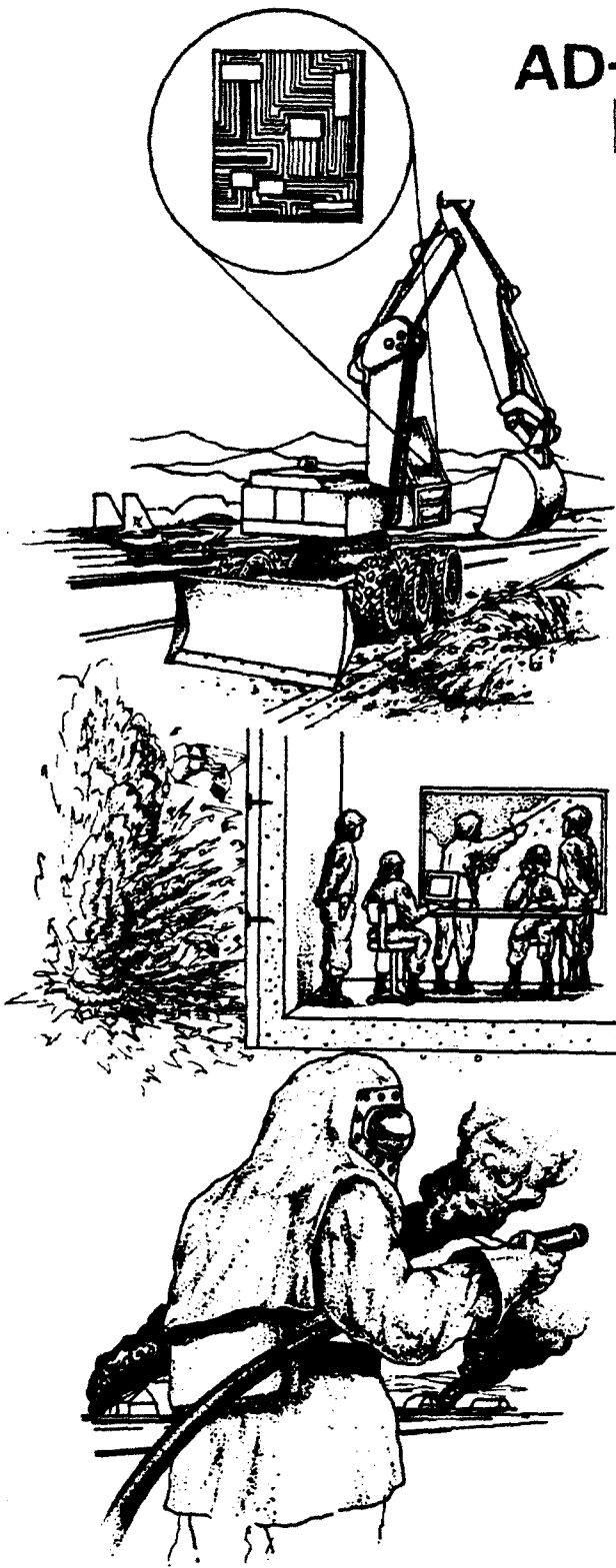


AD-A270 121



ESL-TR-91-22
Volume X

1



**THE POST-DAM SYSTEM
VOLUME X - INTEGRATED
DATABASE MANAGEMENT AND
PROJECT SCHEDULING SYSTEM
FOR POSTATTACK DAMAGE
ASSESSMENT**

M.D. SMITH, M.C. SAWYER, R.H. SUES

**APPLIED RESEARCH ASSOCIATES, INC,
POST OFFICE BOX 40128
TYNDALL AFB FL 32403**

OCTOBER 1992

FINAL REPORT

JUNE 1991 - APRIL 1992

**APPROVED FOR PUBLIC RELEASE:
DISTRIBUTION UNLIMITED**

**DTIC
SELECTED
SEP 30 1993
S B D**

93-22657



**ENGINEERING RESEARCH DIVISION
Air Force Civil Engineering Support Agency
Civil Engineering Laboratory
Tyndall Air Force Base, Florida 32403**



NOTICE

PLEASE DO NOT REQUEST COPIES OF THIS REPORT FROM HQ AFCESA/RA (AIR FORCE CIVIL ENGINEERING SUPPORT AGENCY). ADDITIONAL COPIES MAY BE PURCHASED FROM:

**NATIONAL TECHNICAL INFORMATION SERVICE
5285 PORT ROYAL ROAD
SPRINGFIELD, VIRGINIA 22161**

FEDERAL GOVERNMENT AGENCIES AND THEIR CONTRACTORS REGISTERED WITH DEFENSE TECHNICAL INFORMATION CENTER SHOULD DIRECT REQUESTS FOR COPIES OF THIS REPORT TO:

**DEFENSE TECHNICAL INFORMATION CENTER
CAMERON STATION
ALEXANDRIA, VIRGINIA 22314**

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188	
1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b. RESTRICTIVE MARKINGS		
2a. SECURITY CLASSIFICATION AUTHORITY			3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release. Distribution unlimited.		
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE					
4. PERFORMING ORGANIZATION REPORT NUMBER(S) 5691			5. MONITORING ORGANIZATION REPORT NUMBER(S) ESL-TR-91-22 Volume X		
6a. NAME OF PERFORMING ORGANIZATION Applied Research Associates, Inc.		6b. OFFICE SYMBOL (If applicable)		7a. NAME OF MONITORING ORGANIZATION HQ AFCEA/RACS Tyndall AFB FL 32403	
6c. ADDRESS (City, State, and ZIP Code) Southeast Division 6404 Falls of Neuse Road, Suite 200 Raleigh, North Carolina 27615			7b. ADDRESS (City, State, and ZIP Code)		
8a. NAME OF FUNDING / SPONSORING ORGANIZATION Air Force Civil Engineering Support Agency		8b. OFFICE SYMBOL (If applicable) RACS		9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
8c. ADDRESS (City, State, and ZIP Code) Air Base Survivability Branch, Engineering Research Division Air Force Civil Engineering Support Agency, HQ AFCEA/RACS Tyndall Air Force Base, FL 32403-6001			10. SOURCE OF FUNDING NUMBERS		
			PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.
			WORK UNIT ACCESSION NO.		
11. Title (Include Security Classification) Integrated Database Management and Project Scheduling System for Postattack Damage Assessment					
12. PERSONAL AUTHOR(S) Smith, M. D., Sawyer, M. C., and Sues, R. H.					
13a. TYPE OF REPORT Technical		13b. TIME COVERED FROM 6/1/91 TO 4/15/92		14. DATE OF REPORT (Year, Month, Day) October 1992	
15. PAGE COUNT					
16. SUPPLEMENTARY NOTATION					
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUB-GROUP	Damage Assessment, ABO, POSTDAM, Expert System, Resource Manager/Repair Scheduler (RMRS)		
19. ABSTRACT (Continue on reverse if necessary and identify by block number)					
In a postattack environment, field information on mission-critical facility damage is collected and analyzed to determine structural integrity and usability. From this analysis, a repair schedule is developed. This process is time-consuming when unaided by a computerized system. Consequently, the POST-DAM system was developed to determine repair strategies using an expert system; keep track of materials and equipment using a relational database management system; and schedule repairs based on manpower and equipment availability using a project management system. This report addresses the combination of the relational database management system and the project management system of the original POST-DAM architecture into one integrated RMRS system. The RMRS prototype software demonstrates an architecture that is able to quickly and efficiently manage Air Base resources and schedule expedient repairs for mission critical facilities. For repairs with no resource conflicts, the RMRS system operates with minimal user intervention. When conflicts are present, resolution is possible by allowing the user direct access to the RMRS data files and complete control of the resource allocation and scheduling processes. User interaction is completely menu-driven, with context-sensitive help to minimize training requirements and maximize system integrity.					
20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED / UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED		
22a. NAME OF RESPONSIBLE INDIVIDUAL MAJ. JACOB GHERIANI			22b. TELEPHONE (Including Area Code)		22c. OFFICE SYMBOL HQ AFCEA/RACS

DD FORM 1473, JUN 86

Previous editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

EXECUTIVE SUMMARY

A. OBJECTIVE

This report describes the software developed by Applied Research Associates, Inc. for the host computer portion of the airbase facility postattack damage assessment (POST-DAM) system. The objective of this research was to develop an integrated prototype software system to manage repair resources and schedule repairs based on resource availability and facility priority. This software system, entitled Resource Manger/Repair Scheduler (RMRS), was design to be user friendly and require minimal user interaction.

B. BACKGROUND

In a postattack environment, field information on mission-critical facility damage is collected and analyzed to determine structural integrity and usability. From this analysis, a repair schedule is developed. This process is time-consuming when unaided by a computerized system. Consequently, the POST-DAM system was developed to determine repair strategies using an expert system; keep track of materials and equipment using a relational database management system; and schedule repairs based on manpower and equipment availability using a project management system. This report addresses the combination of the relational database management system and the project management system of the original POST-DAM architecture into one integrated RMRS system. The original POST-DAM work is documented in a nine-volume set. Volume I describes software and hardware used with the prototype POST-DAM system, and recommends software and hardware for full-scale development. Volumes II through VIII are software user's manuals that describe how to install and use the prototype software with the POST-DAM system. Volume IX is a field manual that contains diagrams of structures that are used with the POST-DAM system to locate damaged elements.

C. CONCLUSIONS

The RMRS prototype software demonstrated an architecture that was able to quickly and efficiently manage Airbase resources and schedule expedient repairs for mission critical facilities. For repairs with no resource conflicts, the RMRS system could operate with minimal user intervention. When conflicts were present, resolution was possible by allowing the user direct access to the RMRS data files and complete control of the resource allocation and scheduling processes.

D. RECOMMENDATIONS

For full-scale development of the RMRS software, the following issues should be addressed:

1. Include a communications module to allow RMRS to receive PDES data files in the background while continuing to work uninterrupted.
2. Incorporate Expert System capabilities to provide RMRS the capability to suggest resource conflict solutions and changes in repair strategies based on current resource supplied and equipment usage.
3. Develop an optimized scheduler to generate schedules based on facility priority and time and resource optimization.

PREFACE

This report was prepared by Applied Research Associates, Inc. (ARA), 6404 Falls of Neuse Road, Suite 200, Raleigh, NC 27615, under Contract F08635-88-C-0067, for the Air Force Civil Engineering Support Agency, Tyndall Air Force Base, Florida.

This report (Volume X) summarizes work completed between 1 June 1991 and 31 December 1991. Lt. James Underwood (USN) and Maj. Jacob Gheriani (IAF) were the HQ AFCESA/RACS Project Officers.

This report has been reviewed by the Public Affairs Office, and is releasable to the National Technical Information Service (NTIS). At NTIS it will be available to the public, including foreign nations.

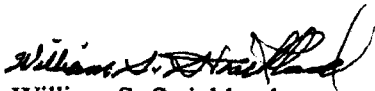
This technical report has been reviewed and is approved for publication.



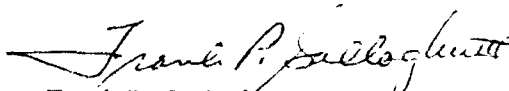
Dan Shenbach, Maj., IAF
Project Officer



Felix Uhlik, Lt. Col., USAF
Chief, Engineering Research Division



William S. Strickland
Chief, Airbase Survivability Branch



Frank P. Gallagher, III, Col., USAF
Director, Civil Engineering Laboratory

DTIC QUALITY INSPECTED 1'

Allocation For	
100-100000	☒
100-100000	☐
100-100000	☐
100-100000	☐

A-1

TABLE OF CONTENTS

Section	Title	Page
I	INTRODUCTION.....	1
A.	Objective.....	1
B.	Background.....	1
II	SYSTEM TECHNICAL DESCRIPTION.....	3
A.	General.....	3
B.	Database Schema.....	4
1.	FACTPRT.DBF.....	4
2.	EQPSUP.DBF.....	4
3.	MATSUP.DBF.....	5
4.	REPAIR.DBF.....	5
5.	EQPREQ.DBF.....	6
6.	MATREQ.DBF.....	7
7.	REPINFO.DBF.....	9
C.	Data Flow.....	9
1.	General.....	9
2.	New Status.....	9
3.	Possible Status.....	11
4.	Queued Status.....	11
5.	Complete Status.....	12
D.	Resource Allocation.....	12
E.	Repair Scheduling.....	13
III	COMMAND REFERENCE.....	15
A.	SYSTEM menu (Alt-Y).....	15
B.	INIT/SETUP menu (Alt-I).....	17
C.	PROCESS menu (Alt-P).....	17
D.	SCHEDULE menu (Alt-S).....	19
E.	CHANGE STATUS menu (Alt-C).....	21
F.	GEN REPORT menu (Alt-R).....	21
IV	EXAMPLE SESSION.....	25
A.	REPAIR LOADING/PROCESSING.....	25
B.	SCHEDULE REPAIRS.....	27
C.	REPORT GENERATION.....	28
V	CONCLUSIONS AND RECOMMENDATIONS.....	31
VI	REFERENCES.....	33
Appendix		
A	INSTALLATION PROCEDURE for RMRS.....	35
B	RMRS SOURCE CODE.....	37

LIST OF FIGURES

Figure	Title	Page
1	Early and New POST-DAM System Architectures	2
2	Repair Status Values.....	10
3	System Menu.....	16
4	About Screen	16
5	Edit Data Menu.....	17
6	Init/Setup Menu	18
7	Process Menu	18
8	Compromise Dialog Box.....	20
9	Schedule Menu.....	20
10	Change Status Menu.....	22
11	Gen Report Menu.....	22
12	Startup Screen	25
13	<i>Process / Auto Mode Command</i>	26
14	<i>Schedule / Full Schedule Command</i>	28
15	<i>Gen Report / Gantt Chart</i> Menu Choice	29
16	Gantt Chart Report.....	29

