 ANBS	E 050	001
				COMMUNICATION TYPE		3 A	E 646	001
	M	3		ORIGINATOR		1- 20 ANBS	E 146	001
	M	4		DATE OF REFERENCE, YEAR-MONTH-DAY		6 N	C 075	010
				DAY-TIME OF REFERENCE		7 AN	C 143	005
				DATE-TIME GROUP		13 ANB	C 647	001
				DATE OF REFERENCE, DAY-MONTH-YEAR		6 N	C 648	002
				DATE OF REFERENCE, DAY-ALPHAMONTH-YEAR		7 AN	C 649	002
	O	5		MESSAGE SERIAL NUMBER		7 N	E 147	005
				REPORT SERIAL		4- 5 ANS	E 147	006
	D	6		SPECIAL NOTATION		5 A	E 1042	001
	O,R	7		NASIS CODE		3 A	E 332	001
ANPN	C							
NARR	C							

MESSAGE NUMBER: SO26

TITLE: POL LOCATIONS (POLLOC)

```
          12345678901234567890123456789012345678901234567890123456789012345678901234567890

          EXER/XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
               1
          /XXXXXXXXXXXXXXXA//
           2

          OPER/XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/XXXXXXXXXXXXXXXXXXXXXX
               1                                2
          /XXXXXXXXXXXXXXXXXXXXXXX/XXXXXXXXXXXXXXXXXXXXXX//
           3                       4

          MSGID/POLLOC/XXXXXXXXXXXXXXXXXXX/NNNNNNN/AAA/AAA/NNN//
                1      2                   3       4   5   6
                                           NNNNNNN
                                           AXNXX

          REF/A/XXXXXXXXXXXXXXXXXXXX/XXXXXXXXXXXXXXXXXXX/NNNNNNABAAANN/NNNNNNN
              1 2                    3                    4             5
                XXXXXXXXXXXXXXXXXXXX                      NNNNNNA       NNNNNNN
                AAA                                       NNNNNNA       AXNXX
                                                          NNNNNNABAAANN
                                                          NNNNNN
                                                          NNAAANN

          /AAAAA/AAA//
           6     7

          AMPN/

          NARR/
```


ALCS-A3-500-003
JUNE 1984

MESSAGE NUMBER: S026

TITLE: POL LOCATIONS (POLLOC)

SET IDENT	CAT	FIELD NO	MANDATORY ENTRY/ FLD DESC/COL HEADER	FIELD NAME	START COL J	NO/TYPE	DFI NO	DUI NO
DTGM	M							
	M	1		AS OF DATE-TIME		6 N	C 914	A25
3KPOLLOC	M							
	M	1	DE	DATA ENTRY	2R	2 N	E 082	001
	M	2	PTNAME	POINT NAME	5L	1- 26 ANBS	E 468	004
	M	3	LOCATION	LOCATION, SECONDS	32L	15 AN	C 011	043
			LOCATION	LOCATION, UTM 10-METER	32L	13 AN	C 012	005
			LOCATION	LOCATION, MINUTES	32L	11 AN	C 469	011
			UNITLOC	UNIT LOCATION, NAME	32L	1- 20 ANBS	E 500	043
			LOCATION	LOCATION, UTM 100-METER	32L	11 AN	C 542	012
3KCLTHRE	M							
	M	1	DE	DATA ENTRY	2R	2 N	E 082	001
	M	2	FUEL-QTY-TYP	FUEL QUANTITY AND TYPE	5L	5- 11 AN	C 4007	A01
	M	3	OIL-QTY-TYP	OIL QUANTITY AND TYPE	18L	9- 14 ANS	C 4032	A01
	M	4	LUB-QTY-TYP	LUBRICANT QUANTITY AND TYPE	33L	9- 14 ANS	C 4033	A01
	O	5	CMNTS	3KCLTHRE COMMENTS	48L	1- 15 ANBS	E 150	A01
AMPN	C							
RMKS	O							
DWNGRADE	C							
	M	1		DOWNGRADING AND DECLASSIFICATION MARKINGS		1- 25 ANBS	E 679	001

MESSAGE MAP

MESSAGE NUMBER: SO26

TITLE: POL LOCATIONS (POLLOC)

```
          12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789

          DTGM/NNNNNN//
               1

          3KPOLLOC
          /DE PTNAME                        LOCATION
          /NN XXXXXXXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXX//
           1  2                            3
                                            NNNNNNANNNNNNNA
                                            NNAAANNNNNNNN
                                            NNNNANNNNNA
                                            XXXXXXXXXXXXXXXXXX
                                            NNAAANNNNNN

          3KCLTHRE
          /DE FUEL-QTY-TYP OIL-QTY-TYP    LUB-QTY-TYP    CMTS
          /NN NXXXXXXXXXX  NXXXXXXXXXXXXX NXXXXXXXXXXXXX XXXXXXXXXXXXXXX//
           1  2            3              4              5

          AMPN/

          RMKS/

          DWNGRADE/AAAXXXXXXXXXXXXXXXXXXXXXX//
                   1
```

MESSAGE EXAMPLE

MESSAGE NUMBER: S026

TITLE: POL Locations (POLLOC)

C O N F I D E N T I A L**
EXER/DESERT GALE 88//
MSGID/POLLOC/9SPTCMD/1911825//
DTGM/240600//
3KPOLLOC

/DE	PTNAME	LOCATION
/01	PIPE END 3	62RST12345678
/02	BLUE STAR	62RST56781234//

3KCLTHRE

/DE	FUEL-QTY-TYP	OIL-QTY-TYP	LUB-QTY-TYP	CMTS
/01	25000GALMGS	550GALSAE40	200LBOG90	NOTE 1
/-	50000GALDSL	40BBLSAE10	-	-
/02	50000GALDSL	100GALSAE40	-	NOTE 2//

AMPN/(NOTE 1) 400 GALSGF AVAILABLE, CL III PT CAPABLE OF SERVICING 2 TANKERS AT THE SAME TIME. (NOTE 2) 2000STCOAL AVAILABLE IN 100LB SACKS, PT CAPABLE OF SERVICING 3 TANKERS AT THE SAME TIME, MGS EXPECTED TO BE AVAILABLE 241200//
DWNGRADE/DECL 24 NOV 90//

** Information on this page is UNCLASSIFIED. Classification is shown for example purposes only.

Appendix 2 - TACCNET Demonstration Scenarios

The following pages contain the demonstration scenarios used for the Advanced Experimental Demonstrations presented during the project. These scenarios were developed jointly by AIRMICS and Georgia Tech personnel and are intended to be representative of actual field operations which might be performed by the CSSCS units.

Appendix 3 - TACCNET Presentation Materials

The following pages contain copies of the visual aids and presentation materials developed for the IPR and other presentations during the project.

CSSCS Advanced Experimental Demonstrations

1983 - 1986

Development of a Tactical Army Command and Control Network

(TACCNET)

Objectives:

To examine issues related to information transfer among loosely-coupled, occasionally connected, heterogeneous, asynchronous networks of networks.

To develop a prototype Combat Service Support Computer System and a prototype Command and Control Database to be used in the exploration of CSS information processing requirements.

Approach

Iterative Refinement

- Develop expertise
- Design and build prototype
- Demonstrate capabilities
- Examine and refine

Experimental Demonstrations

- Advanced Experimental Demonstration (AED)
- Demonstrate capabilities
- Highlight issues
- Incorporate previous work
- Provide recommendations for future work

End Product

- Working, portable, full-featured prototype
- Documentation of issues and concerns
- Specification for interim, fieldable system

Accomplishments

- Information transfer among network of widely differing machines (S/1, CDC, IBM 4300, Vax/Unix, PC) over a variety of links (3780, BISYNC, asynchronous dialup, token ring)
- Prototype TACCNET using PC/Unix and Honeywell/GCOS featuring automated routing, failure detection, and rerouting
- Extended TACCNET featuring database backup and recovery, file transfer, message processing, and screen-oriented user interface
- C^2 Database with automated JINTACCS interface
- JINTACCS screen-oriented, automated composition tool
- Source-level system portability
- Simulation of CSSCS network (SLAM)

Overview of Research

1983 - 1984

Heterogeneous communications
Low-speed asynchronous networks
Routing and identification

1984 - 1985

Development of TACCNET
Failure detection and management
Automatic routing
C^2 database analysis

1985 - 1986

Expansion of TACCNET
JINTACCS message composition
C^2 database interface
Database backup and recovery
User interface

Overview of Research

1986 - ?