LIST OF TABLES

Table	Title	Page
1	RESOURCE AVAILABILITY ANALYSIS: REPID #105.....	27

SECTION I

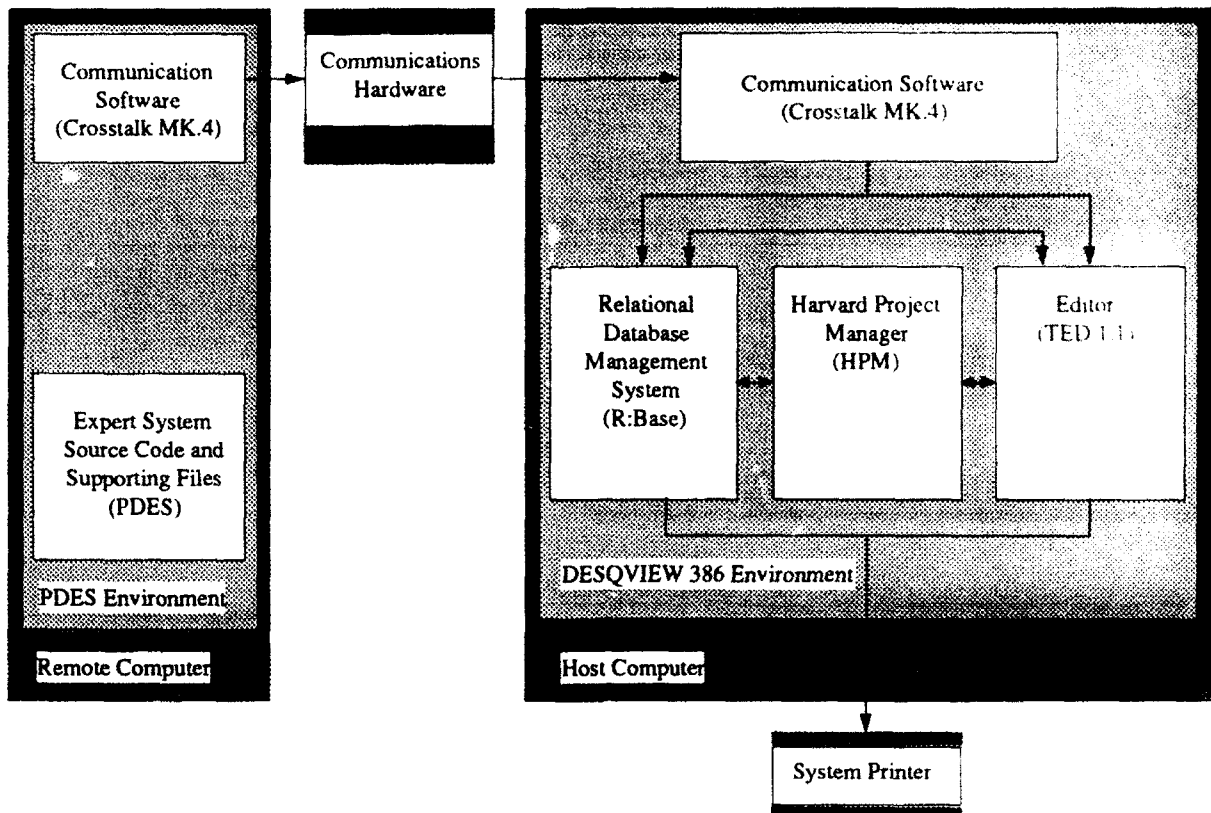
INTRODUCTION

A. OBJECTIVE

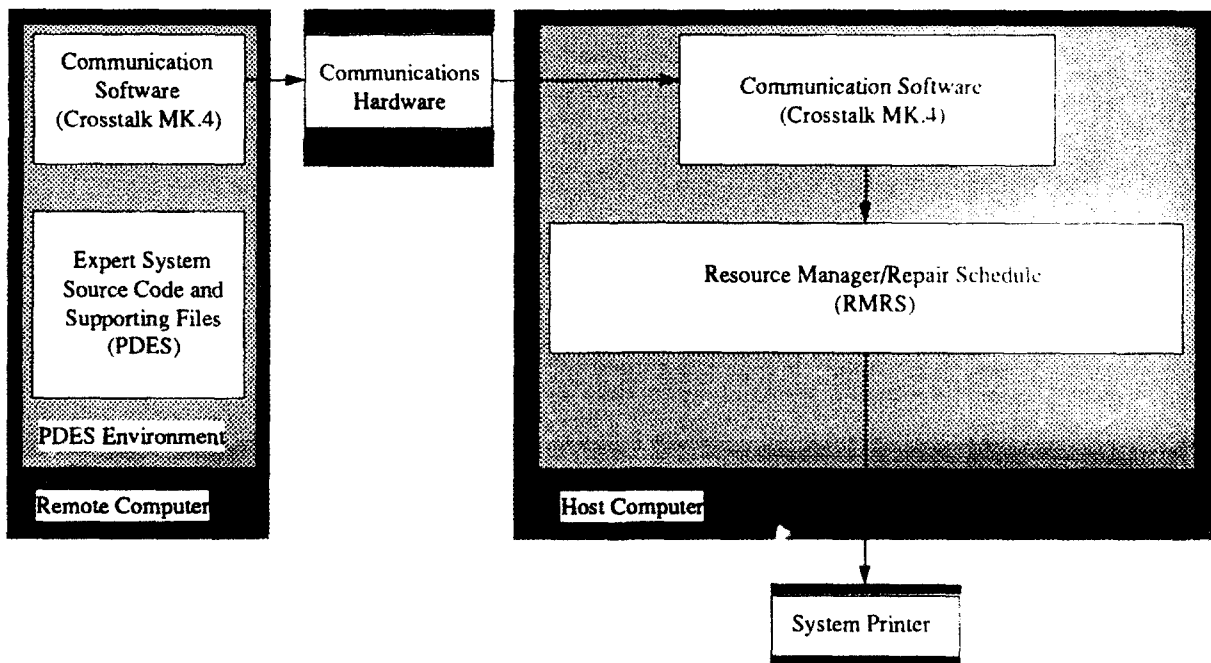
Our objective is to develop a personal computer-based software system to aid base engineers in postattack facility damage assessment (POST-DAM) of mission-critical facilities. This system is to integrate a database manager to track airbase material and equipment resources and a facility priority-based repair scheduler. The result of this effort is the Resource Manager/Repair Scheduler (RMRS) host system, which works in conjunction with the previously developed POST-DAM expert System (PDES).

B. BACKGROUND AND SCOPE

The RMRS host module was developed in response to limitations identified during testing of the early POST-DAM System prototype. In its original form, the host system portion of POST-DAM consisted of four separate commercial off-the-shelf (COTS) software packages and required more user interaction and training than was desired. Thus, a custom application was developed that combined the resource supply management function of the relational database manager and the repair scheduling function of the project scheduler. The RMRS module was designed for optimal host system automation, requiring user intervention only under special conditions, while allowing the host system operator complete control over resource usage and repair scheduling. Figure 1 shows the difference in the POST-DAM architecture with RMRS.



a. Early POST-DAM System Architecture.



b. New POST-DAM System Architecture.

Figure 1. Early and New POST-DAM Architectures.

SECTION II

SYSTEM TECHNICAL DESCRIPTION

A. GENERAL

RMRS is a tool to aid Damage Control Center (DCC) personnel in efficiently allocating resources and schedule expedient repairs to mission-critical Air Force base facilities following an enemy attack. The RMRS module operates as part of the Postattack Damage Assessment System (POST-DAM).

The RMRS module was developed using the C Language and, when possible, third party libraries for such functions as the low-level database file management, user interface, etc. It is menu driven with context sensitive help and mouse support. A major design criterion was that the system operate with minimal user interaction. To this end, the RMRS module can generate repair schedules and perform the following functions with as few as three commands:

Command: PROCESS | AUTO MODE ¹

- Retrieves POST-DAM EXPERT SYSTEM (PDES) from host hard disk storage.
- Creates entry in RMRS database for each repair.
- Compares equipment requirements for repair with available resources.
- If the equipment resources are available, RMRS will continue; otherwise RMRS will let the user know that this repair is not possible.
- Compares material requirements for repair with available resources.
- If the resources are available, RMRS will allocate the resources to the repair; otherwise RMRS will enter an interactive mode to allow the user to resolve the resource conflict.

Command: SCHEDULE | FULL SCHEDULE

- Makes list of all repairs to be scheduled in facility priority order.
- For each repair, finds the earliest time slot available for all equipment resources required and allocates this time slot to the repair.

Command: GEN REPORT | REPAIR SCHEDULE

- Based on results of the schedule process, outputs schedule information in desired format.

¹This terminology **PROCESS | AUTO MODE** refers to the *Process* menu and the *Auto Mode* choice.

B. DATABASE SCHEMA

The RMRS system is organized around seven data files, or databases, that maintain the required information to track and schedule materials, equipment, and repairs.

The seven databases are:

1. FACPRT.DBF - Airbase facility priority listing.
2. EQPSUP.DBF - Airbase equipment supply listing.
3. MATSUP.DBF - Airbase materials supply listing.
4. REPAIR.DBF - status records for each individual repair.
5. EQPREQ.DBF - records for each equipment requirement associated with repairs.
6. MATREQ.DBF - records for each material requirement associated with repairs.
7. REPINFO.DBF - misc. info records for each individual repair.

The first three databases are populated from existing Airbase-specific data in the system initialization phase. The remaining four databases are populated with the data that is received from the PDES. Each processed repair will have at least one related record in the REPAIR.DBF, EQPREQ.DBF, MATREQ.DBF, and REPINFO.DBF databases (excluding the special case where a repair does not require any materials, in which case, no material record would exist in MATREQ.DBF for that repair). The following sections describe the format of and the information contained in each of the RMRS databases.

1. FACTPRT.DBF

The database file FACPRT.DBF correlates the Airbase facility designations to their corresponding mission-critical priority rating.

FIELD#	FIELD NAME	FIELD DESC
1	FACNUM	Aibase facility number designation
2	PRIORITY	Aibase-assigned mission critical facility priority

2. EQPSUP.DBF

EQPSUP.DBF represents the current Airbase equipment supply. Only large equipment, such as bulldozers, shotcrete machines, etc., is included in this database. Smaller equipment, such as hammers, screwdrivers, etc., is considered to be standard equipment carried by all damage repair teams. Additionally, damage repair teams, containing approximately five members, are included in this database. The record format consists of a unique equipment identification number and a textual description of the equipment piece.

FIELD#	FIELD NAME	FIELD DESC
1	EQPID	RMRS system-assigned equipment id number
2	EQPDESC	Textual equipment description

3. MATSUP.DBF

MATSUP.DBF represents the current unallocated Airbase material supply. The record format consists of a unique material identification number, a textual description of the material, the total quantity of the material available, and a textual "units" description (e.g., ea, gal).

FIELD#	FIELD NAME	FIELD DESC
1	MATID	RMRS system-assigned material id number
2	MATDESC	Textual material description
3	QTY	Quantity of unallocated material units resident in Airbase supply
4	UNIT	Textual "units" description (e.g., ea, gal)

4. REPAIR.DBF

The REPAIR.DBF database consists of records (one per repair) containing identification and status data for each individual repair processed by the RMRS system. The record format consists of five fields, as follows:

FIELD#	FIELD NAME	FIELD DESC
1	REPID	RMRS system-assigned repair id number
2	FACNUM	Airbase facility number designation
3	PRIORITY	Airbase-assigned mission critical facility priority
4	SCHDPRTY	RMRS system scheduling priority (initially equal to (3) selectable)
5	STATUS	RMRS system repair status

The "repair id" (Field 1) is assigned sequentially by the RMRS system at repair load-time by the RMRS system (initial phase of PROCESS cycle). Although the "repair id" is mainly used internally by RMRS, it is provided to the user along with other repair information when appropriate. Most importantly, this "repair id" provides the required link between the repair and its associated resource requirements and allocations.

The "facility number designation" (Field 2) is populated at repair load-time. The facility number is extracted from the PDES output file name (e.g., B4058.EQP -> 4058).

The "mission critical facility priority" (Field 3) is the priority corresponding to the Airbase priority listing for mission critical facilities. This data is retrieved from the facility priority database, FACPRT.DBF.

The "scheduling priority" (Field 4) allows user control over repair ordering in the prioritized RMRS scheduler queue. By default, the "mission critical facility" and "scheduling priority" are the same, the scheduler *always* looks at the "scheduling priority" when ordering the queue. Therefore, the user can manipulate the scheduling order by using the SET PRIORITY menu function to change the value in Field 4, while maintaining the original priority in Field 3.

The "RMRS system repair status" (Field 5) is denoted by a single character designation that indicates the repair status as related to the RMRS processing cycle. The possibilities are as follows:

- N - (N)ew repair
- P - (P)ossible repair
- Q - (Q)ueued repair
- C - (C)ompleted repair
- O - (O)verriden repair
- S - (S)uspended repair
- X - (X)anceled repair

5. EQPREQ.DBF

The EQPREQ.DBF database consists of records (one or more per repair) containing equipment requirement and time-slot allocation information associated with a repair. Consequently, the actual schedule information is embedded in this database. Each individual equipment requirement for a repair is assigned a record that is divided into six fields:

FIELD#	FIELD NAME	FIELD DESC
1	EQPID	Equipment id number
2	EQPDESC	Textual equipment description
3	START	Scheduled repair start time (in seconds referenced from Greenwich Mean Time (00:00:00 1/1/70))
4	DURATION	Estimated repair duration in seconds
5	REPID	Associated RMRS repair id number
6	STATUS	RMRS system allocation/repair status

The "equipment id" (Field 1) corresponds to the RMRS system-assigned number used in the Airbase equipment supply database, EQPSUP.DBF.

The "equipment description" (Field 2) is populated at repair load-time with the corresponding entry from the *.EQP PDES output facility file. The RMRS system currently uses this field as the search key when checking for equipment existence in the Airbase equipment supply database, EQPSUP.DBF.

The "scheduled repair start time" (Field 3) is set when a repair is scheduled. Its value is represented in seconds, referenced from Greenwich Mean Time (GMT) 00:00:00 1/1/70. Every equipment requirement record for a given scheduled repair will have the same start time value.

The "estimated repair duration" (Field 4) is populated at repair load-time with the corresponding entry from the *.EQP PDES output facility file. Its value is represented in seconds.

The "repair id" (Field 5) provides the link between the particular equipment requirement and its corresponding repair record in REPAIR.DBF.

The "RMRS system scheduling/repair status" (Field 6) is denoted by a single character designation that indicates the equipment requirement status as related to the RMRS equipment scheduling/repair processing scheme. The possibilities are as follows:

- N - associated with (N)ew repair.
- Q - (Q)ueued (associated with UNLOCKED (Q)ueued repair).
- L - (L)ocked (associated with LOCKED (Q)ueued repair).
- C - associated with (C)ompleted repair.
- O - associated with (O)verriden repair.
- S - associated with (S)uspended repair.
- X - associated with (X)anceled repair.

6. MATREQ.DBF

MATREQ.DBF consists of records (zero or more per repair) containing material requirement and allocation status information associated with a repair. Each individual material requirement for a repair is assigned a record that is divided into six fields:

FIELD#	FIELD NAME	FIELD DESC
1	MATID	Material id number
2	MATDESC	Textual material description
3	QTY	Quantity required
4	UNIT	Textual "units" description
5	REPID	Associated RMRS repair id number
6	STATUS	RMRS system allocation/repair status

The "material id" (Field 1) corresponds to the RMRS system-assigned number used in the Airbase material supply database, EQPSUP.DBF.

The "material description" (Field 2) is populated at repair load-time with the corresponding entry from the *.MAT PDES output facility file. The RMRS system currently uses this field as the search key when checking for material existence in the Airbase equipment supply database, MATSUP.DBF.

The "quantity required value" (Field 3) is populated at repair load-time with the corresponding entry from the *.MAT PDES output facility file.

The "'units' description" (Field 4) is populated at repair load-time with the corresponding entry from the *.MAT PDES output facility file. This value is not currently used by the RMRS system except as information output.

The "repair id" (Field 5) provides the link between the particular material requirement and its corresponding repair record in REPAIR.DBF.

The "RMRS system allocation/repair status" (Field 6) is denoted by a single character designation that indicates the material requirement status as related to the RMRS material allocation/repair processing scheme. The possibilities are as follows:

- A - (A)llocated (associated with (P)ossible or UNLOCKED (Q)ueued repair).
- L - (L)ocked (associated with LOCKED (Q)ueued).
- U - (U)sed (associated with (C)ompleted repair).
- N - (N)ew (associated with (N)ew repair).
- O - associated with (O)verriden repair.
- S - associated with (S)uspended repair.
- X - associated with (X)anceled repair.

7. REPINFO.DBF

REPINFO.DBF consists of records (one per repair) containing detailed information associated with a repair. The information is assembled from the data contained in the *.OUT PDES output file. Each individual repair is assigned a repair information record that consists of six fields:

FIELD#	FIELD NAME	FIELD DESC
1	REPID	Associated RMRS repair id number
2	FACFUN	Facility function
3	FACDESC	Facility description
4	ELENUM	Element number
5	ELEDESC	Element description
6	DAMMODE	Damage mode
7	DAMW	Damage width
8	DAML	Damage length
9	DAMH	Damage height
10	REPSTGY	Repair strategy
11	REMARKS	Remarks

The "repair id" (Field 1) provides the link between the particular repair information record and its corresponding repair record in REPAIR.DBF.

Fields #2-#11 are self-explanatory.

C. DATA FLOW

1. General

As a repair moves through the RMRS system it progresses from one stage, or status, to the next. Although conceptually a repair "moves" through the system, repair data is not actually copied from one place to another. Once repair data is initially recorded into the appropriate databases, the repairs status changes via the value of its status field as shown in Figure 2.

2. New Status

When the PDES output facility files (*.EQP, *.MAT, *.OUT) are loaded into the four repair databases, file format validation is performed and the new repair data records are

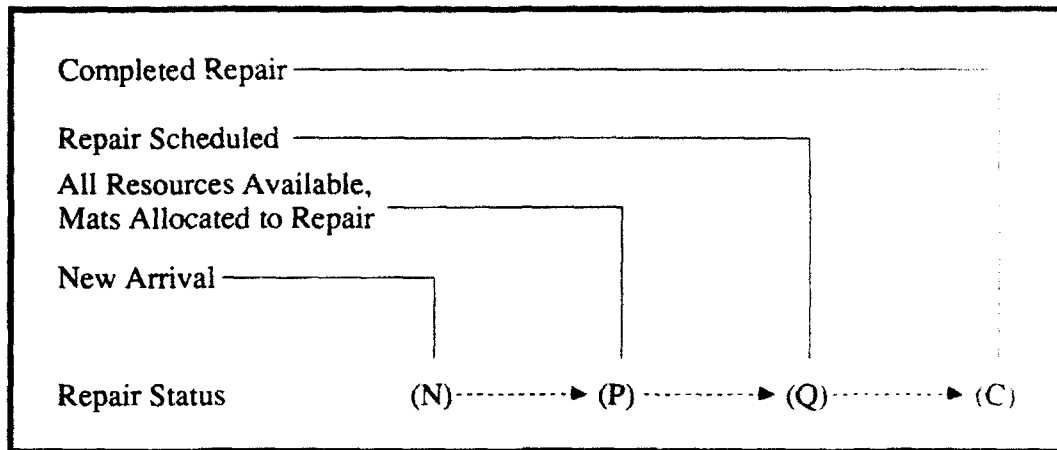


Figure 2. Repair Status Values.

marked with the (N)ew status. Typically the next status for a repair after (N)ew is (P)ossible, however there are other possibilities based on existing conditions and user decisions. Listed below are the status transitions that can occur from (N)ew:

(N)ew -> (X)anceled : NONEXISTENT EQUIPMENT. Repair required on a non-existent piece of equipment (for supply purposes, e.g., it was destroyed in the attack), the repair would be "automatically" (X)anceled by the RMRS system (RMRS assumes that only material conflicts can be resolved through substitution, equipment conflicts are not resolvable, however the user may perform manual resolutions).