JINTACCS message processing

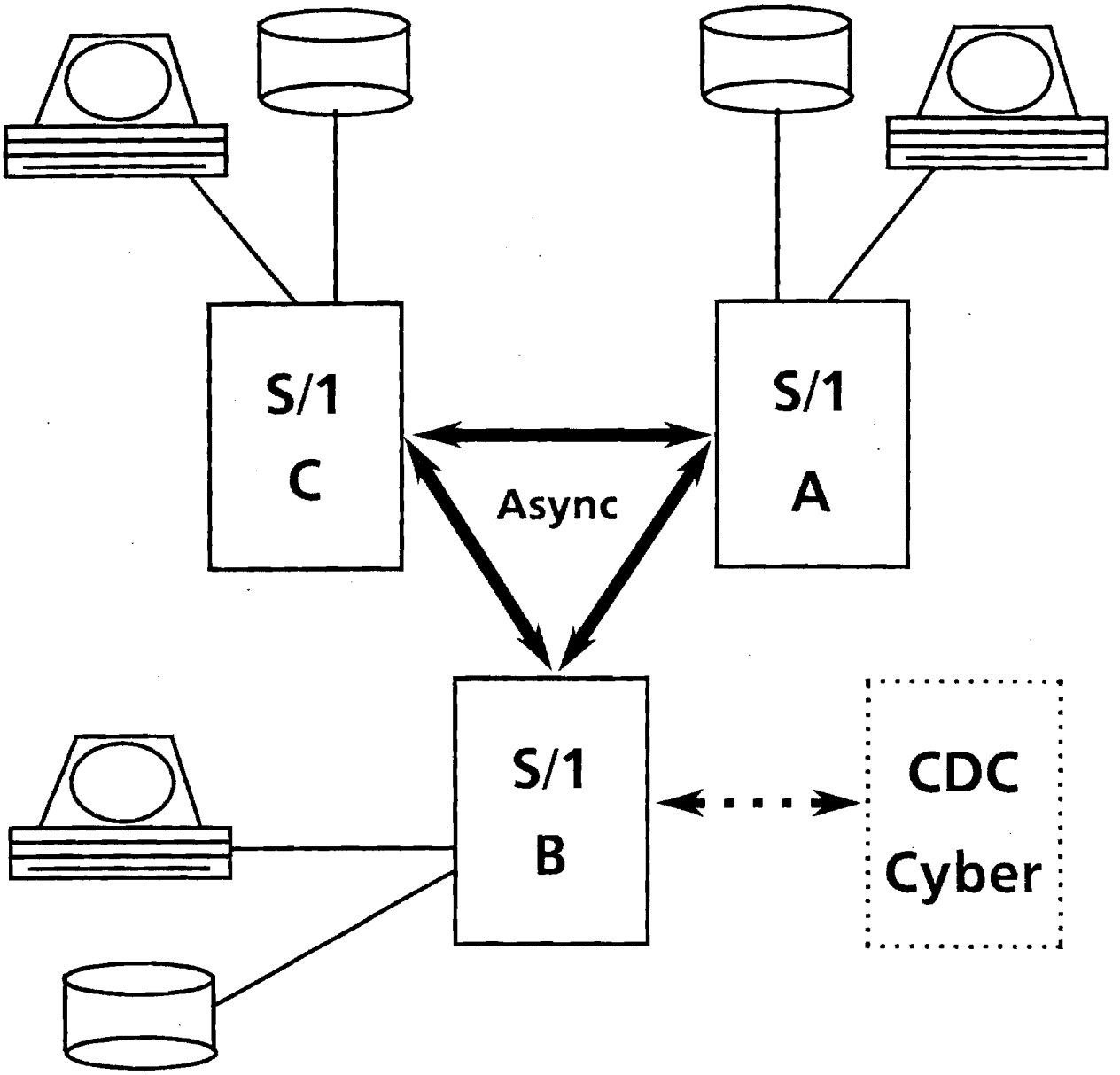
- Analysis of interface between JINTACCS messages and C2 database
- Functional model of JINTACCS messages
- Development of JINTACCS grammar or definition language
- Development of a **generic** JINTACCS message handler (parser?)
- JINTACCS message composition aid expansion to include editing capabilities

1983 - 1984

System Features

- Terminal on one system acts as a console on another system
- Files may be transferred between systems
- Asynchronous communication links
- Three or more machines in network (3 Series/1's, 1 CDC Cyber)
- Positive identification of remote systems

1983 - 1984
Network Configuration



1984 - 1985

System Features

- On-demand communications via asynchronous links
- Bidirectional communications
- Error-detecting and error-correcting packet-oriented protocol
- Failure detection and rerouting
- Queue-oriented message processing
- Positive identification of remote systems

Plans

- Finalize and document TACCNET prototype

- Explore JINTACCS message processing issues
  - JINTACCS grammar or definition language
  - Functional definition of messages
  - Message processing tools

- Design Command and Control Database
  - Top-down design approach
  - Analysis of intended usage/user requirements
  - Determine structure and content from usage requirements
  - Interface with JINTACCS

- Convert to ADA

Status

- Completed and installed a well-defined, fully featured prototype TACCNET for CSSCS environment

- Developing detailed specification of TACCNET system design and implementation

- Beginning first year of two-year investigation of automated JINTACCS message processing

CSSCS Environment

- Nodes subject to catastrophic failures
- Nodes are physically mobile but logically static
- Frequent, expected, but unpredictable reconfiguration
- Nodes are loosely coupled and occasionally connected
- Machines are physically small (microcomputers)
- Communications links are undetermined (media transparency required)
- On-demand communication links
- Time constraints/priority messages
- Most messages in JINTACCS format
- Well-defined hierarchy of nodes

Why TACCNET?

Why not use *uucp*, Kermit, or other widely available data transfer systems?

- No existing product conforms to CSSCS environmental constraints (rerouting, failure management, JINTACCS message handling, time constraints, priority messages, observance of node hierarchy, etc.)

Why UNIX?

Advantages:

- Availability on many different architectures
- Portability (many machines in desired size class)
- Good environment for software development
- Convenient file structure (i.e., directories as queues)
- Process control and inter-process communication
- Multi-user, multi-tasking system
- Standard, portable high-level language (C)

Disadvantages:

- Unix is a "moving boundary"
- Not "friendly" to naive user
- Many variants in distribution
- Missing features (such as file locking)
- Security

TACCNET Capabilities and Functions

- Heterogeneous communications
  - Media transparency
  - Error-detecting protocol with retransmission
  - Logging of connections, errors, and message transfers
  - Bidirectional, on-demand links
  - Tunable parameters (i.e., speed, packet size, retry delays)
  - Remote system identification
  - Broadcast and message rejection
  - Failure detection and management
- File transfer
- Electronic mail
- JINTACCS to and from C2 database
- Automated JINTACCS message composition interface
- Distributed C2 database backup and recovery
- Single machine emulation of multiple nodes
- Network management functions via messages
- Dynamic network configuration
- Screen-oriented, menu-driven user interface
- Message forwarding/holding
- Store-and-forward message transfer
- Automatic routing via shortest path

TACCNET

Objectives:

- Pass JINTACCS messages
- Detect and handle failures
- Automatic (re)routing
- Dynamic network configuration
- Messages to and from C^2 database
- Database backup and recovery
- User interface
- JINTACCS message composition aids

TACCNET

Constraints:

- Ordinary telephone lines
- 1200 bps transmission rate
- Auto-dial / auto-answer modems
- Media transparency
- TACCS/UNIX, DAS3/GCOS

TACCNET

Additional Features:

- Error detection and recovery
- Data transparency
- Binary data transfer
- Store and forward capability
- Priority message scheduling
- On-line JINTACCS message dictionary
- Password security
- File transfer
- Electronic mail
- Multiple node emulation
- Tunable system parameters
- Portability (all code written in C language)
- Menu-driven system interface

TACCNET Composition

Communications

qms	-	scheduling
caller	-	connections
iocontrol	-	transmission

Message Generation and Processing

genmsg	-	generation
msgproc	-	processing

Database Operations

jms	-	message composition
server	-	messages into C^2 DB
build	-	messages from C^2 DB

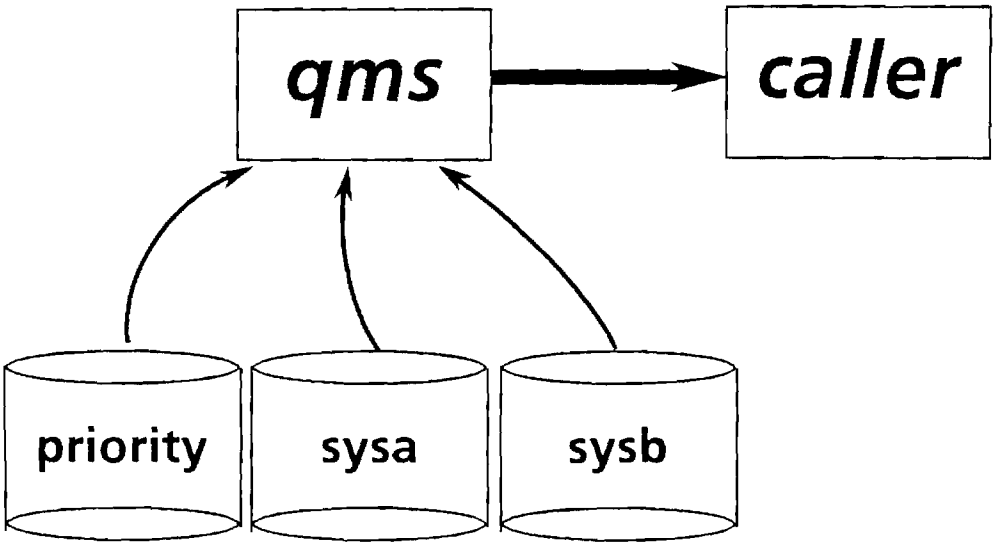
User Interface

console	-	system administration

Communications

qms

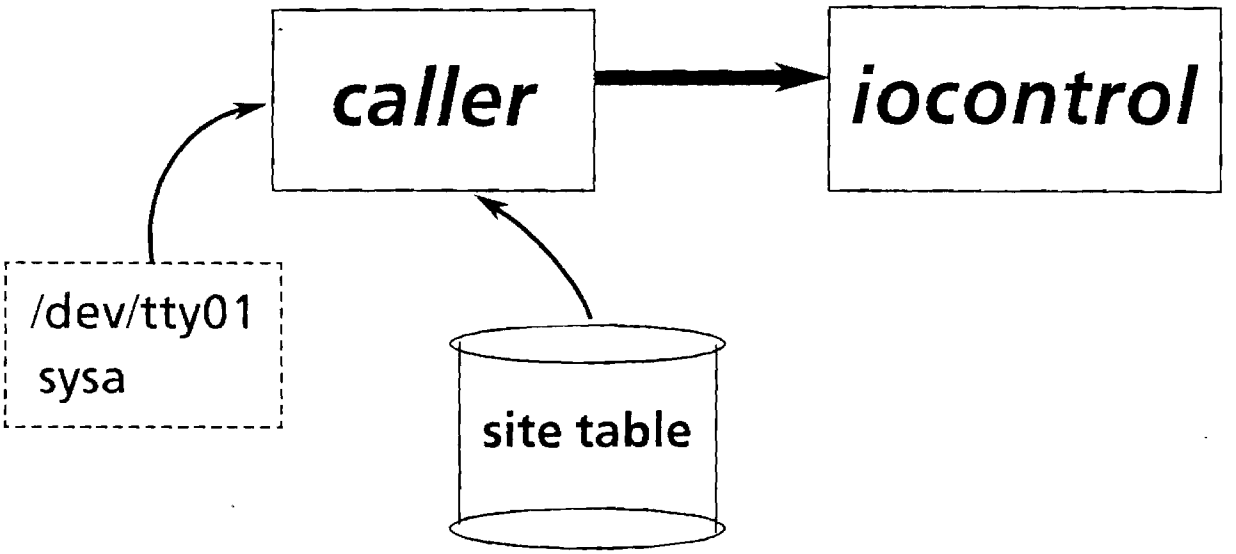
- Runs in background (sleep or cron)
- Scans priority queue first, then system queues in order taken from site table
- Invokes *caller*
- Handles preemption for priority messages



Communications

caller

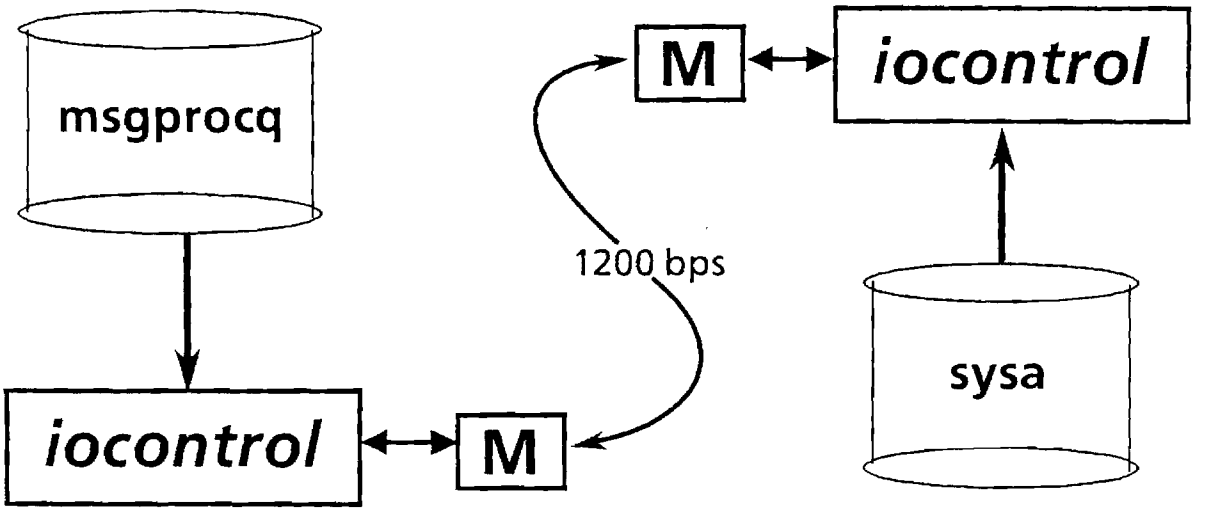
- Validates site information
- Establishes connection
- Handles connection failures
- Starts *iocontrol* process
- Handles transmission failures
- Releases port and system queue
- Updates site table



Communications

iocontrol

- Transmits / receives files
- Gets files from system queue
- Puts files into message processor queue
- Error detection / correction
- Priority preemption
- Data transparency / binary data transfer



Communications

Transmission Protocol

- Similar to BSC (Stop & Wait, Window = 1)
- Data packets / control packets
- ASCII control codes

  DLE - Data Link Escape
  STX - Start of Text
  ETX - End of Text
  ETB - End Text Block
  EM - End of Message
  EOT - End of Transmission
  ACK - Acknowledge
  NAK - Negative Acknowledge
  CAN - Cancel

- Packets "punctuated" with **CR** for GCOS

Communications

Packet Formats

- **o** Data Packets

  12 bytes for frame, text block is variable length (tunable parameter)

- **o** Control Packets

  Always 4 bytes

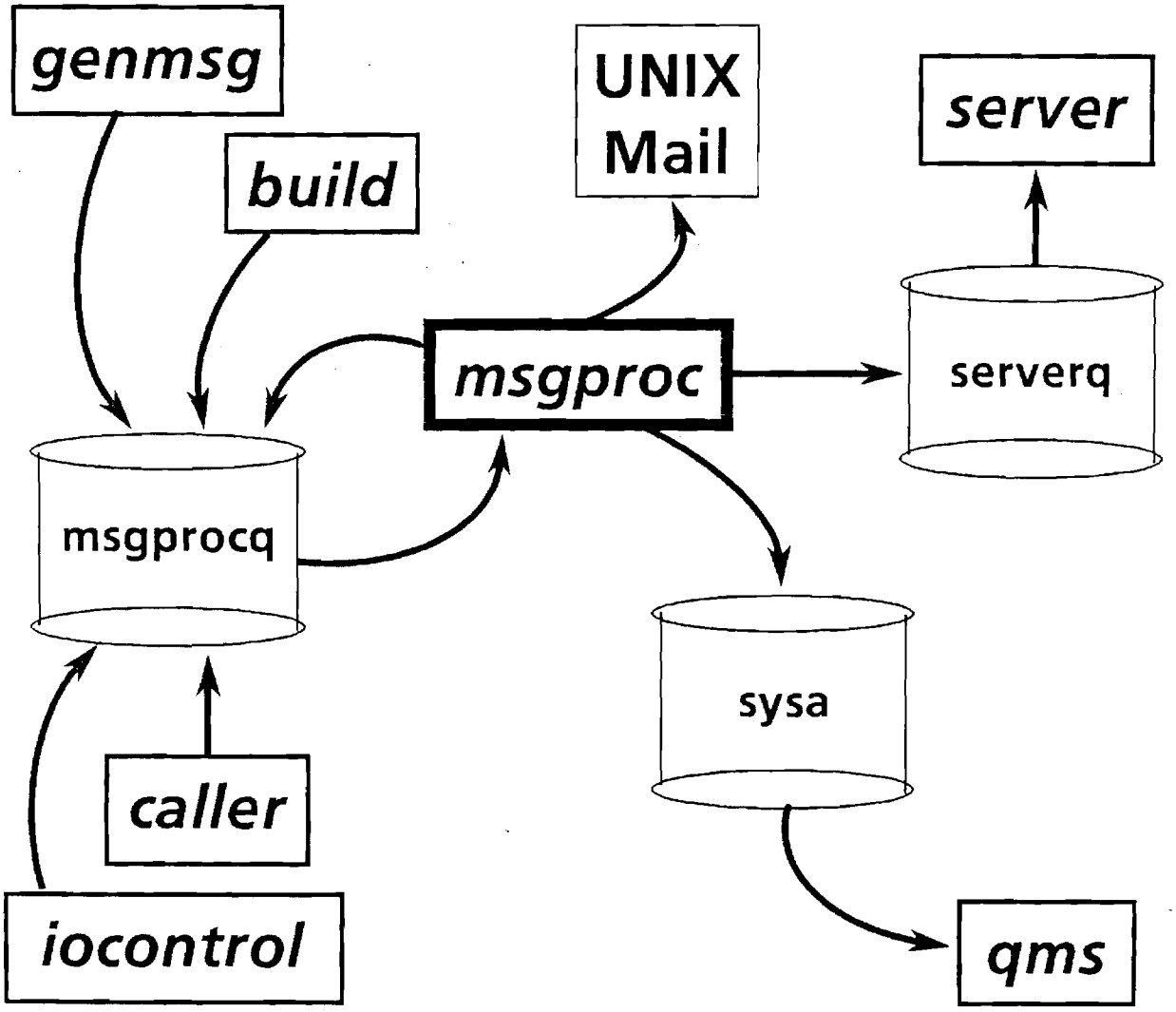
Data packet format:

Control packet format:

Message Processing

msgproc

- All messages pass through msgproc
- Processing is based on message type
- Routing is based on message header



Message Processing

msgproc

- Message file name indicates type

  Format:

  T*sysname*XXXXX

  Where:

  T = message type

  sysname = originating node

  XXXXX = hex timestamp

- Valid message types:

  A - Administrative
  C - Courtesy copy
  E - Undeliverable
  H - Bad header
  M - New message
  N - Rejected message
  P - Priority message
  R - Routine message
  S - Invalid path
  U - User mail

Message Processing

msgproc

- Routing based on message header

  >priority [c-flag]
  >source path
  >destination path

- Path format:

  site[!site!site...][!user]

  Where:

  site is a valid node ID;

  user is either "net.adm", "server", or a valid user on the node

- Message may have multiple headers (first is current)

Database Design

Message Dictionary

Hierarchical system may be preferred:

- JINTACCS message defined in hierarchical fashion
- Message database is primarily used as (static) message dictionary
- Information accessed hierarchically (fields within sets within messages)
- Speed of operation is desirable
- No need for relational query capability
- Reduced redundancy

Database Design

Command and Control (C^2)

Relational may be preferred:

- Information **not** defined in hierarchical manner
- Data items may be related in *many-to-many* fashion
- Database contents are dynamic
- Relational query capability desirable
- User-oriented interface necessary

First cut: extract from JINTACCS messages and normalize

Database Operations

jms

- User interface for message composition
- Uses message dictionary to build prompt panels
- Builds message in JINTACCS format and submits to *msgproc*
- User can review, edit, or save message during composition
- New messages are easily added for automated composition assistance

Database Operations

server (automated message posting)

- Reads JINTACCS messages
- Extracts variable data into file
- Builds UNIFY update in file
- Calls UNIFY to enter data update
- Old data overwritten by new

Limitations:

- C source module for each message
- Needs embedded query language

Database Operations

build (automated message generation)

- Given: message ID
  destination
  priority
- Executes UNIFY query, capturing data into file
- Reads data from file, puts into JINTACCS template in new file
- Submits new message file to *msgproc* for input to system
- Table-driven; does not use message dictionary

Limitations:

- C source module for each message
- Need embedded query language

Database Operations

Backup and Recovery

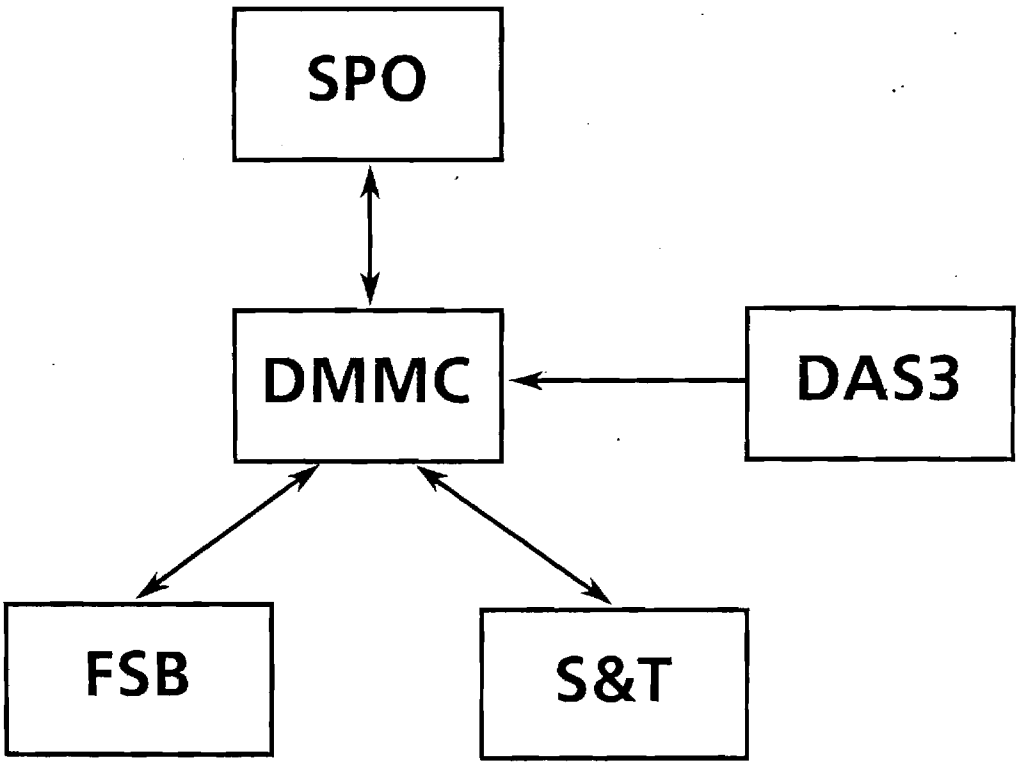
- Uses TACCNET to copy snapshot of database to remote node(s)
- Broadcast messages used to retrieve messages sent after snapshot
- Backup and recovery procedures initiated by user or *cron*

To recover from a failure:

- The snapshot is retrieved from one of the remote backup sites,
- A broadcast message is sent to the network requesting retransmission of all messages sent to the failed site after the snapshot was made.

Network Simulation

- SLAM simulation on CDC Cyber
- Low, moderate, and high traffic
- Assume basic network segment
- 1 - 3 ports (*dialin* and *dialout*)

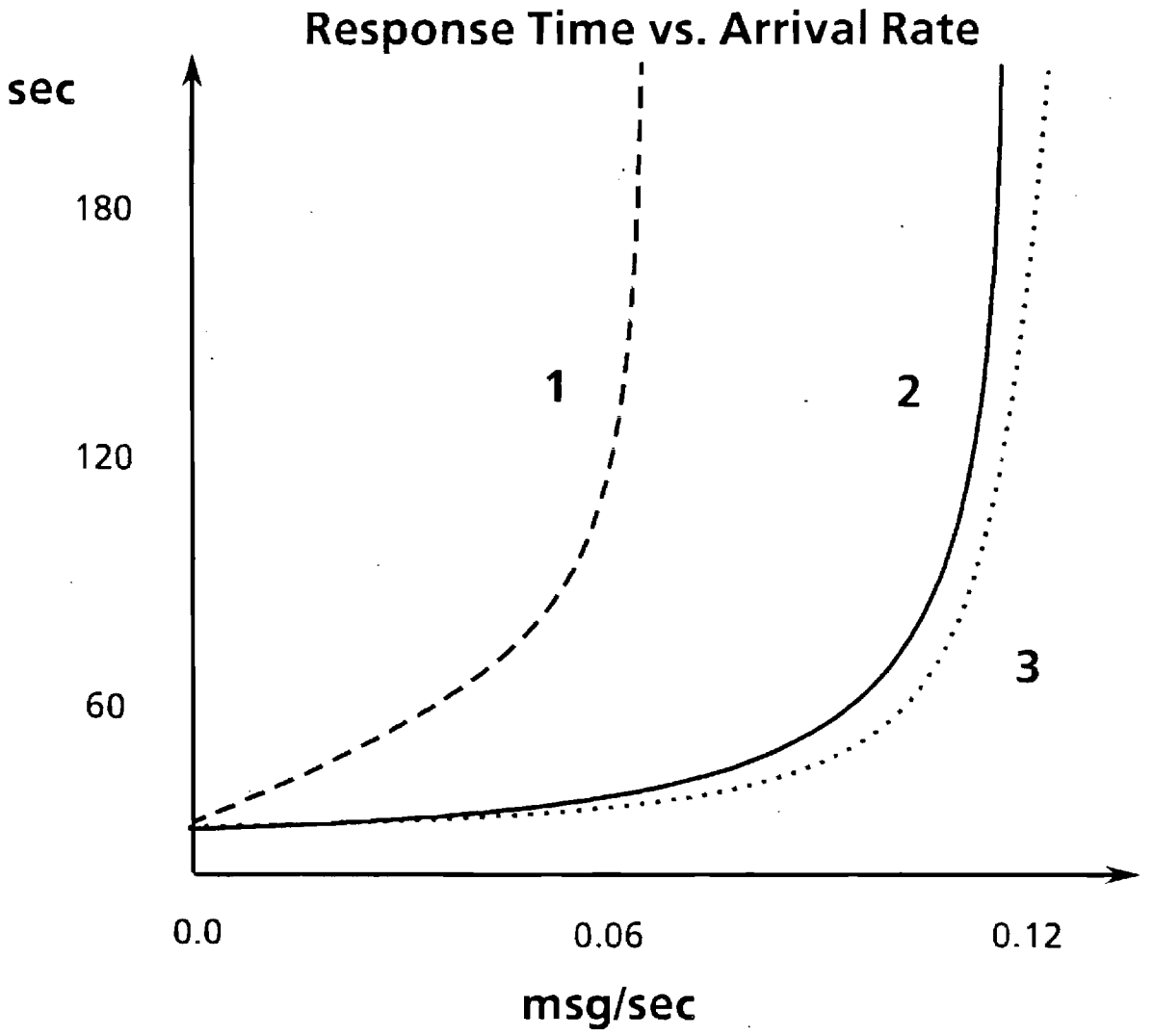


Basic network segment

Network Simulation

Results

- Bottleneck at DMMC
- 2 *dialin*, 2 *dialout* gives best results
- 1 *dialin*, 1 *dialout* is OK for leaf nodes