(N)ew -> (S)uspended : ONLY MATERIAL CONFLICTS. Repair has one or more material requirement conflicts. A material conflict arises when a material proposed by the PDES module is either non-existent (no corresponding entry in Airbase material supply listing) or the Airbase material supply lacks sufficient quantities to satisfy the requirement. The RMRS system incorporates provisions to resolve material conflicts through substitution, according to the user's discretion. However, the user can defer this resolution opportunity to a later time by choosing not to attempt a conflict compromise. In this case, the RMRS system would (S)uspend the repair.

(N)ew -> (P)ossible : NO RESOURCE CONFLICTS. This status progression occurs when no resource conflicts arise.

3. Possible Status

A (P)ossible status indicates that the repair is ready for scheduling, the required equipment resources are available, and the required material resources have been allocated.

(P)ossible -> (Q)ueued : NORMAL SCHEDULING. This status progression occurs upon normal scheduling of the repair.

(P)ossible -> (S)uspended : MANUAL SUSPENSION. This status progression occurs when, before being scheduled, a repair with no resource conflicts is selected for suspension by the user.

(P)ossible -> (O)verriden : PRIORITY OVERRIDE RESOLUTION. This status progression occurs when, before this particular repair was scheduled, a different, higher-priority repair needs materials that are not available in the supply, but can be taken from this (P)ossible repair. In this case, the RMRS system asks the user for override permission which, if granted, will cause the required materials to be released to the high-priority repair, while the other materials from the overridden repair return to the material supply.

(P)ossible -> (X)cancel : MANUAL CANCELTION. This status progression occurs when, before being scheduled, a repair with no resource conflicts is cancelled by the user.

4. Queued Status

A queued status indicates that the repair has been scheduled and that the required equipment resources have been reserved for the appropriate time.

(Q)ueued -> (C)omplete : NORMAL COMPLETION. This status progression occurs upon normal completion of the repair.

(Q)ueued -> (S)uspended : MANUAL SUSPENSION. This status progression occurs when, before being started, a scheduled repair is selected for suspension by the user.

(Q)ueued -> (O)verriden : PRIORITY OVERRIDE RESOLUTION APPRVD. This status progression occurs when, before this particular repair begins, a different, higher-priority repair needs materials that are not available in the supply, but can be taken from this (Q)ueued repair. In this case, the RMRS system asks the user for override permission which, if

granted, will cause the required materials to be released to the high priority repair while the other materials from the overridden repair return to the material supply.

(Q)ueued -> (X)cancel : MANUAL CANCELATION. This status progression occurs when, before being started, a scheduled repair is cancelled by the user.

5. Complete Status

Once a repair has acquired the **(C)omplete** status it remains in the database files until explicitly deleted or the database files are reset.

D. RESOURCE ALLOCATION

Resource allocation by the RMRS system is accomplished automatically through software manipulation of the Airbase supply databases and the repair resource requirement databases. The two resource types, equipment and material, are allocated in separate phases of the RMRS processing cycle due to the different concerns for each type of resource. For material resources we are mainly concerned with quantity available; while for equipment resources we are concerned with the equipment allocation over time. The *Process* menu commands accomplish the material resource allocation, while the *Schedule* menu commands handle the allocation of equipment resources.

Material resource allocation is automatic through the normal execution of the RMRS *Process* commands: (1) Auto mode; (2) Single facility; and (3) Compromise retry.

The *Process / Auto Mode* and *Process / Single Facility* commands involve the processing of newly arrived PDES facility repair file sets. This processing includes database population, resource availability analysis, and resource requirement compromise, if needed. The *Process Compromise Retry* command involves availability analysis and compromise. Regardless of the command used, the overall objective here is allocate material resources to a repair if no conflicts occur or if those occurring can be resolved.

The *Process* menu commands work on an "all or nothing" resource availability analysis scheme. This scheme verifies that all the resource requirements can be satisfied before resource allocation actually occurs since, for most repair strategies, it is unlikely that the repair could be satisfactorily completed with only a fraction of the required resources. Thus, the *Process* menu commands verify that the required equipment exists and then that the appropriate material quantities

are available. Then, upon no conflicts or upon resolution thereof, the materials are allocated to that repair. The equipment is allocated at schedule time.

The RMRS system requirement of prioritized resource allocation is provided for in two ways. First, all newly arrived PDES facility repair file sets are processed through resource availability analysis in priority order. If all resources are readily available from the Airbase supply databases, then the material resources are allocated at that time. Second, when a material conflict occurs because of inadequate supply, the RMRS system turns to the material requirements database (MATREQ.DBF) to attempt to locate a priority-override resolution. In other words, the search tries to locate a material requirement that has sufficient quantity and is associated with a repair that has a lower priority than that repair looking for resources. This search only includes the matching material requirements for repairs that have not been started yet. Specifically, these qualifying material requirements will have a STATUS designation of (A)llocated. In contrast, unallocated material requirements will be marked (S)uspended or (X)anceled, and those associated with a completed repair are marked (U)sed. When used in conjunction, these two methods assure that the material resources are efficiently distributed in a prioritized fashion, as required.

On the data manipulation level, material allocation involves both the Airbase material supply database (MATSUP.DBF) and the material requirements database (MATREQ.DBF). In a no-conflict scenario, the material quantities specified in the repair's material requirement records are simply deducted from the QTY field in the AB material supply database, and the corresponding material requirement records are given a STATUS of (A)llocated. Of course, a side effect of this transaction is that the associated repair identification record (in REPAIR.DBF) has its STATUS field changed to (P)ossible. Whereas, with the override resolution scenario, the overridden repair is dropped from its scheduled status of (Q)ueued. This must be the case, because once the required override materials have been taken from the repair, it no longer has access to all the material resources it needs. Upon demotion of the subordinate repair, all of its allocated material resources are first returned to the AB material supply by adding the requirement quantities to the QTY fields in the corresponding supply entries. The overridden repair and its associated resource requirement records are given a STATUS of (O)verridden. Then the materials are allocated to the high-priority repair in the normal way.

E. REPAIR SCHEDULING

Once resource allocation has been successfully completed through the use of one of the Process menu commands, the REPAIR.DBF database file should contain one or more records with the status field set to (P)ossible. A (P)ossible status indicates that the repair has been allocated

material resources and is ready to be scheduled. Remember that the resource allocation cycle verified that the equipment required for each repair existed, therefore the only concern at this point is to find the earliest common time that all pieces of equipment required for the repair will be available. When the time slot is found, the status field in REPAIR.DBF is set to "Q" for "queued," which means that the repair has been scheduled and equipment allocated. Also the status, start, and duration fields in the EQPREQ.DBF file are set to the appropriate values to indicate that the time slot for that piece of equipment has been allocated.

As an example of the way RMRS schedules repairs, consider the case where we schedule one repair with a duration of one hour that requires the following pieces of equipment: one Shotcrete Machine, two Ramsets, and one Repair Team. Assume that other repairs have already been scheduled and that the current time is 1200 hours today. To begin the scheduling process RMRS will first determine the equipment requirements for the repair (which we already know from the problem statement) from the EQPREQ.DBF file. Next, the earliest start time for the repair is determined by adding an appropriate lag value to the current time, for example thirty minutes; therefore, we will not attempt to schedule this repair before 1230 hours today. Now we choose the first of the equipment pieces (the selection order is not important) and try to locate a one-hour time slot (one-hour duration repair) when this piece of equipment is available not before 1230 hours today (earliest start time). The beginning of the time slot we find is called the "equipment start time." We repeat the process for each piece of equipment. Depending on the current availability for each piece of equipment, the equipment start times may or may not be the same. If the start times are the same we have scheduled the repair, however if they are different we must try again. For the retry we set the earliest start time for the repair to the latest of all the equipment start times, and repeat the process. Eventually we will find a common time during which all of the equipment pieces required for the repair will be available. The above approach ensures that the common time is the earliest possible.

SECTION III

COMMAND REFERENCE

The Resource Manager/Repair Scheduler (RMRS) operates in a pull-down menu environment. All menu choices can be selected by mouse or via the keyboard. The system menu bar consists of the following six pull-down menus:

- (1) SYSTEM
- (2) INIT/SETUP
- (3) PROCESS
- (4) SCHEDULE
- (5) CHANGE STATUS
- (6) GEN REPORT

The following is a brief description of each menu bar selection:

A. SYSTEM MENU (ALT-Y) ²

This menu allows the user to access information about the software release and view/modify the raw data contained in the RMRS system databases, along with the conventional operating system access functions that allow DOS shell operations and program termination.

About. Choosing this command will cause the appearance of a dialog box that shows version information for the RMRS system software. Press any key to close the box.

Edit Data (Alt-E). The *System / Edit Data* command lets you view and/or modify the raw data contained in the seven RMRS databases. No special administrator privileges are needed to modify the databases at this time. However, since the integrity of the RMRS system output depends entirely on that of the data contained within these databases, future system enhancements should include some security mechanism that regulates the data modification privilege (e.g., a password). For a description of the features available in the datafile edit module consult the online help.

DOS shell (Alt-D). The *System / DOS Shell* command lets you temporarily exit the RMRS system to enter a DOS command or program. To return to the RMRS system, type 'EXIT' and press <ENTER>.

²Some menus and menu choices have short cut keys. These keys are indicated in parenthesis.

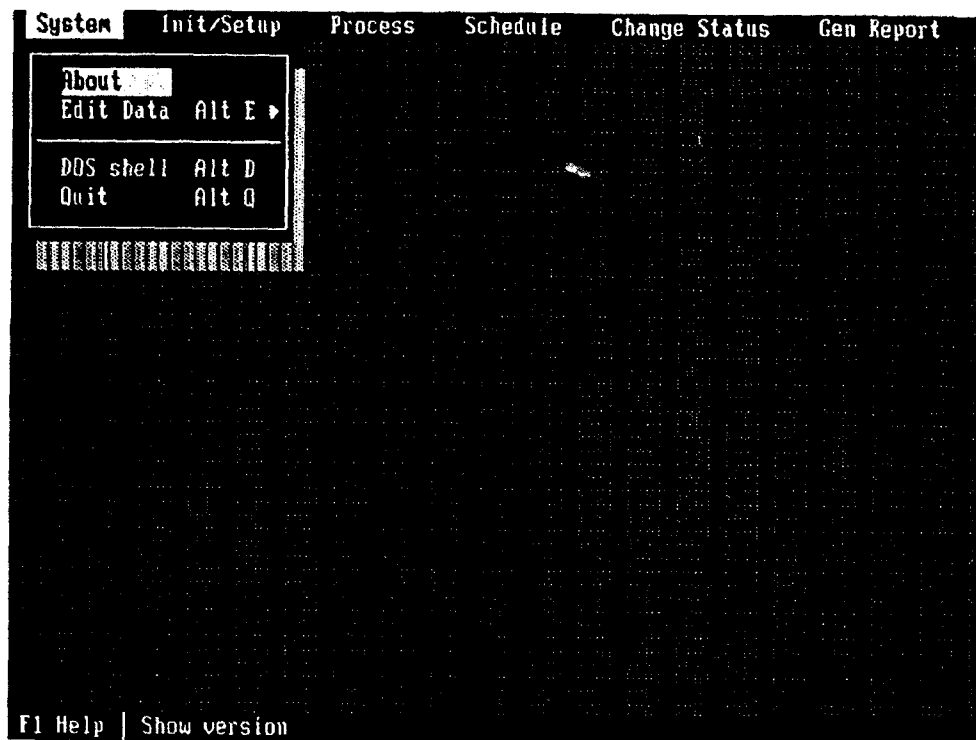


Figure 3. System Menu.

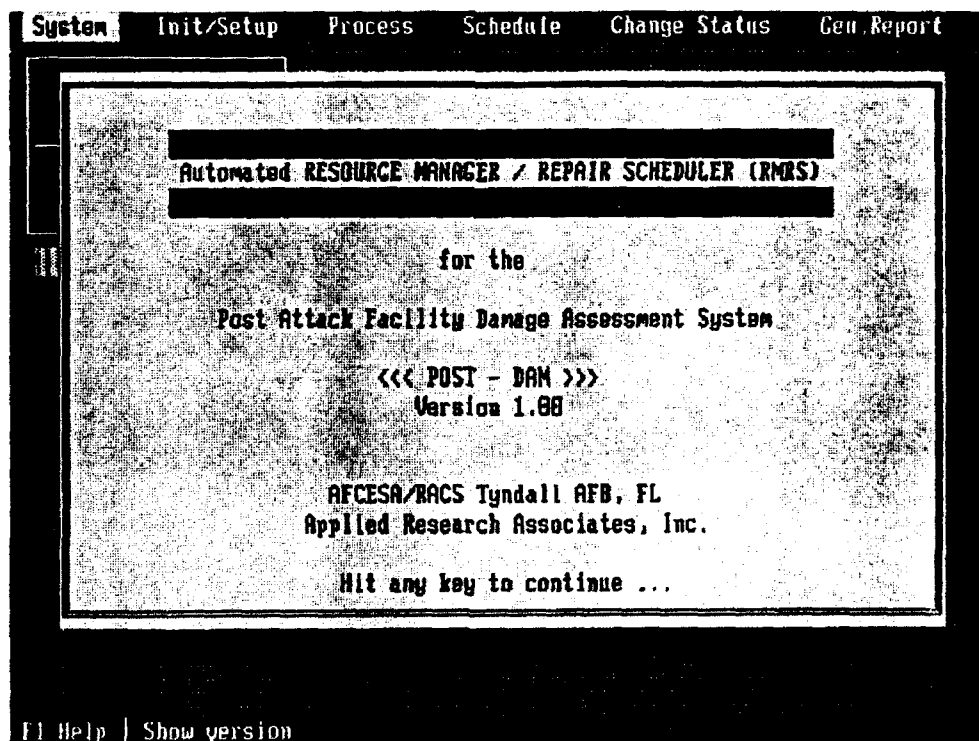


Figure 4. About Screen.

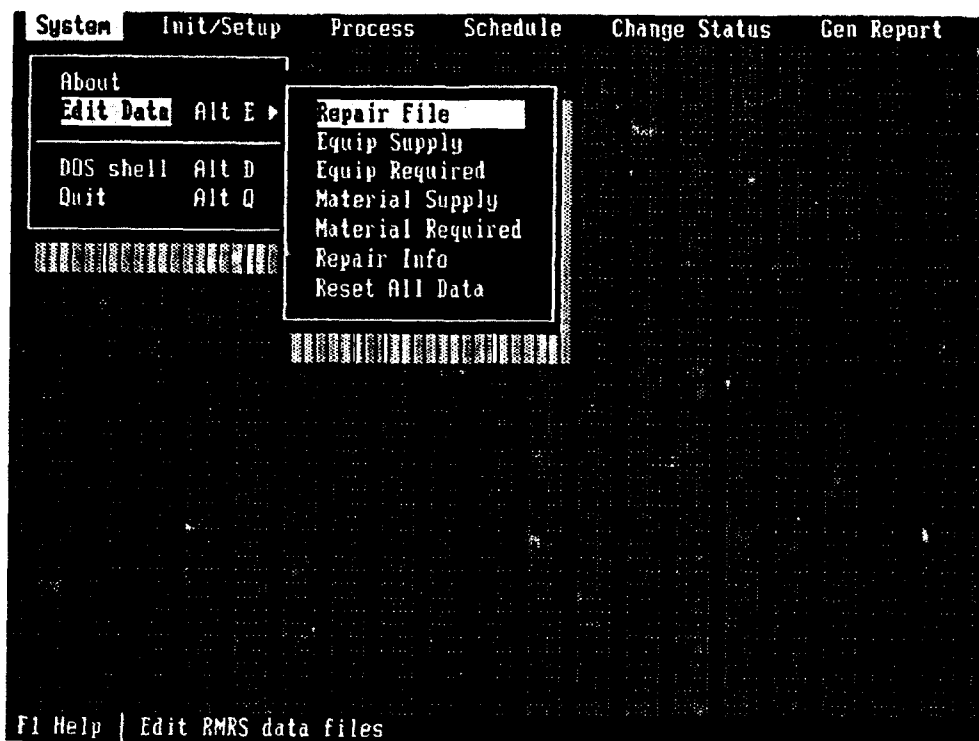


Figure 5. Edit Data Menu.

Quit (Alt-Q). The *System / Quit* command, upon verification, exits the RMRS system program, removes it from memory, and returns you to the DOS command line.

B. INIT/SETUP MENU (ALT-I)

This menu allows the user to load new supply files after program start-up.

Load new AB supply file. The *Init/Setup / Load New Supply File* command lets you convert a raw Airbase supply file to its corresponding RMRS system database. This command selects a secondary pulldown menu presenting two choices: AB materials supply file, and AB equipment supply file. Both commands open a dialog box that will accept the complete path/filename specification for the supply file to be loaded.

C. PROCESS MENU (ALT-P)

This menu allows the user to load newly arrived PDES output repair files and, subsequently, allocate material resources to a repair, barring any unresolved resource conflicts, thereby generating possible repairs to be passed to the RMRS scheduler.



Figure 6. Init/Setup Menu.

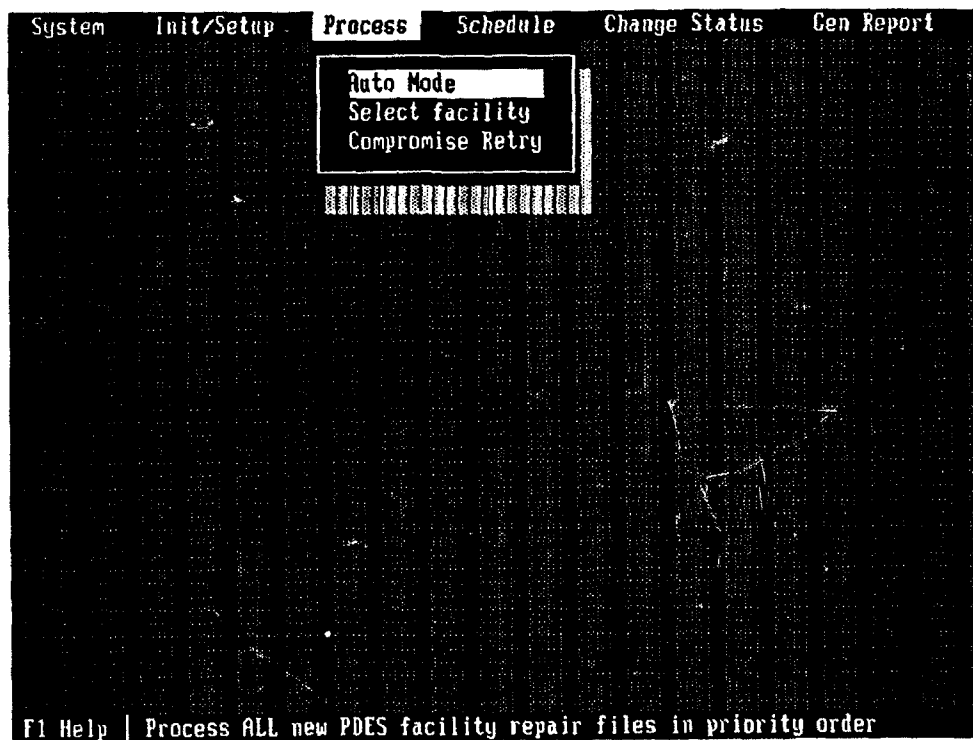


Figure 7. Process Menu.

Auto mode. The *Process / Auto Mode* command lets you (1) "process" all of the newly arrived PDES facility repair files; (2) "process" load repair data into RMRS databases from PDES .EQP, .MAT, and .OUT files; and (3) allocate material resources to those repairs if either no resource conflicts occurred or all conflicts were resolved.

"Newly arrived" PDES files are those found in the PDES Input Directory. After "processing," the PDES files (.EQP, .MAT, and .OUT files) are placed in a PDES Archive Directory. These archived copies are kept as a safety precaution because the corresponding copies in the PDES Input Directory are removed after the RMRS system 'Process' phase.

Select. The *Process / Select Facility* command lets you "process" only the facilities that you select from the newly arrived PDES facility repair files. It displays a file-selection dialog box for you to select the facility repair files that are to be "processed."

The dialog box contains a file list of .EQP files corresponding to each available PDES facility repair file set. Press the UP/DN arrows to peruse the list and press <ENTER> to mark a facility file for selection. When finished simply type Ctrl-<ENTER> to accept or <ESC> to cancel.

Compromise. The *Process / Compromise Retry* command lets you attempt to retry the resolution of the material resource conflicts associated with a (S)uspended or (O)verridden repair. It is called "retry" because when a resolvable resource conflict arises during the "process" stage, the user is alerted and given the opportunity to initiate a "Compromise" session. It first displays a dialog box with a complete list of all the repairs with (S)uspended or (O)verridden status. It then allows the user to select the repair with which to work.

Upon selection, a "Compromise" session is initiated that presents the needed information to make material resource substitution decisions. The session window contains a repair data box, a conflict count status box, a scrollable material conflict list, a scrollable material supply listing, a step-by-step instruction box, and several selection display boxes.

D. SCHEDULE MENU (ALT-S)

This menu allows the user to manipulate scheduling criteria, if desired, and to schedule the repairs that are possible.

Full. The *Schedule / Full Schedule* command lets you generate a new schedule with regard only to repair priority. Repairs that may have been previously scheduled and have not begun are rescheduled along with newly arrived repairs.

COMPROMISE Environment

Repair information			
Rapid	: 9		
Facnum	: B454		
Priority	: 1		
Elemnum	: 188		
Elem desc	: EXTERIOR WALL		
Def mode	: EXCESSIVE CRACKING		
Rep strategy	: SHOTCRETE		

181 water

3.3 gal

inadequate supply

Current status:

1 conflicts out of

2 material requirements

Selected conflict:

Selected substitution:

All material supply

INSTRUCTIONS:

Select a material conflict to resolve

F1 Help ▾/▲ Select ◀ Accept ESC Cancel All

Figure 8. Compromise Dialog Box.

System	Init/Setup	Process	Schedule	Change Status	Gen Report
--------	------------	---------	-----------------	---------------	------------

Full Schedule

Update Schedule

Set Priority

|||||

F1 Help | Schedule all repairs (i.e. new AND previously scheduled repairs)

Figure 9. Schedule Menu.

Update. The *Schedule / Update Schedule* command lets you generate an amended schedule in which previously scheduled repairs retain their time slot and new repairs are scheduled on a priority order basis.

Set. The *Schedule / Set Priority* command lets you change the scheduling of a repair. This allows manipulation of the prioritized scheduling queue order. This command displays an input screen that allows the user to select the repair of interest and specify the new priority.

E. CHANGE STATUS MENU (ALT-C)

This menu allows the user to designate a job in the repair possible list as completed or canceled.

Completed. The *Change Status / Completed* command lets you change the repair status of a (Q)ueued repair to (C)ompleted. This should be used when notification has been received that a repair has been completed.

Canceled. The *Change Status / Canceled* command lets you change the repair status of a (Q)ueued or (S)uspended repair to (X)anceled. This should be used when notification has been received that a repair has been canceled.

NOTE: The associated material resource requirements (if any) that were previously allocated, will be returned to the Airbase material supply database. Consequently, any reactivation of that repair would require that it again be "processed" and scheduled as before.

F. GEN REPORT MENU (ALT-R)

This menu allows the user to generate repair schedules and reports.

Summary. The *Gen Report / Summary Schedule* command generates an abbreviated format repair schedule which contains the "repair id" and description, repair start and completion times and overall duration for all repairs scheduled. This information is presented to the user on the screen and also output to a disk file named SUMMARY.REP.

Detailed. The *Gen Report / Detailed Schedule* command generates a schedule with same information as the summary schedule and also includes more detailed information about the repair such as equipment and material requirements, facility priority, and overall repair statistics such as number of repairs scheduled, number of repairs completed, etc. This information is presented to the user on the screen and also output to a disk file named DETAILED.REP.

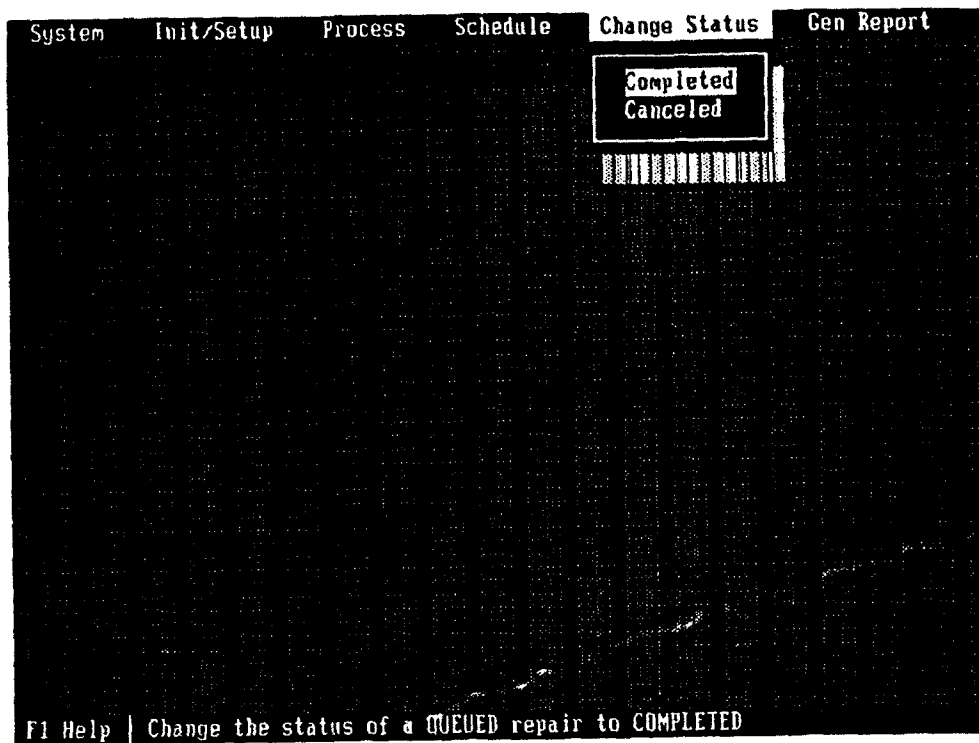


Figure 10. Change Status Menu.



Figure 11. Gen Report Menu.

Gantt. The *Gen Report / Gantt Chart* command generates a Gantt chart that allows the user to quickly visualize the temporal distribution of the repair schedule. This information is presented to the user on the screen and also output to a disk file named GANTT.REP.

SECTION IV

EXAMPLE SESSION

This section presents an example case and its corresponding RMRS operations in order to acquaint the user with the operation of the RMRS system.

Our example case involves two PDES Input facility repair file sets -- one for facility B138 and one for facility B4058 and assumes the RMRS system repair-specific databases are not empty (i.e., some repairs have been processed by the RMRS system). These file sets have just arrived from the POST-DAM Expert System (PDES). At this point, the PDES has evaluated the on-site damage reports from the Damage Assessment Teams (DATs). Its chosen repair strategy is outlined in the standard system facility repair file set, consisting of a required equipment listing (.EQP), a required material listing (.MAT), and a repair information report (.OUT). Figure 12 shows the RMRS startup as displayed when the system is started.

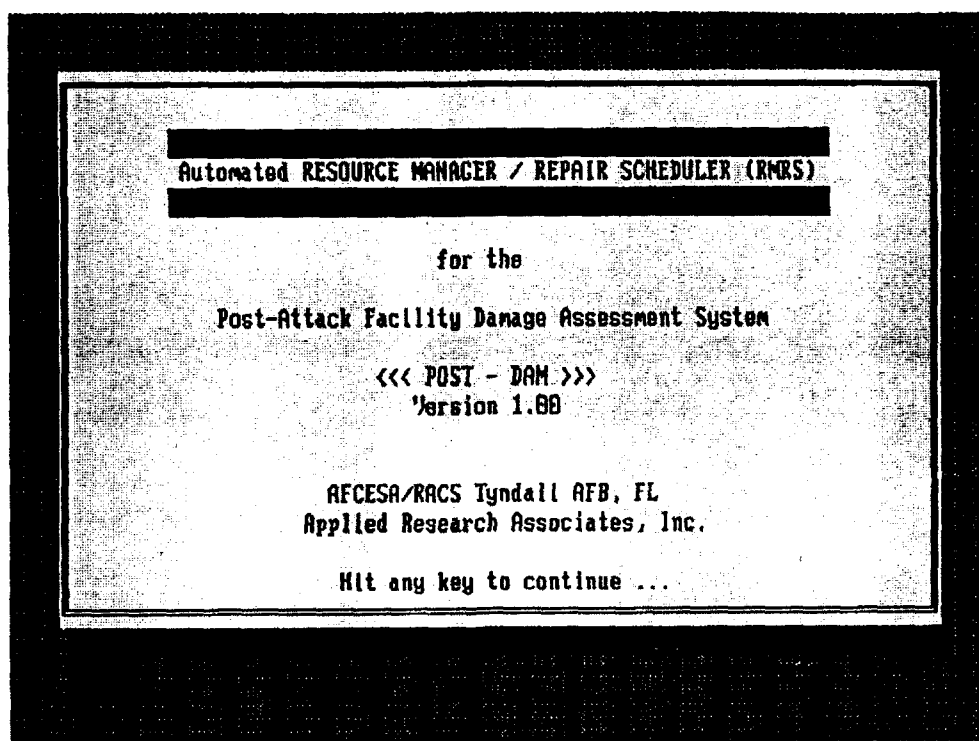


Figure 12. Startup Screen.

A. REPAIR LOADING | PROCESSING

To initiate the RMRS processing of these two facility repair file sets, select either the *Process / Auto Mode* or the *Process / Select Facility* command (i.e. the *AutoMode* or *Select Facility*

choice from the *Process* menu). In this case, we would like to process both facility file sets in sequence, so we select *Process / Auto Mode* as shown in Figure 13. On the other hand, to process only one facility, the *Process / Select Facility* command should be selected. Both of the *Process* menu commands perform three tasks:

- (1) Load new repair data
- (2) Resolve resource conflicts if possible
- (2) Allocate the appropriate material resources

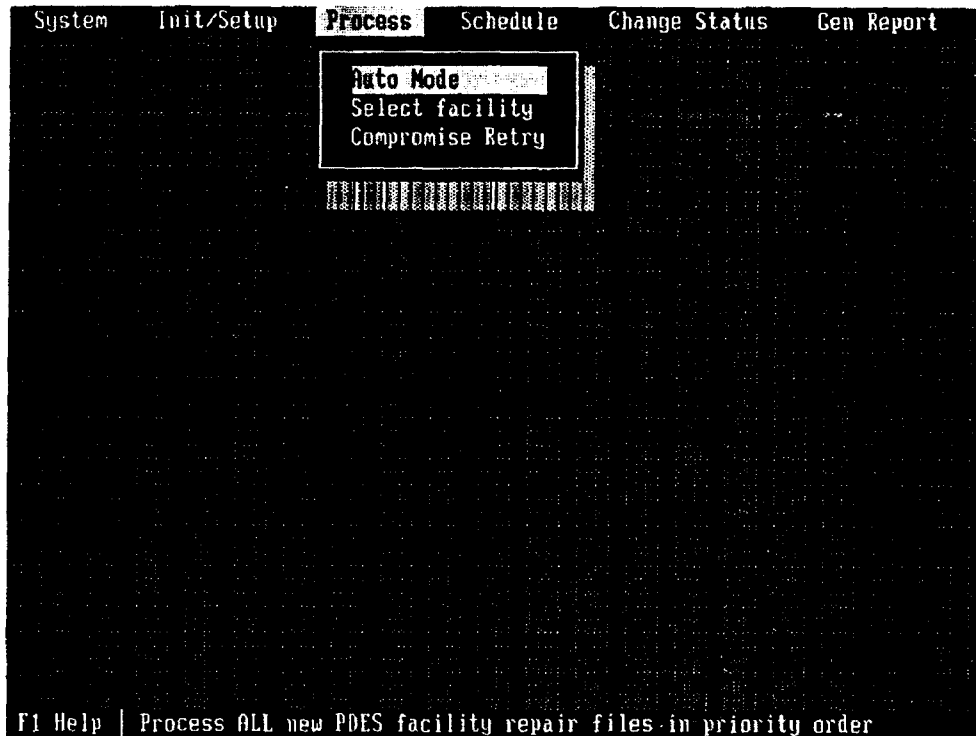


Figure 13. *Process / Auto Mode* Command.