Further Investigation

JINTACCS processing

Message grammar, parsing, functional description

Security

Data and network security

Expert Systems

Structured format with ambiguities; message processing

Voice Technology

Voice / data interface for composition and display

User Interface / Tools

Edit / display messages during creation; insulation from JINTACCS

Distributed Database

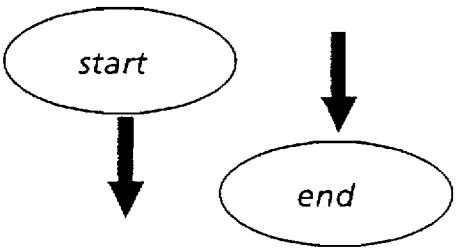
Consistency, redudancy, fault tolerance

Appendix 4 - TACCNET Data Flow Diagrams

The following pages contain the high level data flow diagrams for all of the major systems and subsystems comprising the TACCNET software system. A legend is provided at the beginning.

Legend

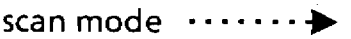
Module entry and exit points



Flow of program control

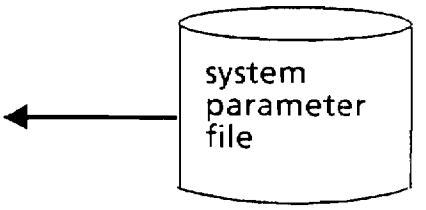
Flow of data to/from disk files or program modules

Command line arguments from user or parent program

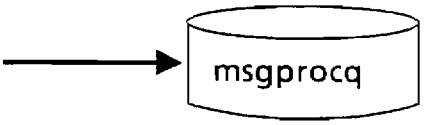


Program control statements

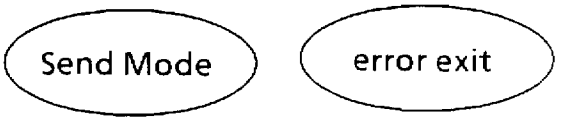
Flow of data to/from tables or messages (files)



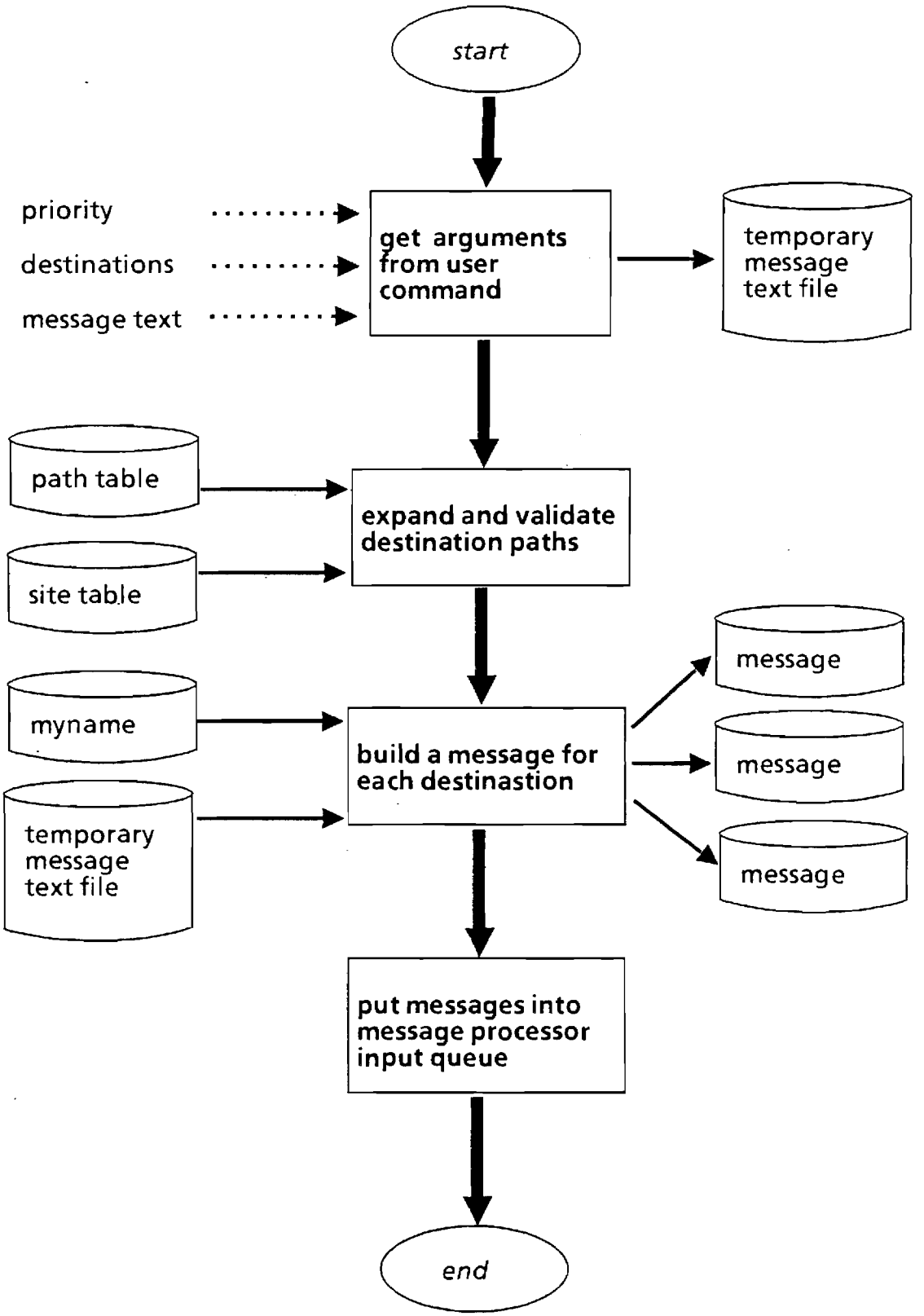
Flow of data (files) to/from directories (queues)



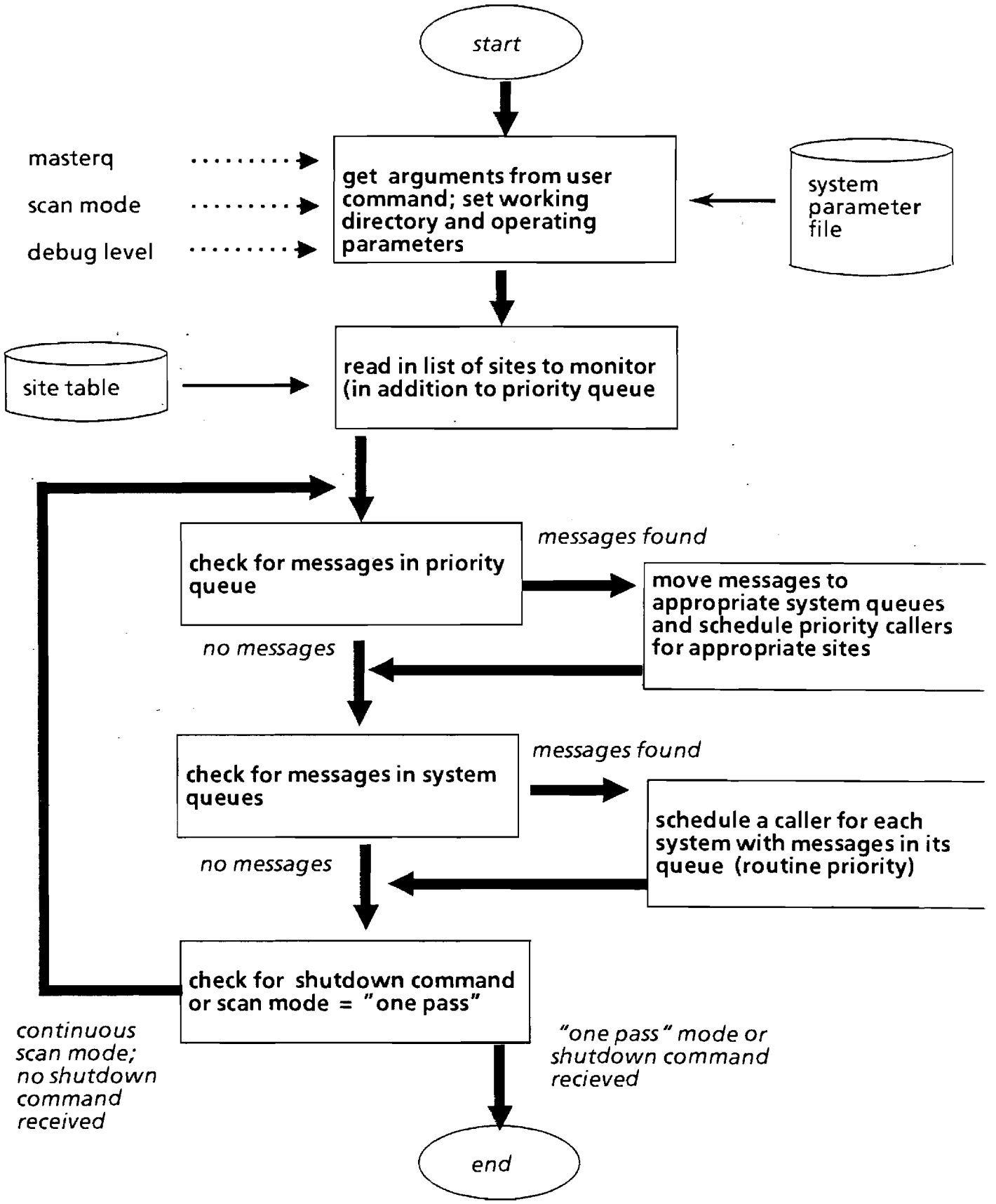
Special entry and exit points for errors and procedure calls