After loading all new repairs, the *Process / Auto Mode* command sets up a prioritized repair queue from which repairs are individually processed to determine resource availability and are then allocated materials when no resource conflicts exists. In our case, facility 4058 has the higher priority and, consequently, its repairs are processed first.

The first repair for facility 4058 has been assigned a REPID of 105 by RMRS, and is evaluated for existence of resource conflicts as shown below.

The remaining repairs for facility 4058 are processed likewise. The equipment and material description strings serve as the search keys into the respective supply databases. If located, the

TABLE 1. RESOURCE AVAILABILITY ANALYSIS: REPID #105.

EQP requirements	Found ?	(✓-yes/X-no)
sc machine		✓
ramset		✓
ramset		✓
repair team		✓
MAT requirements	Found ?	Enough avail ? (✓-yes/X-no)
2x4 16ft (2 ea)	✓	✓
plywood 4x8 .5in (3 ea)	✓	✓
wire mesh (40.0 sqf)	✓	✓
shotcrete (5.9 cy)	✓	✓

"supply id" number associated with the material resource is added to the corresponding material requirement record to allow a more efficient numerical key search to be used in future searches. In the equipment requirement database, the "equipment id" number assignment is postponed until scheduling in order to address the situation of efficient time allocation for multiple supply listings of identical equipment pieces. When a resource match is not found in the appropriate supply file, that resource requirement constitutes one instance of a resource availability conflict (conflict, for short). All equipment conflicts are of this type. On the other hand, a material conflict can also arise from the situation where the material is listed in the supply, but the quantity available does not satisfy the requirement as specified.

A repair that has resource conflicts is handled in one of two ways, depending upon the types of conflicts involved. If the repair has one or more equipment conflicts, that repair is automatically (X)anceled. However, the user is free to make any manual resolution he desires. On the other hand, if the resource conflicts are limited to just materials, the user is alerted to this repair status and given the opportunity to enter the RMRS Compromise mode to make appropriate material substitutions towards a resolution. In addition, a priority override solution is also available for resolution of a material conflict, provided that the needed materials were previously allocated to a lower priority repair (that has not been started and is not (L)ocked in schedule) from which they can be stolen, if desired by the user.

For this example, no resource conflicts occurred for any of the repairs processed. The required materials for each repair were deducted from the available amounts in the material supply, the material requirement records were marked as (A)llocated, and the repair id records were marked as (P)ossible. Also, the equipment requirement records associated with these are left marked as (N)ew.

B. SCHEDULE REPAIRS

Having been Processed by the RMRS system, all (P)ossible repairs are now ready to be included in the schedule. At this point there are two scheduling options, *Full Schedule* and *Update Schedule*. The *Full Schedule* command generates a new schedule in which all previously scheduled repairs are rescheduled along with the newly arrived repairs. The *Update Schedule* command retains the current schedule and amends the newly arrived repairs. This example uses select the *Schedule / Full Schedule* command as shown in Figure 14.

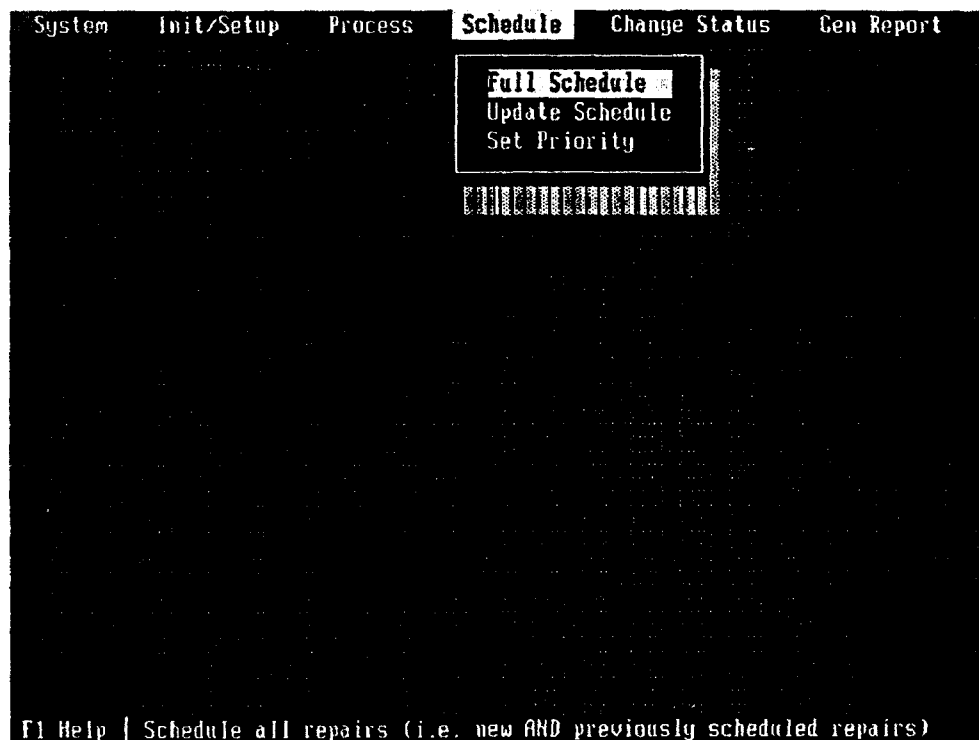


Figure 14. *Schedule / Full Schedule* Command.

The actual database manipulation involved here occurs primarily within the equipment requirements database, EQPREQ.DBF. Each repair is taken in facility priority order and scheduled at the earliest possible time based on equipment availability. Upon completion, all of the repairs will have corresponding equipment requirement records possessing a valid "equipment id" number, a START time, and a (Q)ueued status. This information constitutes the repair schedule.

C. REPORT GENERATION

The RMRS system allows for the generation of several output reports including the Summary, the Detail, and the Gantt chart formats. For this case we will choose the *Gen Report Gantt Chart* menu choice as shown in Figure 15. Once this menu choice is selected a scrollable

report is displayed on the screen similar to the one shown in Figure 16. This report contains information on the repair scheduling and equipment allocation. This report is useful for viewing a schedule and then iterating on the results. A copy of the report also is also written to a file, named GANTT.REP, which can be output on a hard copy device.



Figure 15. Gen Report / Gantt Chart Menu Choice.

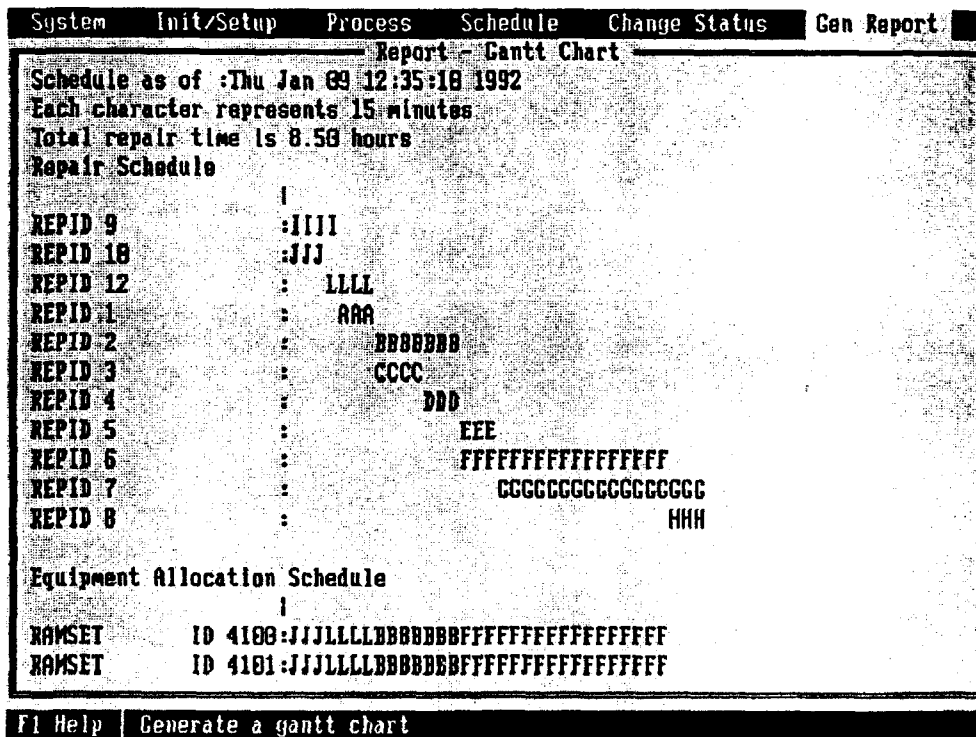


Figure 16. Gantt Chart Report.

SECTION V

CONCLUSIONS AND RECOMMENDATIONS

The RMRS prototype software clearly demonstrated an architecture that was able to quickly and efficiently manage Airbase resources and schedule expedient repairs for mission critical facilities. For repairs with no resource conflicts, the RMRS system could operate with minimal user intervention. When conflicts were present, resolution was quick and simple in the compromise environment. By allowing the user unlimited access to all of the RMRS data files, complete control of the resource allocation and scheduling processes was possible.

For full-scale development of the RMRS software, the following issues should be addressed:

1. Include a communications module to allow RMRS to receive PDES data files in the background while continuing to work uninterrupted.
2. Incorporate Expert System capabilities to provide RMRS the capability to suggest resource conflict solutions and changes in repair strategies based on current resource supplied and equipment usage.
3. Develop an optimized scheduler to generate schedules based on not only facility priority but also time and resource optimization.

REFERENCES

1. *Postattack Damage Assessment of Facilities*, Subtask 2.02, Air Force Engineering and Services Center, SETA Contract F08635-88-C-0067, December 1987.
2. *Postattack Damage Assessment of Facilities*, Subtask 2.02.1, Air Force Engineering and Services Center, SETA Contract F08635-88-C-0067, October 1988.
3. *Postattack Damage Assessment of Facilities*, Subtask 2.02.2, Air Force Engineering and Services Center, SETA Contract F08635-88-C-0067, February 1989.
4. *The POST-DAM System, Volume 1, Introduction to the POST-DAM System*. Applied Research Associates, Inc., ESL-TR-91-22, AFESC/RDCS, March 1991.
5. *The POST-DAM System, Volume 2, Software User's Manual for the Expert System*. Applied Research Associates, Inc., ESL-TR-91-22, AFESC/RDCS, February 1991.
6. *The POST-DAM System, Volume 3, Software User's Manual for the DESQ view 386*. Applied Research Associates, Inc., ESL-TR-91-22, AFESC/RDCS, December 1990.
7. *The POST-DAM System, Volume 4, Software User's Manual for the Relational Data Base Management System*, Applied Research Associates, Inc., ESL-TR-91-22, AFESC/RDCS, December 1990.
8. *The POST-DAM System, Volume 5, Software User's Manual for the Harvard Project Manager*, Applied Research Associates, Inc., ESL-TR-91-22, AFESC/RDCS, December 1990.
9. *The POST-DAM System, Volume 6, Software User's Manual for Crosstalk Mk.4 on the Host Computer*, Applied Research Associates, Inc., ESL-TR-91-22, AFESC/RDCS, December 1990.
10. *The POST-DAM System, Volume 7, Software User's Manual for the TED 1.1 Editor*. Applied Research Associates, Inc., ESL-TR-91-22, AFESC/RDCS, December 1990.
11. *The POST-DAM System, Volume 8, Software User's Manual for Crosstalk Mk.4 on the Remote Computer*, Applied Research Associates, Inc., ESL-TR-91-22, AFESC/RDCS, March 1990.

12. *The POST-DAM System, Volume 9, Fixed Manual of Mission-Critical Facilities for Use with the Prototype POST-DAM System*, Applied Research Associates, Inc., ESL-TR-91-22, AFESC/RDCS, March 1991.
13. *Code Base 4.2*, Sequiter Software, Inc., 1988-1990.
14. *Greenleaf Data Windows, Volume I, Version 2.20*, Greenleaf Software, Inc., 1983-1990.
15. *Greenleaf Data Windows, Volume II, Version 2.20*, Greenleaf Software, Inc., 1983-1990.

APPENDIX A

INSTALLATION PROCEDURE FOR RESOURCE MANAGER/REPAIR SCHEDULER (RMRS)

I. INSTALLATION

A. System Requirements

The Resource Manager/Repair Scheduler (RMRS) is designed to run on the IBM PC family of computers, including the XT and AT, along with all true IBM compatibles. The following "minimum" hardware configuration has been established:

RAM storage: 640k

Hard Disk storage: 2 mb free (disk space limits number of repairs)

Peripherals:

- (1) color or monochrome text display adaptor and monitor
- (1) 5 1/4 or 3 1/2 floppy disk drive
- (1) line printer

B. Installation Procedure

The Resource Manager/Repair Scheduler (RMRS) comes with an automatic installation program called `INSTALL`. `INSTALL` automatically copies the RMRS files to the appropriate directories on your hard disk drive. For reference, the `README` file on the installation disk includes a list of the distribution files along with recent information about RMRS.

To install RMRS:

- (1) Insert the installation disk into drive A
- (2) Type `A:INSTALL <return>`

- (3) Follow the prompts

Note: If your installation drive is drive B, substitute B for A in steps 1 and 2.

APPENDIX B

RMRS SOURCE CODE

I. RMRS SOURCE CODE

RMRS was developed using the C Language and two subroutine libraries. Borland C++ 2.0 was chosen as the development environment because of availability and favorable past experience. Although this compiler supports a full implementation of AT&T's C++ version 2.0, as well as ANSI C, only ANSI C was used to develop RMRS. Greenleaf DataWindows[®] version 2.12 was used to develop the user interface, and Code Base 4.2[®] was used for the data file management. These libraries were chosen for their maturity and because both libraries have UNIX implementations. Both the DataWindows and Code Base license agreement allow for royalty-free distribution of applications, such as RMRS, that contain object code from their libraries. However, distribution of the libraries is prohibited. Thus, any person wishing to make modifications to the RMRS program will be required to obtain both DataWindows 2.12 and Code Base 4.2. Both of these packages are readily available at minimal cost.

```
1 /*
2 *
3 *      comprom.c
4 *
5 *      module to compromise repair material conflicts
6 *
7 *      DoCompromise()
8 *
9 *      LAST REV: 12/30/91
10 *
11 *
12 */
13
14 #include <stdio.h>
15 #include <stdlib.h>
16 #include <string.h>
17
18 /* toolkit header files */
19 #include "dw.h"
20 #include "pwrbase.h"
21 #include "color.h"
22 #include "lkumod.h"
23 #include "d4base.h"
24 #include "u4error.h"
25
26 /* custom header files */
27 #include "rmrsutil.h"
28 #include "sysdefs.h"
29 #include "procutil.h"
30 #include "miscutil.h"
31 #include "dbutil.h"
32
33 /* items that can be highlighted */
34 #define CONFSTAT 1
35 #define SELTCONF 2
36 #define SELTSUB 3
37 #define CURSTAT 4
38 #define INSTRCT 5
39
40 /* ctrl items that can be selected */
41 #define CONFSEL 1
42 #define SUPPSEL 2
43 #define SELTQTY 3
44 #define NOSELCT 4
45
46 /* key defs for lku return value */
47 #define CANCEL_SEL -1
48 #define ABORTSUB_SEL -2
49
50 /* DoCompromise() support fns *****/
51
52 HWND initcompwin(void);
53 HWND initkeymapbar(void);
54 void delcompwin(void);
55 void delkeymapbar(void);
56 void setupwinformat(void);
57 void dsplyrepinfo(REPDATANODE *);
58 void dsplycompstatus(int);
59 void dsplyconfstat(LBOXSTA *);
60 void dsplycompsuccessbox(void);
61 void updtcompstatus(LISTITEM *);
62 char * getnextmatconf(REQLSTNODE **);
63 char * getfirstmatconf(REQLSTNODE *, REQLSTNODE **);
64 char * getsupprecstr(long);
65 int isoverrideposs(LBOXSTA *);
66 int asktoacceptoverride(void);
67 void hilite(int);
68 void unhilite(int);
69 void ctrlpas(int, int);
70 float getqty(void);
71 int verifyabort(void);
72 int chkqtyvssupp(float, long);
73 int askifcorrectqty(void);
74 void alerttoqtylimit(long);
```

```
75 void dsplysubaccepted(void);
76 void recordsub(LBOXSTA *, float, long);
77 void setovrdflag(LBOXSTA *);
78
79 /*-----*/
80
81 static HWND KeymapBar;
82 static HWND CompWin;
83
84
85
```

```

int DoCompromise(long repid, REQLSTNODE * reqlstptr)
85 (
86 /*
87  * ARGUMENT
88  * (long)      repid      - repair id
89  * (REQLSTNODE *) reqlstptr - pointer to resource availability analysis list
90  *
91  * DESCRIPTION
92  * Allows the user to interactively attempt a conflict resolution strategy
93  * through material substitutions and priority overrides on a repair
94  *
95  * RETURNS
96  * (int) SUCCESS or FAILURE
97  *
98  * DATABASES AFFECTED (whether direct or indirect)
99  *   matsup_dbf (matsup_ndx2 - MATID index      ) - (CS$) to be added
100  *
101  */
102
103  LISTITEM * conflboxlst;
104  LBOXSTA * conflbox;
105  REQLSTNODE * currptra;
106  char * strptr, * choice;
107  REPDATANODE * repinfoblk;
108  long selrecno;
109  char * suppressstr;
110  bool success = FALSE, acceptovrd = FALSE, validqty, acceptqty;
111  /* these will be running totals */
112  int numofmatreqs = 5;
113  unsigned int key;
114  float subqty;
115
116
117  if (isreqpconf(reqlstptr))
118  {
119      u4error(INVCOMPCALL, "Invalid call to COMPROMISE", "function: ",
120             "NO CONFLICTS DETECTED !", (char *) 0);
121      return (-1);
122  }
123  KeymapBar = initkeymapbar();
124  CompWin = initcompwin();
125  repinfoblk = GetRepData(repid);
126  dsplyrepinfo(repinfoblk);
127  dsplycompstatus(numofmatreqs);
128  setupwinformat();
129
130
131  /* load list box linked list with conflicting mat record strings */
132  conflboxlst = initlist();
133  strptr = getfirstmatconf(reqlstptr, & currptra);
134  while (strptr != NULL)
135  {
136      adtolist(conflboxlst, strptr);
137      strptr = getnextmatconf(& currptra);
138  }
139  updtcompstatus(conflboxlst);
140
141  /* CONFLICT PROCESSING LOOP */
142  do
143  {
144      ctrlpass(NOSELECT, CONFSEL);
145
146      hilite(CONFSTAT);
147      hilite(CURSTAT);
148      hilite(INSTRCT);
149
150      /* select material conflict from conflict list */
151      conflbox = lboxinit(4, 37, 4, REVHIGHNORML, HELP, "", REVNORML, FRSINGLE,
152                        REVNORML, conflboxlst);
153      dsplyconfstat(conflbox);
154      CONFSELBOX :          // conflict selection box
155      while ((key = getkey()) != ESC && key != ENTR && key != F3)
156      {