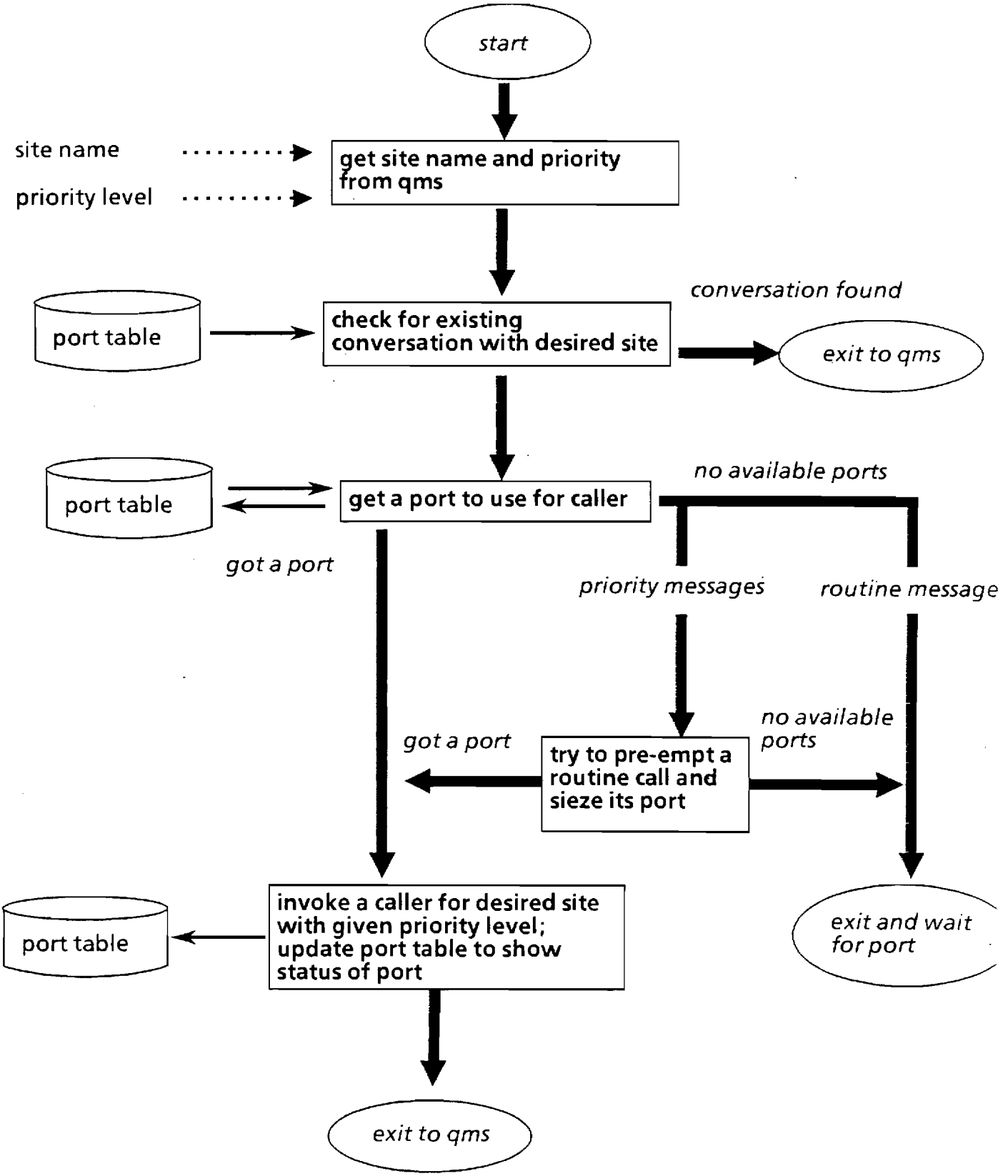
Genmsg


start
priority
destinations
message text
get arguments from user command
temporary message text file
path table
site table
expand and validate destination paths
myname
temporary message text file
build a message for each destinastion
message
message
message
put messages into message processor input queue
end

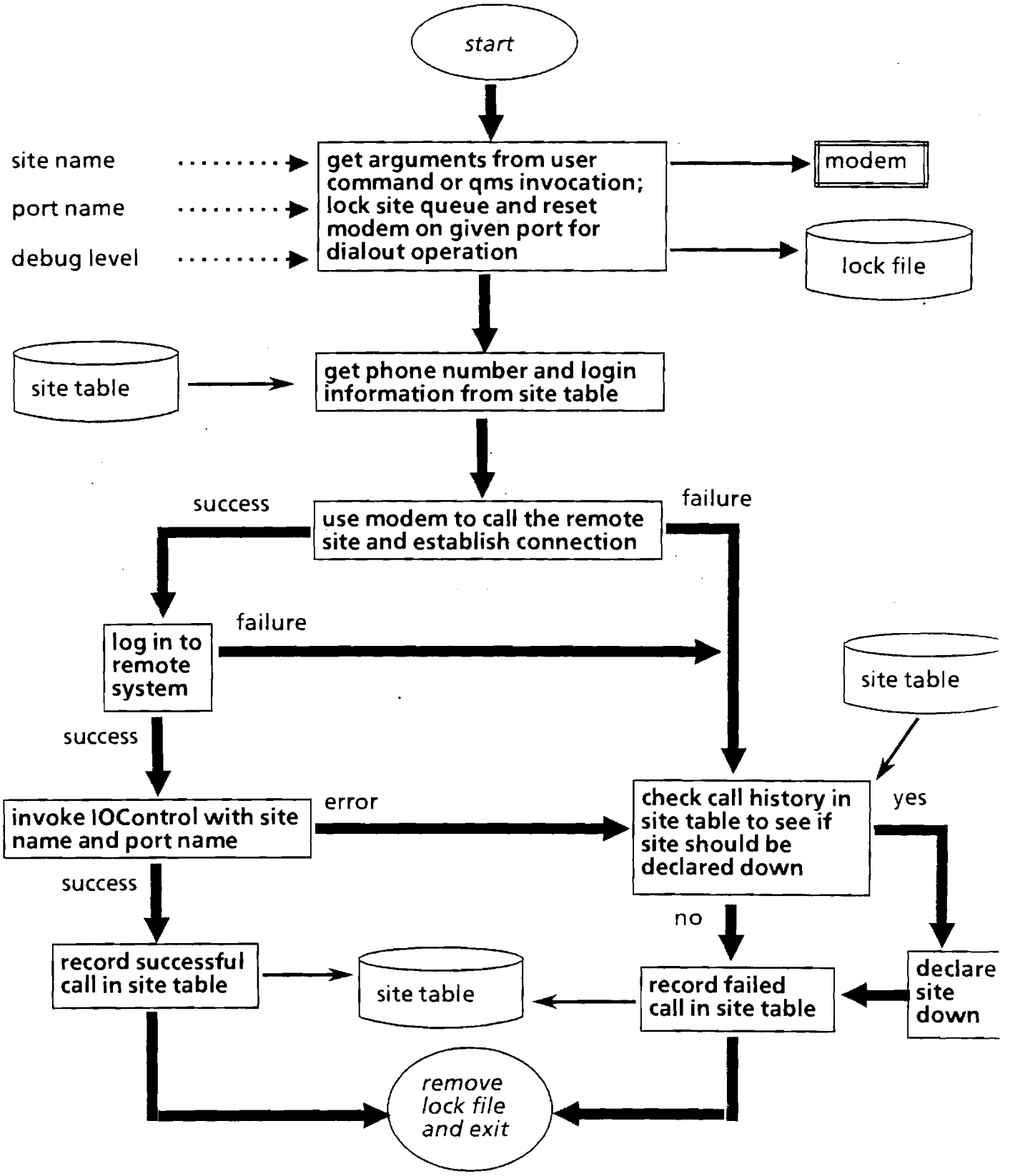
QMS


start
masterq
scan mode
debug level
get arguments from user command; set working directory and operating parameters
system parameter file
site table
read in list of sites to monitor (in addition to priority queue
check for messages in priority queue
messages found
move messages to appropriate system queues and schedule priority callers for appropriate sites
no messages
check for messages in system queues
messages found
schedule a caller for each system with messages in its queue (routine priority)
no messages
check for shutdown command or scan mode = "one pass"
continuous scan mode; no shutdown command received
"one pass" mode or shutdown command recieved
end

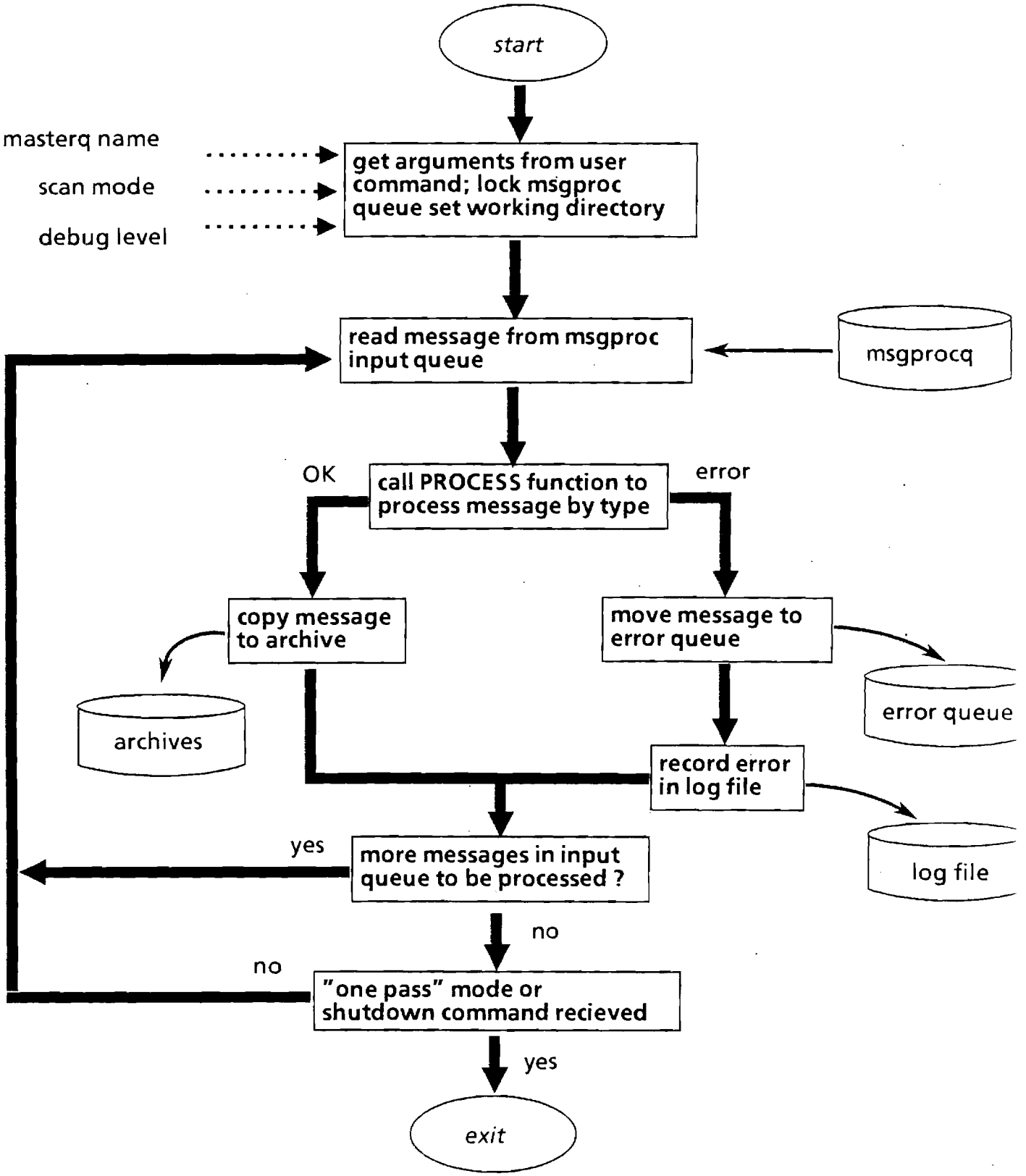
Schedule


start
site name
priority level
get site name and priority from qms
port table
check for existing conversation with desired site
conversation found
exit to qms
port table
get a port to use for caller
no available ports
got a port
priority messages
routine message
got a port
try to pre-empt a routine call and sieze its port
no available ports
port table
invoke a caller for desired site with given priority level; update port table to show status of port
exit and wait for port
exit to qms

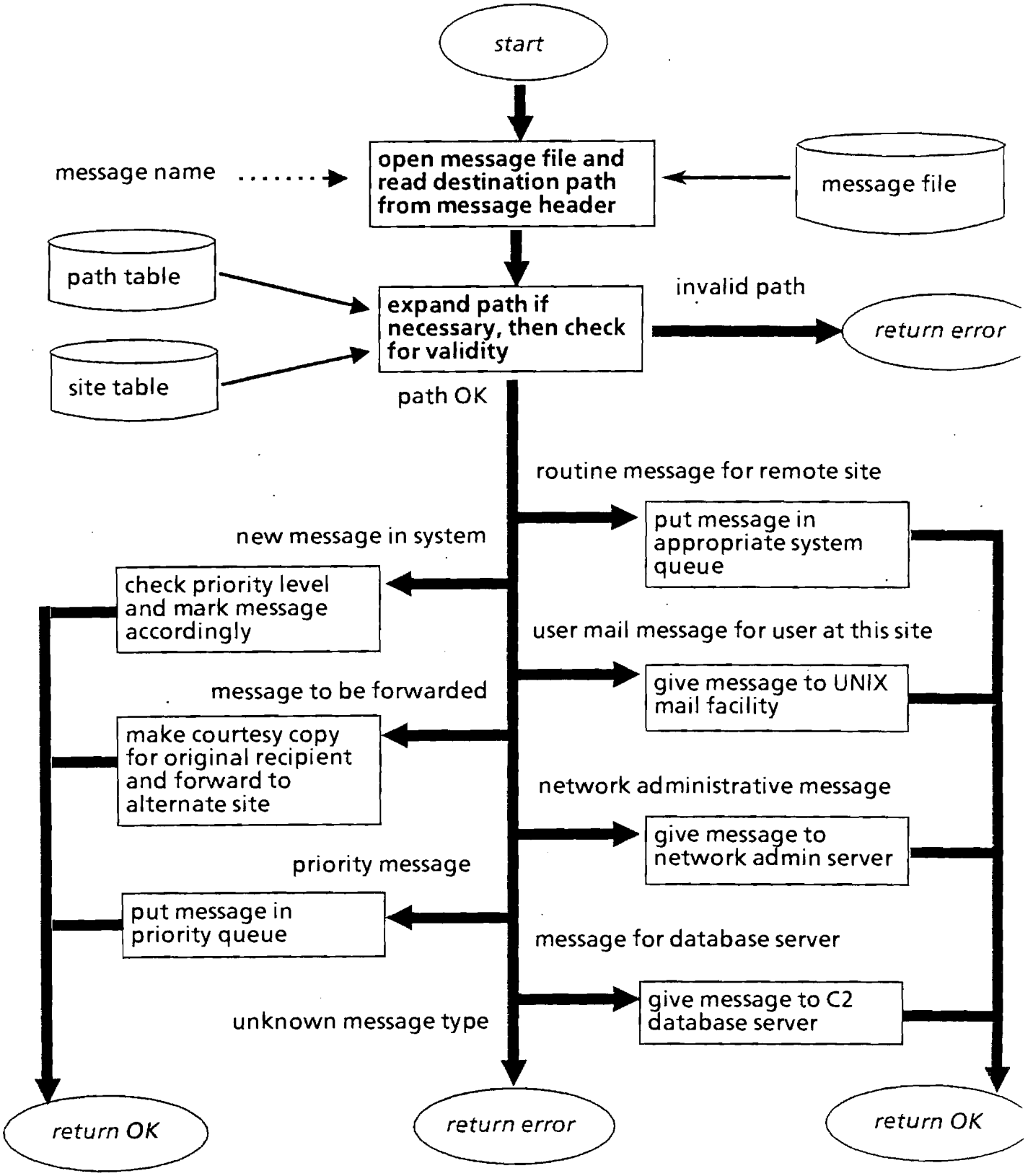
Caller


start
site name
port name
debug level
get arguments from user command or qms invocation; lock site queue and reset modem on given port for dialout operation
modem
lock file
site table
get phone number and login information from site table
success
use modem to call the remote site and establish connection
failure
log in to remote system
failure
site table
success
invoke IOControl with site name and port name
error
check call history in site table to see if site should be declared down
yes
success
no
record successful call in site table
site table
record failed call in site table
declare site down
remove lock file and exit