```

```

154         lboxctrl(conflbox, key);
155         dsplyconfstat(conflbox);
156     }
157     if (key == ESC)
158     {
159         if (verifyabort())
160             goto CANCEL;
161         goto ABORTSUB;
162     }
163     else if (key == F3) goto CONFSELBOX;
164     choice = lboxsel(conflbox);
165     unhlite(CONFSTAT);
166     hlite(SELTCNF);
167     vratputf(CompWin, 11, 36, HELP, " %-38.38s ", choice);/* display conflict selection*/
168     ctrlpas{CONFSEL, NOSELCT};
169
170     if (isoverridepos{conflbox})
171         acceptovrd = asktoacceptoverride();
172     else
173         acceptovrd = FALSE;
174     if (! acceptovrd)
175     {
176         ctrlpas{NOSELCT, SUPPSEL};
177
178         /* select substitution material from AB supply */
179         LKUmodinit();
180         LKUmodposition(17, 3, 4);
181         LKUmodattributes(0x10 | 0x03, 0x20 | 0x00);
182         LKUmodselect(matsup_dbf, -1);
183         if ((selrecno = LKUmodlookup()) < 0)
184         {
185             ctrlpas{SUPPSEL, NOSELCT};
186             switch ((int) selrecno)
187             {
188                 case (CANCEL_SEL) : if (verifyabort())
189                                     goto CANCEL;
190                 case (ABORTSUB_SEL) : goto ABORTSUB;
191                 default ;;
192             }
193         }
194         else
195         {
196             /* getsupprecstr and chkqtyvssupp require MATSUP.DBF */
197             supprecstr = getsupprecstr(selrecno);
198             hlite(SELTSUB);
199             vratputf(CompWin, 13, 36, HELP, " %-38.38s ", supprecstr);/* display supply sel
200             ection*/
201             ctrlpas{SUPPSEL, SELTQTY};
202
203             acceptqty = FALSE;
204             /* get qty, validate, verify substitution */
205             do
206             {
207                 acceptqty = FALSE;
208                 if ((subqty = getqty()) < 0)
209                 {
210                     switch ((int) subqty)
211                     {
212                         case (CANCEL_SEL) : if (verifyabort())
213                                             {
214                                                 ctrlpas{SELTQTY, NOSELCT};
215                                                 goto CANCEL;
216                                             }
217                         break;
218                     }
219
220                     case (ABORTSUB_SEL) : ctrlpas{SELTQTY, NOSELCT};
221                                         goto ABORTSUB;
222                     default ;;
223                 }
224             }
225             else
226             {

```

```

217         validqty = chkqtyvssupp(subqty, selrecno);
218         if (validqty)
219             acceptqty = askifcorrectqty();
220         else
221             alerttoqtylimit(selrecno);
222     }
223     while (! validqty || ! acceptqty);
224
225     d4select(matsup_dbf);
226     ctrlpass(SELTQTY, NOSELECT);
227     dsplysubaccepted();
228 }
229 /* record substitution */
230 recordsub(conflbox, subqty, selrecno);
231 }
232 else
233     setovrdflag(conflbox);
234 rmfm1st(conflbox1st, choice);
235
236 ABORTSUB :
237 unhilite(SELTCONF);
238 unhilite(SELTSUB);
239 vratputf(CompWin, 11, 36, HELP, " %-38.38s ", " "); /* clr conf selection display */
240 vratputf(CompWin, 13, 36, HELP, " %-38.38s ", " "); /* clr sub selection display */
241 lboxdel(conflbox);
242 if (! (listcnt(conflbox1st)))
243     success = TRUE;
244 updtcompstatus(conflbox1st);
245
246 }
247 while (! success);
248
249 if (success)
250 {
251     unhilite(INSTRCT);
252     dsplycompsuccessbox();
253 }
254
255 CANCEL : // cancel compromise
256 lboxdel(conflbox);
257 freelist(conflbox1st, 0);
258 free((REPDATANODE *) repinfoblk);
259 delcompwin();
260 delkeymapbar();
261
262 if (success)
263     return (SUCCESS);
264 return (FAILURE);
265 }
266
267 /* End of main ===== */
268
269 /* functions not called directly from DoCompromise() */
270 /*-----*/
271 char getavailstatofsel(LBOXSTA *);
272 char * bld1ststr(REQLSTNODE *);
273 char * getconfstat(LBOXSTA *);
274 HWND dsplymessbox(void);
275 void delmessbox(HWND);
276 void delay(int);
277 /*-----*/
278
279 HWND initkeymapbar(void)
280 {
281     /*
282     * ARGUMENT
283     * DESCRIPTION
284     * Initializes key map bar for COMPROMISE environment
285     * RETURNS
286     * (HWND) key map bar window handle

```

```

287 *
288 */
289
290     HWND keymapbar;
291
292 /* composed attributes */
293     #define KEYS REVERR
294
295     keymapbar = vcreat(3, 80, EMPHNORML, YES);
296     vwind(keymapbar, 1, 80, 0, 0);
297     vlocate(keymapbar, 24, 0);
298     visible(keymapbar, YES, YES);
299     vatps(0, 0, " F1 Help / Select DY Accept ESC Cancel All");
300     modattr(keymapbar, 0, 1, 2, KEYS);
301     modattr(keymapbar, 0, 10, 1, KEYS);
302     modattr(keymapbar, 0, 12, 1, KEYS);
303     modattr(keymapbar, 0, 22, 3, KEYS);
304     modattr(keymapbar, 0, 34, 3, KEYS);
305     vatps(1, 0, " F1 Help / Select DY Accept F2 Seek Matid F3 Abort Sub ESC Cancel All"
);
306     modattr(keymapbar, 1, 1, 2, KEYS);
307     modattr(keymapbar, 1, 10, 1, KEYS);
308     modattr(keymapbar, 1, 12, 1, KEYS);
309     modattr(keymapbar, 1, 22, 3, KEYS);
310     modattr(keymapbar, 1, 34, 2, KEYS);
311     modattr(keymapbar, 1, 49, 2, KEYS);
312     modattr(keymapbar, 1, 63, 3, KEYS);
313     vatps(2, 0, " F1 Help <0..9,'.'> Enter Qty BKSP Edit DY Accept F3 Abort Sub");
314     modattr(keymapbar, 2, 1, 2, KEYS);
315     modattr(keymapbar, 2, 11, 1, KEYS);
316     modattr(keymapbar, 2, 14, 1, KEYS);
317     modattr(keymapbar, 2, 17, 1, KEYS);
318     modattr(keymapbar, 2, 32, 4, KEYS);
319     modattr(keymapbar, 2, 43, 3, KEYS);
320     modattr(keymapbar, 2, 55, 2, KEYS);
321     visible(keymapbar, YES, NO);
322     return (keymapbar);
323 }
324
325 HWND initcompwin(void)
326 {
327 /*
328 *
329 * DESCRIPTION
330 *     Initializes COMPROMISE environment window
331 *
332 * RETURNS
333 *     (HWND) COMPROMISE environment window handle
334 *
335 */
336     HWND compwin;
337
338
339     compwin = vcreat(22, 78, WHITE_ON_BLUE, YES);
340     vlocate(compwin, 1, 1);
341     vframe(compwin, WHITE_ON_BLUE, FRSINGLE);
342     vmttitle(compwin, TOP, CENTERJUST, WHITE_ON_BLUE, " COMPROMISE Environment ");
343     visible(compwin, YES, YES);
344     return (compwin);
345 }
346
347 void delcompwin(void)
348 {
349 /*
350 * ARGUMENT
351 *
352 * DESCRIPTION
353 *     Deletes COMPROMISE environment window
354 *
355 * RETURNS
356 *
357 */

```

```
358     vdelete(CompWin, NONE);
359 }
360
361 void delkeymapbar(void)
362 {
363     /*
364     * ARGUMENT
365     *
366     * DESCRIPTION
367     *   Deletes key map bar window
368     *
369     * RETURNS
370     */
371     vdelete(KeymapBar, NONE);
372 }
373
374 void dsplycompsuccessbox(void)
375 {
376     /*
377     * ARGUMENT
378     *
379     * DESCRIPTION
380     *   Displays COMPROMISE "Success" message box
381     *
382     * RETURNS
383     */
384     HWND compsuccwin;
385
386     compsuccwin = dsplymessbox();
387     vratps(1, 0, GRYBTXT, "      **** SUCCESS ****      ");
388     vratps(3, 0, GRYBTXT, "   All conflicting material   ");
389     vratps(4, 0, GRYBTXT, "   requirements resolved!   ");
390     vratputf(compsuccwin, 7, 0, KEYALERT, "      < Press any key >      ");
391     getkey();
392     delmessbox(compsuccwin);
393     return;
394 }
395
396
397 HWND dsplymessbox(void)
398 {
399     /*
400     * ARGUMENT
401     *
402     * DESCRIPTION
403     *   Displays a standard message box
404     *
405     * RETURNS
406     *   (HWND) message box window handle
407     */
408     HWND messwin;
409
410     messwin = vcreat(8, 30, GREYBOX, YES);
411     vlocate(messwin, 8, 24);
412     vframe(messwin, GREYBOX, FRDOUBLE);
413     vshadow(messwin, CURRENT, SHADOW100, BOTTOMRIGHT);
414     visible(messwin, YES, YES);
415     return (messwin);
416 }
417
418
419 void delmessbox(HWND messwin)
420 {
421     /*
422     * ARGUMENT
423     *   (HWND) messwin - message box window handle
424     *
425     * DESCRIPTION
426     *   Deletes message box
427     *
428     * RETURNS
```

```

428 *
429 */
430
431     vdelete(messwin, NONE);
432     vcurrent(CompWin);
433     return;
434 }
435
436 void dsplyreinfo(REPDATANODE * blkptr)
437 {
438     /*
439     * ARGUMENT
440     * (REPDATANODE *) blkptr - pointer to repair data block
441     *
442     * DESCRIPTION
443     * Displays the repair data block on the COMPROMISE environment window
444     *
445     * RETURNS
446     */
447
448     if (blkptr == (REPDATANODE *) NULL)
449     {
450         u4error(ERROR, "error", (char *) 0);
451         return;
452     }
453     vratputf(CompWin, 1, 2, WHITE_ON_BLUE, "Repair information");
454     vratputf(CompWin, 2, 2, ERR, " Repid      : %-15d", blkptr->repid);
455     vratputf(CompWin, 3, 2, ERR, " Facnum    : B%-14d", blkptr->facnum);
456     vratputf(CompWin, 4, 2, ERR, " Priority   : %-15d", blkptr->priority);
457     vratputf(CompWin, 5, 2, ERR, " Elenum    : %-15d", blkptr->elenum);
458     vratputf(CompWin, 6, 2, ERR, " Elem desc  : %-15s", blkptr->eledescstr);
459     vratputf(CompWin, 7, 2, ERR, " Dam mode   : %-15s", blkptr->dammodestr);
460     vratputf(CompWin, 8, 2, ERR, " Rep stratgy : %-15s", blkptr->repstgstr);
461     return;
462 }
463
464 void dsplycompstatus(int numofmatreqs)
465 {
466     /*
467     * ARGUMENT
468     * (int) numofmatreqs - number of material requirements for a repair
469     *
470     * DESCRIPTION
471     * Displays COMPROMISE status block of repair (i.e. block specifies
472     * how many material conflicts exist out of total material requirements)
473     *
474     * RETURNS
475     */
476
477     vratputf(CompWin, 10, 2, WHITE_ON_BLUE, "Current status");
478     vratputf(CompWin, 11, 2, REVHIGHHELP, " %2.2s ", " ");
479     vratputf(CompWin, 11, 7, ERR, " conflicts out of ");
480     vratputf(CompWin, 12, 2, REVHIGHHELP, " %2d ", numofmatreqs);
481     vratputf(CompWin, 12, 7, ERR, " material requirements ");
482     return;
483 }
484
485
486 char * bldlststr(REQLSTNODE * ptr)
487 {
488     /*
489     * ARGUMENT
490     * (REQLSTNODE *) ptr - pointer to resource availability analysis list
491     *                        node corresponding to material
492     *                        requirement
493     *
494     * DESCRIPTION
495     * Builds a string representation of material record,
496     * string storage is within the same REQLSTNODE node
497     *
498     * RETURNS

```

```

498 *   (char *) Pointer to that string within the REQLSTNODE node to be
499 *   given to DATAWINDOWS listbox function
500 *
501 * DATABASES AFFECTED (whether direct or indirect)
502 *   matreq_dbf (no index used)
503 *
504 */
505
506   extern matreq_dbf;
507   char matdesc[60], unit[10];
508
509   d4select(matreq_dbf);
510   d4go(ptr->recno);
511   strcpy(matdesc, f4str(f4ref("MATDESC")));
512   strcpy(unit, f4str(f4ref("UNIT")));
513   strcpy(ptr->recstr, "");
514   sprintf(ptr->recstr, "%5ld %20.20s %7.1f %3.3s ",
515           f4long(f4ref("MATID")),
516           matdesc,
517           f4double(f4ref("QTY")),
518           unit);
519   return (ptr->recstr);
520 }
521
522 char * getfirstmatconf(REQLSTNODE * hdptr, REQLSTNODE **ptrptr)
523 {
524 /*
525  * ARGUMENT
526  *   (REQLSTNODE *) hdptr - pointer to headnode of resource availability
527  *                           analysis list
528  *   (REQLSTNODE *) ptrptr - pointer to a pointer to the current material
529  *                           conflict node (allowing pointer modification)
530  *
531  * DESCRIPTION
532  *   Increments node pointer to first material conflict node in the
533  *   resource availability analysis list at which time a material
534  *   record node-resident text string is built for that conflict
535  *
536  * RETURNS
537  *   (char *) Pointer to that string within the REQLSTNODE node to be
538  *   given to DATAWINDOWS listbox function
539  *
540  * DATABASES AFFECTED (whether direct or indirect)
541  *   matreq_dbf (no index used)
542  */
543
544   char * retstrptr;
545
546   (* ptrptr) = (REQLSTNODE *) NULL;
547   if (hdptr == (REQLSTNODE *) NULL)
548   {
549       u4error(EMPTYLST, "Empty list error", (char *) 0);
550       return (NULL);
551   }
552   (* ptrptr) = (hdptr->next)->next; /* skip past first eqp rec */
553
554   /* find first material record */
555   for (; (* ptrptr)->type != 'M'; (* ptrptr) = (* ptrptr)->next)
556   {
557       if ((* ptrptr) == (REQLSTNODE *) NULL)
558       {
559           u4error(NONULLEXPT, "No null ptr expected", (char *) 0);
560           return (NULL);
561       }
562   }
563
564   /* find first material conflict */
565   for (; (* ptrptr)->availstatus != 'E'; (* ptrptr) = (* ptrptr)->next)
566   {
567       if ((* ptrptr) == (REQLSTNODE *) NULL)
568       {
569           u4error(NONULLEXPT, "No null ptr expected", (char *) 0);
570           return (NULL);
571       }
572   }