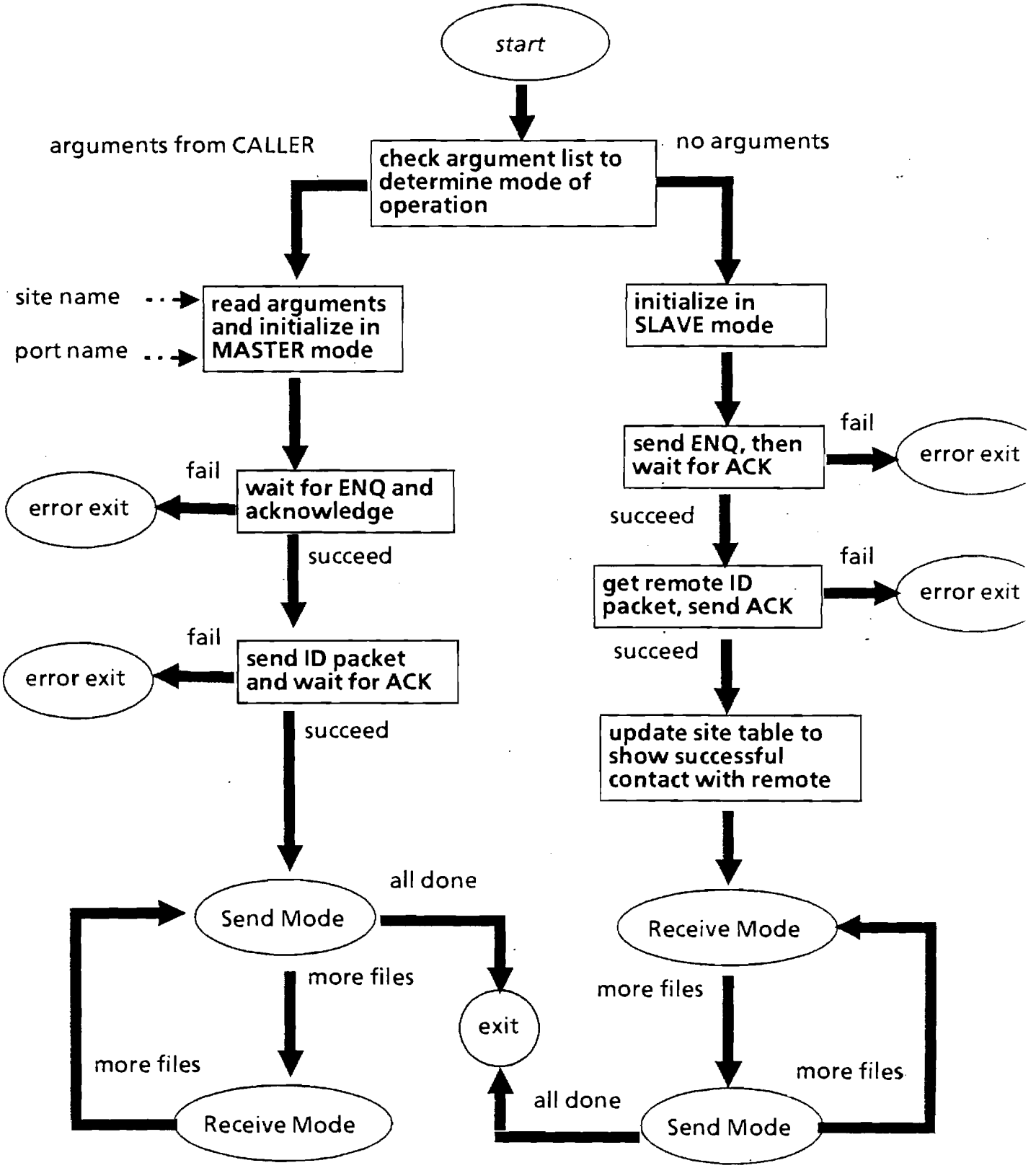
Msgproc


start
masterq name
scan mode
debug level
get arguments from user command; lock msgproc queue set working directory
read message from msgproc input queue
msgprocq
OK
call PROCESS function to process message by type
error
copy message to archive
archives
move message to error queue
error queue
record error in log file
log file
yes
more messages in input queue to be processed ?
no
no
"one pass" mode or shutdown command recieved
yes
exit

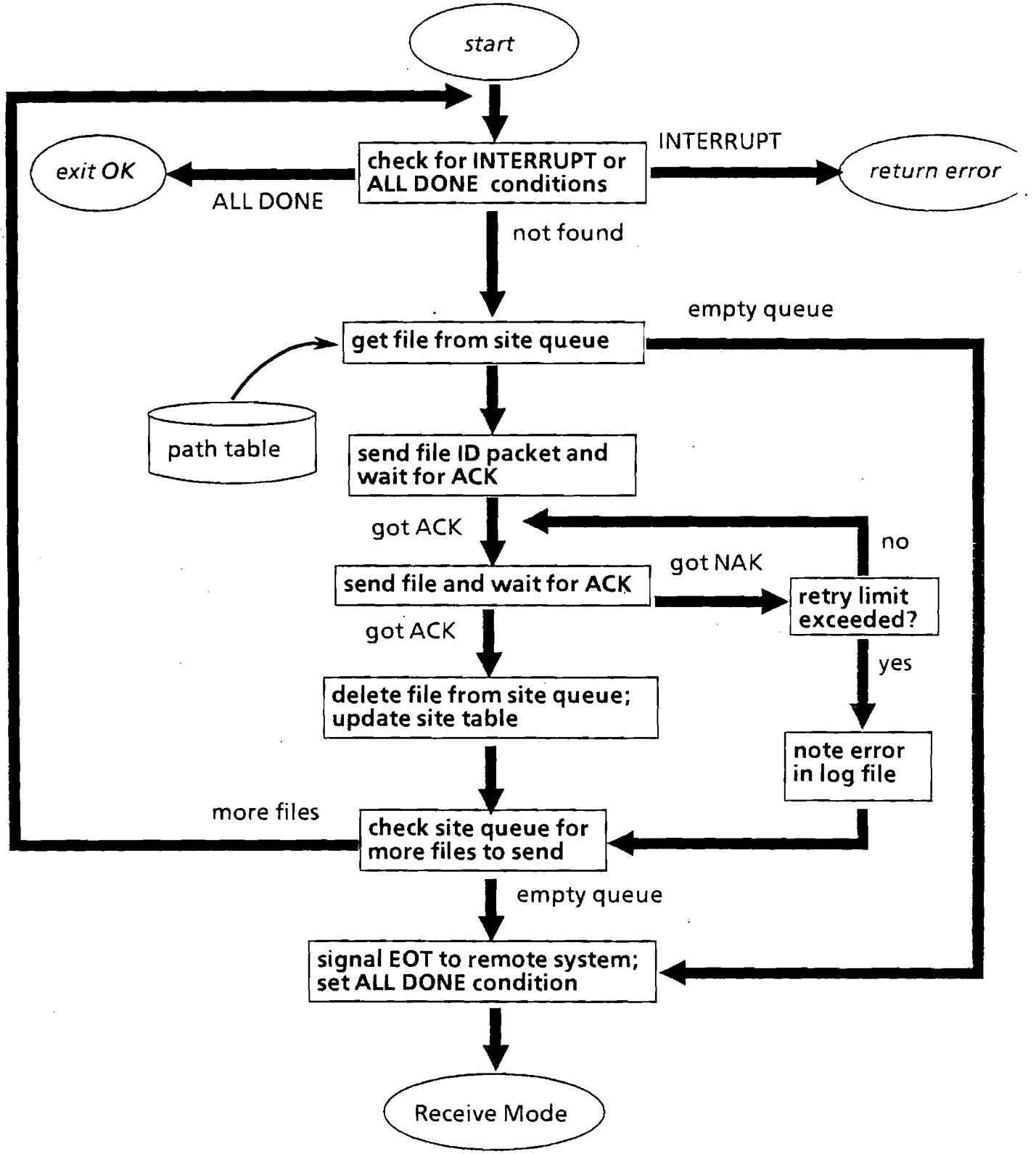
Process


start
message name
open message file and read destination path from message header
message file
path table
site table
expand path if necessary, then check for validity
invalid path
return error
path OK
routine message for remote site
put message in appropriate system queue
new message in system
check priority level and mark message accordingly
user mail message for user at this site
give message to UNIX mail facility
message to be forwarded
make courtesy copy for original recipient and forward to alternate site
network administrative message
give message to network admin server
priority message
put message in priority queue
message for database server
give message to C2 database server
unknown message type
return OK
return error
return OK

IOControl


start
arguments from CALLER
check argument list to determine mode of operation
no arguments
site name
port name
read arguments and initialize in MASTER mode
initialize in SLAVE mode
fail
wait for ENQ and acknowledge
error exit
succeed
send ENQ, then wait for ACK
fail
error exit
succeed
fail
send ID packet and wait for ACK
error exit
succeed
get remote ID packet, send ACK
fail
error exit
succeed
update site table to show successful contact with remote
all done
Send Mode
more files
more files
Receive Mode
exit
more files
Receive Mode
more files
all done
Send Mode

Send Mode


start
check for INTERRUPT or ALL DONE conditions
INTERRUPT
return error
exit OK
ALL DONE
not found
empty queue
get file from site queue
path table
send file ID packet and wait for ACK
got ACK
no
got NAK
send file and wait for ACK
retry limit exceeded?
got ACK
yes
delete file from site queue; update site table
note error in log file
more files
check site queue for more files to send
empty queue
signal EOT to remote system; set ALL DONE condition
Receive Mode

Receive Mode