```

```

567     retstrptr = bldlststr(* ptrptr);
568     (* ptrptr) = (* ptrptr)->next;
569
570     return (retstrptr);
571 }
572
573
574 char * getnextmatconf(REQLSTNODE **ptrptr)
575 {
576     /*
577     * ARGUMENT
578     * (REQLSTNODE *) ptrptr - pointer to a pointer to the current material
579     *                           conflict node (allowing pointer modification)
580     *
581     * DESCRIPTION
582     * Increments node pointer to next material conflict node in the
583     * resource availability analysis list at which time a material
584     * record node-resident text string is built for that conflict
585     *
586     * RETURNS
587     * (char *) Pointer to that string within the REQLSTNODE node to be
588     * given to DATAWINDOWS listbox function
589     * DATABASES AFFECTED (whether direct or indirect)
590     * matreq_dbf (no index used)
591     */
592
593     char * retstrptr;
594
595     if ((* ptrptr) == (REQLSTNODE *) NULL)
596     {
597         return (NULL);
598     }
599     /* find next material conflict */
600     for (; (* ptrptr)->availstatus == 'E'; (* ptrptr) = (* ptrptr)->next)
601     {
602         if (((* ptrptr)->next) == (REQLSTNODE *) NULL)
603         {
604             return (NULL);
605         }
606     }
607     retstrptr = bldlststr(* ptrptr);
608     (* ptrptr) = (* ptrptr)->next;
609     return (retstrptr);
610 }
611
612 char * getsupprecstr(long recno)
613 {
614     /*
615     * ARGUMENT
616     * (long) recno - material supply record number (returned by LKU fn)
617     *
618     * DESCRIPTION
619     * Takes supply record number and gets the specified record in string form
620     *
621     * RETURNS
622     * (char *) string representation of material supply record
623     *
624     * DATABASES AFFECTED (whether direct or indirect)
625     * matsup_dbf (no index used)
626     */
627
628     static char str[80];
629     MATSUPREC * msuprec;
630
631     msuprec = GetSuppRec(recno);
632     strcpy(str, "");
633     sprintf(str, "%5.5ld %20.20s %7.7s %3.3s ",
634             msuprec->matid,
635             msuprec->matdesc,

```

```

636     " ",
637     msuprec->unit);
638     free((MATSUPREC *) msuprec);
639     return (str);
640 }
641
642 char getavailstatofsel(LBOXSTA * lboxptr)
643 {
644     /*
645     * ARGUMENT
646     * (LBOXSTA *) lboxptr - pointer to list box structure of conflict list
647     *
648     * DESCRIPTION
649     * Uses pointer in list box structure to access the availability status
650     * code resident in the corresponding material resource requirement node
651     * in the resource availability analysis list
652     *
653     * RETURNS
654     * (char) availability status code (e.g. (M)issing, (I)nadequate, (O)verride)
655     */
656     return (* ((lboxptr->selectitem->listitem) + LBOXSTRLEN));
657 }
658
659 void setovrdflag(LBOXSTA * lboxptr)
660 {
661     /*
662     * ARGUMENT
663     * (LBOXSTA *) lboxptr - pointer to list box structure of conflict list
664     *
665     * DESCRIPTION
666     * Uses pointer in list box structure to set override flag
667     * resident in the corresponding material resource requirement node
668     * in the resource availability analysis list
669     *
670     * RETURNS
671     */
672     /*
673     *
674     */
675     ((REQLISTNODE *) ((lboxptr->selectitem->listitem) - sizeof (char) - sizeof (long)))->ovrd =
TRUE;
676 }
677
678 void recordsub(LBOXSTA * lboxptr, float qty, long recno)
679 {
680     /*
681     * ARGUMENT
682     * (LBOXSTA *) lboxptr - pointer to list box structure of conflict list
683     * (float) qty - substitution qty
684     * (long) recno - substitution record number
685     *
686     * DESCRIPTION
687     * Uses pointer in list box structure to record the material substitution
688     * using the appropriate fields (i.e. resrecno, resqty)
689     * resident in the corresponding material resource requirement node
690     * in the resource availability analysis list
691     *
692     * RETURNS
693     */
694     /*
695     *
696     */
697     ((REQLISTNODE *) ((lboxptr->selectitem->listitem) - sizeof (char) - sizeof (long)))->resrecno =
recno;
698     ((REQLISTNODE *) ((lboxptr->selectitem->listitem) - sizeof (char) - sizeof (long)))->resqty =
qty;
699     return;
700 }
701
702 char * getconfstat(LBOXSTA * lboxptr)
703 {
704     /*
705     * ARGUMENT

```

```
703 * (LBOXSTA *) lboxptr - pointer to list box structure of conflict list
704 *
705 * DESCRIPTION
706 *   Retrieves and decodes the conflict status of a material requirement
707 *
708 * RETURNS
709 *   (char *) conflict status explanation string
710 *
711 */
712
713 static char confstat[30];
714
715 switch (getavailstatofsel(lboxptr))
716 {
717 case ('I') : strcpy(confstat, "inadequate supply");
718               break;
719 case ('O') : strcpy(confstat, "priority override possible");
720               break;
721 case ('M') : strcpy(confstat, "mat not found in AB supply");
722               break;
723 default : strcpy(confstat, "invalid status");
724 }
725 return (confstat);
726 }
727 void dsplyconfstat(LBOXSTA * lboxptr)
728 {
729 /*
730 * ARGUMENT
731 *   (LBOXSTA *) lboxptr - pointer to list box structure of conflict list
732 *
733 * DESCRIPTION
734 *   Displays the conflict status explanation string in the COMPROMISE
735 *   environment window
736 *
737 * RETURNS
738 */
739
740   vratputf(CompWin, 8, 46, HELP, " %-26.26s ", getconfstat(lboxptr));
741 }
742
743 int isoverrideposs(LBOXSTA * lboxptr)
744 {
745 /*
746 * ARGUMENT
747 *   (LBOXSTA *) lboxptr - pointer to list box structure of conflict list
748 *
749 * DESCRIPTION
750 *   Checks to see if a priority override has been designated for the
751 *   selected material conflict
752 *
753 * RETURNS
754 *   (int) TRUE or FALSE
755 *
756 */
757
758   if (getavailstatofsel(lboxptr) == 'O')
759       return (TRUE);
760   else
761       return (FALSE);
762 }
763
764 void setupwinformat(void)
765 {
766 /*
767 * ARGUMENT
768 *
769 * DESCRIPTION
770 *   Sets up the COMPROMISE environment window format
771 *
772 * RETURNS
```

```

773 *
774 */
775
776     int i;
777
778     vatps(1, 35, "    Conflicting mat requirements ");
779     vatps(15, 2, "    AB material supply ");
780     vatps(7, 37, "CONFLICT ");
781     vatps(8, 37, "STATUS : ");
782     vratputf(CompWin, 8, 47, HELP, " %-26.26s ", " ");
783     /* setup selection display section */
784     vatps(10, 36, "Selected conflict");
785     vratputf(CompWin, 11, 36, HELP, " %-38.38s ", " ");
786     vatps(12, 36, "Selected substitution");
787     vratputf(CompWin, 13, 36, HELP, " %-38.38s ", " ");
788     vatps(15, 60, "INSTRUCTIONS: ");
789     for (i = 0; i <= 5; i++)
790         vatpas(16 + i, 60, HELP, "                ");
791     return;
792 }
793
794
795
796 void hilite(int item)
797 {
798     /*
799     * ARGUMENT
800     * (int) item - mnemonic constant corresponding to a COMPROMISE
801     * environment window item
802     *
803     * DESCRIPTION
804     * Hilites the specified item on the screen
805     *
806     * RETURNS
807     */
808     switch (item)
809     {
810     case (CONFSTAT) :
811         /* hilite "CONFLICT" , hilite "STATUS : " */
812         modattr(CompWin, 7, 37, 8, HILITE);
813         modattr(CompWin, 8, 37, 8, HILITE);
814         break;
815     case (SELTCONF) :
816         /* hilite "Selected conflict" */
817         modattr(CompWin, 10, 35, 18, HILITE);
818         break;
819     case (SELTSUB) :
820         /* hilite "Selected substitution" */
821         modattr(CompWin, 12, 36, 21, HILITE);
822         break;
823     case (CURSTAT) :
824         /* hilite "Current status" */
825         modattr(CompWin, 10, 2, 14, HILITE);
826         break;
827     case (INSTRCT) :
828         /* hilite "Instruction:" */
829         modattr(CompWin, 15, 60, 14, HILITE);
830         break;
831     default :;
832     }
833     return;
834 }
835
836
837 void unhilite(int item)
838 {
839     /*
840     * ARGUMENT
841     * (int) item - mnemonic constant corresponding to a COMPROMISE
842     * environment window item
843     *
844     * DESCRIPTION

```

```

844 *   Unhilites the specified item on the screen
845 *
846 * RETURNS
847 *
848 */
849   switch (item)
850   {
851     case (CONFSTAT) :
852       /* unhilite "CONFLICT" , hilite "STATUS : " */
853       modattr(CompWin, 7, 37, 8, CURRENT);
854       modattr(CompWin, 8, 37, 8, CURRENT);
855       break;
856     case (SELTCNF) :
857       /* unhilite "Selected conflict" */
858       modattr(CompWin, 10, 35, 18, CURRENT);
859       break;
860     case (SELTSUB) :
861       /* unhilite "Selected substitution" */
862       modattr(CompWin, 12, 36, 21, CURRENT);
863       break;
864     case (CURSTAT) :
865       /* unhilite "Current status" */
866       modattr(CompWin, 10, 2, 14, CURRENT);
867       break;
868     case (INSTRCT) :
869       /* unhilite "Instruction:" */
870       modattr(CompWin, 15, 60, 14, CURRENT);
871       break;
872     default ;;
873   }
874   return;
875 }
876
877 void ctrlpas(int from, int to)
878 {
879   /*
880   * ARGUMENT
881   *   (int) from - current control point
882   *   (int) to   - next control point
883   *
884   * DESCRIPTION
885   *   Coordinates screen control of the COMPROMISE environment between
886   *   the different control points
887   *
888   * RETURNS
889   *
890   */
891   switch (from)
892   {
893     case (CONFSEL) :
894       /* unselect "Conflicting mat requirements" */
895       vatputs(CompWin, 1, 35, " ");
896       modattr(CompWin, 1, 35, 32, CURRENT);
897       break;
898     case (SUPPSEL) :
899       /* unselect "AB material supply" */
900       vatputs(CompWin, 15, 2, " ");
901       modattr(CompWin, 15, 4, 20, CURRENT);
902       break;
903     case (SELTQTY) :
904       /* unselect "Selected substitution" */
905       vatputs(CompWin, 13, 33, " ");
906       modattr(CompWin, 12, 36, 21, CURRENT);
907       break;
908     case (NOSELCT) :
909       default ;;
910   }
911   switch (to)
912   {
913     case (CONFSEL) :

```

```

914      /* select "Conflicting material requirements" */
915      vratputf(CompWin, 1, 35, SELBLINK, " ");
916      modattr(CompWin, 1, 37, 30, SELNORM);
917      vloc(KeymapBar, 0, 0);
918      vratputs(CompWin, 16, 60, HELP, " ");
919      vratputs(CompWin, 17, 60, HELP, " > Select a ");
920      vratputs(CompWin, 18, 60, HELP, " material ");
921      vratputs(CompWin, 19, 60, HELP, " conflict to ");
922      vratputs(CompWin, 20, 60, HELP, " resolve ");
923      vratputs(CompWin, 21, 60, HELP, " ");
924      break;
925      case (SUPPSEL) :
926      /* select "AB material supply" */
927      vratputf(CompWin, 15, 2, SELBLINK, " ");
928      modattr(CompWin, 15, 4, 20, SELNORM);
929      vloc(KeymapBar, 1, 0);
930      vratputs(CompWin, 16, 60, HELP, " ");
931      vratputs(CompWin, 17, 60, HELP, " > Select a ");
932      vratputs(CompWin, 18, 60, HELP, " substitute ");
933      vratputs(CompWin, 19, 60, HELP, " material ");
934      vratputs(CompWin, 20, 60, HELP, " ");
935      vratputs(CompWin, 21, 60, HELP, " ");
936      break;
937      case (SELTQTY) :
938      /* unselect "Selected substitution" */
939      vratputf(CompWin, 13, 33, SELBLINK, " ");
940      vloc(KeymapBar, 2, 0);
941      vratputs(CompWin, 16, 60, HELP, " ");
942      vratputs(CompWin, 17, 60, HELP, " > Select qty ");
943      vratputs(CompWin, 18, 60, HELP, " to ");
944      vratputs(CompWin, 19, 60, HELP, " substitute ");
945      vratputs(CompWin, 20, 60, HELP, " ");
946      vratputs(CompWin, 21, 60, HELP, " ");
947      break;
948      case (NOSELECT) :
949      vratputs(CompWin, 16, 60, HELP, " ");
950      vratputs(CompWin, 17, 60, HELP, " ");
951      vratputs(CompWin, 18, 60, HELP, " ");
952      vratputs(CompWin, 19, 60, HELP, " ");
953      vratputs(CompWin, 20, 60, HELP, " ");
954      vratputs(CompWin, 21, 60, HELP, " ");
955      default :;
956      }
957      return;
958 }
959
960 void updtcompstatus(LISTITEM * conflboxlst)
961 {
962 /*
963  * ARGUMENT
964  * (LISTITEM *) conflboxlst - pointer to conflict list box list
965  * DESCRIPTION
966  * Updates the COMPROMISE status by checking the current number of
967  * material conflicts remaining and displaying the new status on the
968  * COMPROMISE environment window
969  *
970  * RETURNS
971  *
972  */
973      vratputf(CompWin, 11, 2, REVHIGHHELP, " %d ", listcnt(conflboxlst));
974 }
975
976 float getqty(void)
977 {
978 /*
979  * ARGUMENT
980  *
981  * DESCRIPTION
982  * Gets the substitution qty from the user
983  *
984  * RETURNS
985  */

```

```

986 *   (float) n           - qty selected
987 *   CANCEL_SEL - if ESC hit
988 *   ABORT_SEL  - if F3 hit
989 *
990 */
991
992 unsigned int key;
993 int decpntfound, cnt, digafterdecpt, done;
994 char str[10];
995
996 vratputf(CompWin, 13, 64, SELQTYBLINK, "%7.7s", " ? ");/* prompt for qty */
997 for (cnt = 0,
998     decpntfound = FALSE,
999     digafterdecpt = 0,
1000    done = FALSE,
1001    str[0] = '\0',
1002    key = getkey(),
1003    (! done);)
1004 {
1005     if (cnt == 0)
1006         vratputf(CompWin, 13, 64, SELQTY, "%7.7s", " ");/* clear qty space */
1007     switch (key)
1008     {
1009         case (ESC) : return ((float) CANCEL_SEL);/* ESC hit (cancel compromise) */
1010         case (F3) : return ((float) ABORTSUB_SEL);/* F3 hit (abort substitution) */
1011         case (BKSP) : if (cnt > 0)
1012         {
1013             cnt--;
1014             str[cnt] = '\0';
1015             vratputf(CompWin, 13, 64 + cnt, SELQTY, "%c", ' ');
1016             if (decpntfound && (digafterdecpt == 1))
1017                 digafterdecpt = 0;
1018             else if (decpntfound && (digafterdecpt == 0))
1019                 decpntfound = FALSE;
1020         }
1021         break;
1022         case (ENTR) : if (str[0] != '\0')
1023         {
1024             done = TRUE;
1025             dmpxtrakeys();
1026         }
1027         else
1028             /* prompt for qty again */
1029             vratputf(CompWin, 13, 64, SELQTYBLINK, "%7.7s", " ? ");
1030         break;
1031         case ('0') :
1032         case ('1') :
1033         case ('2') :
1034         case ('3') :
1035         case ('4') :
1036         case ('5') :
1037         case ('6') :
1038         case ('7') :
1039         case ('8') :
1040         case ('9') : if ((cnt <= 6) && (digafterdecpt != 1))
1041         {
1042             str[cnt] = key;
1043             str[cnt + 1] = '\0';
1044             vratputf(CompWin, 13, 64 + cnt, SELQTY, "%c", key);
1045             cnt++;
1046             if (decpntfound)
1047                 digafterdecpt = 1;
1048         }
1049         else
1050             vbeep();
1051         break;
1052         case ('.') : if ((cnt <= 6) && (! decpntfound))
1053         {
1054             decpntfound = TRUE;
1055             str[cnt] = key;
1056             str[cnt + 1] = '\0';
1057             vratputf(CompWin, 13, 64 + cnt, SELQTY, "%c", key);
1058             cnt++;
1059         }
1060     }
1061 }

```

```

1054         else
1055             vbeep();
1056         break;
1057         default : vbeep();           /* INVALID keystroke */
1058     }
1059     if (! done)
1060         key = getkey();
1061 }
1062 vratputf(CompWin, 13, 64, HELP, "%7.7s", str);
1063
1064 return (atof(str));
1065 }
1066
1067 int asktoacceptoverride(void)
1068 {
1069     /*
1070     * ARGUMENT
1071     *
1072     * DESCRIPTION
1073     *   Asks the user if override should be accepted
1074     *
1075     * RETURNS
1076     *
1077     *
1078     * DATABASES AFFECTED (whether direct or indirect)
1079     */
1080
1081     HWND win;
1082     unsigned int key;
1083
1084     win = dsplymessbox();
1085     vratputs(win, 1, 0, GRYBTXT, " **** PRIORITY OVERRIDE **** ");
1086     vratputs(win, 3, 0, GRYBTXT, "           possible! ");
1087     vratputs(win, 7, 0, KEYMESS, "   Accept override (Y): ? ");
1088     modattr(win, 7, 24, 3, FLSHCHR);
1089     while (1)
1090     {
1091         key = getkey();
1092         switch (key)
1093         {
1094             case ('Y') :
1095             case ('y') :
1096                 case (ENTR) : vratputs(win, 7, 24, HELP, " Y ");
1097                     delay(2);
1098                     delmessbox(win);
1099                     dmpxtrakeys();
1100                     return (TRUE);
1101             case ('N') :
1102             case ('n') : vratputs(win, 7, 24, HELP, " N ");
1103                     delay(2);
1104                     delmessbox(win);
1105                     dmpxtrakeys();
1106                     return (FALSE);
1107             default : vbeep();           /* INVALID keystroke */
1108         }
1109     }
1110 }
1111
1112 void delay(int sec)
1113 {
1114     /*
1115     * ARGUMENT
1116     *
1117     * DESCRIPTION
1118     *   Implements a delay with 1:1 second correspondence on 25MHz 386
1119     *
1120     * RETURNS
1121     *
1122     * DATABASES AFFECTED (whether direct or indirect)
1123     */

```

```

1124 */
1125
1126     long cnt, time;
1127
1128     time = sec;
1129     for (time = 0; time < sec; time++)
1130     {
1131         for (cnt = 300000L; cnt > 0; cnt--);
1132     }
1133
1134 int verifyabort(void)
1135 {
1136     /*
1137     * ARGUMENT
1138     *
1139     * DESCRIPTION
1140     *   Asks user to verify abort selection
1141     *
1142     * RETURNS
1143     *
1144     *
1145     * DATABASES AFFECTED (whether direct or indirect)
1146     *
1147     */
1148
1149     HWND win;
1150     unsigned int key;
1151
1152     win = dsplymessbox();
1153     vratputs(win, 0, 0, GRYBTXT, "
1154
1155     vratputs(win, 1, 0, GRYBTXT, " ** CANCEL COMPROMISE ** ");
1156     modattr(win, 1, 2, 2, SELQTYBLINK);
1157     modattr(win, 1, 26, 2, SELQTYBLINK);
1158     vratputs(win, 2, 0, GRYBTXT, "
1159     vratputs(win, 3, 0, GRYBTXT, "          chosen! ");
1160     vratputs(win, 4, 0, GRYBTXT, "
1161     vratputs(win, 7, 0, KEYMESS, "   Are you sure (N): ? ");
1162     modattr(win, 7, 23, 3, FLSHCHR);
1163     while (1)
1164     {
1165         key = getkey();
1166         switch (key)
1167         {
1168             case ('N') :
1169             case ('n') :
1170             case (ENTR) : vratputs(win, 7, 23, HELP, " N ");
1171                 delay(2);
1172                 vdelete(win, NONE);
1173                 vcurrent(CompWin);
1174                 dmpxtrakeys();
1175                 return (FALSE);
1176             case ('Y') :
1177             case ('y') : vratputs(win, 7, 23, HELP, " Y ");
1178                 delay(2);
1179                 vdelete(win, NONE);
1180                 vcurrent(CompWin);
1181                 dmpxtrakeys();
1182                 return (TRUE);
1183             default : vbeep();          /* INVALID keystroke */
1184         }
1185     }
1186
1187 int chkqtyvssupp(float qty, long recno)
1188 {
1189     /*
1190     * ARGUMENT
1191     *
1192     * DESCRIPTION
1193     *   Checks to makes sure qty entered for substitution is not

```

```

1193 *   more than is available in supply
1194 *
1195 * RETURNS
1196 *
1197 *
1198 * DATABASES AFFECTED (whether direct or indirect)
1199 *
1200 */
1201
1202     MATSUPREC * msuprec;
1203
1204     msuprec = GetSuppRec(recno);
1205     if (msuprec->qty >= qty)
1206     {
1207         free((MATSUPREC *) msuprec);
1208         return (TRUE);
1209     }
1210     free((MATSUPREC *) msuprec);
1211     return (FALSE);
1212 }
1213
1214 int askifcorrectqty(void)
1215 {
1216     /*
1217     * ARGUMENT
1218     *
1219     * DESCRIPTION
1220     *   Asks user if qty entered is correct
1221     *
1222     * RETURNS
1223     *
1224     * DATABASES AFFECTED (whether direct or indirect)
1225     *
1226     */
1227
1228     HWND win;
1229     unsigned int key;
1230
1231     win = dsplymessbox();
1232     vratputs(win, 2, 0, GRYBTXT, "   IS THIS QUANTITY CORRECT ?   ");
1233     vratputs(win, 6, 0, KEYMESS, "   Please verify (Y): ?   ");
1234     modattr(win, 6, 23, 3, FLSHCHR);
1235     while (1)
1236     {
1237         key = getkey();
1238         switch (key)
1239         {
1240             case ('Y') :
1241             case ('y') :
1242                 case (ENTR) : vratputs(win, 6, 23, HELP, " Y ");
1243                     delay(1);
1244                     vdelete(win, NONE);
1245                     vcurrent(CompWin);
1246                     dmpxtrakeys();
1247                     return (TRUE);
1248             case ('N') :
1249             case ('n') : vratputs(win, 6, 23, HELP, " N ");
1250                     delay(1);
1251                     vdelete(win, NONE);
1252                     vcurrent(CompWin);
1253                     dmpxtrakeys();
1254                     return (FALSE);
1255             default : vbeep(); /* INVALID keystroke */
1256         }
1257     }
1258 }
1259
1260 void dsplysubaccepted(void)
1261 {
1262     /*
1263     * ARGUMENT
1264     *

```

```

1262 *
1263 * DESCRIPTION
1264 *   Displays "substitution accepted" box
1265 *
1266 * RETURNS
1267 *
1268 *
1269 * DATABASES AFFECTED (whether direct or indirect)
1270 *
1271 */
1272
1273     HWND win;
1274
1275     win = dsplymessbox();
1276     vratputs(win, 1, 0, GRYBTXT, " **** MATERIAL **** ");
1277     vratputs(win, 3, 0, GRYBTXT, " SUBSTITUTION ");
1278     vratputs(win, 6, 0, KEYALERT, " ACCEPTED ");
1279     delay(3);
1280     delmessbox(win);
1281     vcurrent(CompWin);
1282 }
1283
1284 void alerttoqtylimit(long recno)
1285 {
1286 /*
1287 * ARGUMENT
1288 *
1289 * DESCRIPTION
1290 *   Displays supply qty limit
1291 *
1292 * RETURNS
1293 *
1294 *
1295 * DATABASES AFFECTED (whether direct or indirect)
1296 *
1297 */
1298
1299     HWND win;
1300     MATSUPREC * msuprec;
1301
1302     msuprec = GetSuppRec(recno);
1303     win = dsplymessbox();
1304     vratputs(win, 1, 0, GRYBTXT, " **** I N V A L I D **** ");
1305     vratputs(win, 3, 0, GRYBTXT, " QUANTITY ENTRY ");
1306     vratputf(win, 5, 0, GRYBTXT, " Total qty avail: %7.1f %3.3s ", msuprec->qty, msuprec->unit);
1307
1308     modattr(win, 5, 18, 11, KEYMESS);
1309     vratputs(win, 7, 0, KEYALERT, " < Press any key > ");
1310     getkey();
1311     delay(1);
1312     dmpxtrakeys();
1313     delmessbox(win);
1314     free((MATSUPREC *) msuprec);
1315     vcurrent(CompWin);
1316     return;
1317 }

```

```

1  /*
2  * dbutil.c
3  *
4  *      utility functions access repair info in dbases
5  */
6
7  #include <stdlib.h>
8  #include <string.h>
9  #include <stdio.h>
10
11 #include <d4base.h>
12 #include <dw.h>
13
14 #include "rmrsutil.h"
15 #include "sysdefs.h"
16 #include "rmrserr.h"
17 #include "miscutil.h"
18
19
20 REPDATANODE * GetRepData(long repid)
20 {
21  /*
22  * ARGUMENT
23  *      (long) repid - repair id
24  *
25  * DESCRIPTION
26  *      Retrieves repair data from databases for use in repair data block
27  *
28  * RETURNS
29  *      (REPDATANODE *) pointer to structure containing the information
30  *      needed for the repair data block as used in the COMPROMISE mode
31  *
32  * DATABASES AFFECTED (whether direct or indirect)
33  *      repair_dbf (repair_ndx1 - REPID index      ) - d4seek used
34  *      repinfo_dbf (repinfo_ndx1 - REPID index    ) - d4seek used
35  */
36
37
38     REPDATANODE * buffer;
39
40     buffer = (REPDATANODE *) malloc(sizeof (REPDATANODE));
41     d4select(repair_dbf);
42     i4select(repair_ndx1);
43     d4seek_double(repid);
44     buffer->repid = repid;
45     buffer->facnum = f4long(f4ref("FACNUM"));
46     buffer->priority = f4int(f4ref("PRIORITY"));
47
48     repinfo_dbf = d4use_excl("REPINFO.DBF");
49     d4select(repinfo_dbf);
50     repinfo_ndx1 = i4open("REPINDEX1.NDX");
51     i4select(repinfo_ndx1);
52     d4seek_double(repid);
53     buffer->elenum = f4int(f4ref("ELENUM"));
54     strcpy(buffer->eledescstr, f4str(f4ref("ELEDESC")));
55     strcpy(buffer->dammodestr, f4str(f4ref("DAMMODE")));
56     strcpy(buffer->repstgystr, f4str(f4ref("REPSTGY")));
57     trimstr(buffer->eledescstr);
58     trimstr(buffer->dammodestr);
59     trimstr(buffer->repstgystr);
60
61     d4select(repinfo_dbf);
62     d4close();
63     return (buffer);
64 }
65
66 MATSUPREC * GetSuppRec(long recno)
66 {
67  /*
68  * ARGUMENT
69  *      (long) recno - record number of desired material supply record
70  *
71  * DESCRIPTION
72  *      Retrieves entire specified material supply record from material

```

```

73  *   supply database
74  *
75  * RETURNS
76  *   (MATSUPREC *) pointer to structure containing the information
77  *   from the specified material supply record
78  *
79  * DATABASES AFFECTED (whether direct or indirect)
80  *   matsup_dbf (no index used)
81  *
82  */
83
84  MATSUPREC * buffer;
85
86  d4select(matsup_dbf);
87  if (d4go(recno))
88  {
89      u4error(RECACESERR, "Record access error", (char *) 0);
90      return ((MATSUPREC *) NULL);
91  }
92  buffer = (MATSUPREC *) malloc(sizeof (MATSUPREC));
93
94  buffer->matid = f4long(f4ref("MATID"));
95  strcpy(buffer->matdesc, f4str(f4ref("MATDESC")));
96  buffer->qty = f4double(f4ref("QTY"));
97  strcpy(buffer->unit, f4str(f4ref("UNIT")));
98  return (buffer);
99
100 int GetFacPrty(long facnum)
100 {
101 /*
102  * ARGUMENT
103  *   (long) facnum - facility number designation
104  *
105  * DESCRIPTION
106  *   Retrieves AB assigned facility priority from AB specific facility
107  *   priority database
108  *
109  * RETURNS
110  *   (int)   n - facility priority
111  *           0 - if facility listing not found
112  *
113  * DATABASES AFFECTED (whether direct or indirect)
114  *   facprt_dbf (facprt_ndxl)
115  *
116  */
117
118  int status, prty;
119
120  /* fetch associated facility priority from FACPTY.DBF */
121  open_facprt(0);
122  d4select(facprt_dbf);
123  i4select(facprt_ndxl);
124  status = d4seek_double(facnum);
125  prty = f4int(f4ref("PRIORITY"));
126  d4close();
127  if (! status)
128      return (prty);
129  else
130      return (0);
131 }

```

```

1 /*
2  * itemfns.cpp
3  *
4  *   This file contains the functions that are called from the
5  *   menu bar pull-down selections. These functions are listed
6  *   below beside their menu selection:
7  *
8  *   System      > About ...      : SystAbotFn()
9  *                Proc info ...   : SystInfoFn()
10 *                DOS shell        : SystXDOSFn()
11 *                Comm Prog        : SystCOMMFN()
12 *                Quit              : SystQuitFn()
13 *   Init/Setup  > Change def specs : InitDefFn()
14 *                Load new supply file > AB mat supply file : InitMatlFn()
15 *                AB equip supply file : InitEquipFn()
16 *   Process     > Auto Mode        : ProcAutoFn()
17 *                Single facility   : ProcFac1Fn()
18 *                Compromise Retry  : ProcCompFn()
19 *                Cancel repair     : ProcCancFn()
20 *   Schedule    > Auto Mode        : SchdAutoFn()
21 *                Upgrade priority  : SchdUpdFn()
22 *                Set order         : SchdOrdFn()
23 *   ChangeStatus > Completed        : ChngCompFn()
24 *                Canceled         : ChngCancFn()
25 *                Restore          : ChngRestFn()
26 *                Delete All       : ChngDelAFn()
27 *   Gen Report  > Repair Schedule  > by repair team : GnRpSchdTeamFn()
28 *                by facility      : GnRpSchdFac1Fn()
29 *                Completed repairs : GnRpCompFn()
30 *                Canceled repairs  : GnRpCancFn()
31 *                Suspended repairs : GnRpSuspFn()
32 *                Possible repairs  : GnRpPossFn()
33 *                Allocated materials : GnRpAMatFn()
34 *                Used materials    : GnRpUMatFn()
35 *                AB materials supply : GnRpABMtFn()
36 *                AB equipment supply : GnRpABEqFn()
37 *
38 *
39 */
40
41 #include <stdio.h>
42 #include <dir.h>
43 #include <stdlib.h>
44 #include <string.h>
45 #include "dw.h"
46 #include "dwmenu.h"
47 #include "dwsystem.h"
48 #include "d4base.h"
49
50 #include "rmrs.h"
51
52 extern void IntroScreen();
53
54 HWND win;
55
56 #define ERROREND -1
57
58 int SystAbotFn(void)
59 {
60     IntroScreen();
61     return (0);
62 }
63
64 int SystInfoFn(void)
65 {
66     long i;
67     HWND InfoFnWin;
68     unsigned short int * screenbucket = NULL;
69     InfoFnWin = vcreat(7, 40, ERR, YES);
70     d4close_all();
71     vlocate(InfoFnWin, 9, 20);
72     vframe(InfoFnWin, ERR, FRDCUBLE);
73     vshadow(InfoFnWin, CURRENT, SHADOW100, BOTTOMRIGHT);
74     vmtitle(InfoFnWin, _TOP, CENTERJUST, ERR, " RMRS Processing Information ");

```

```

75     visible(InfoFnWin, YES, YES);
76     for (i = 0; i < 3200000; i++);
77
78     vdelete(InfoFnWin, NONE);
79     return (0);
80 }
81
82 int SystXDOSFn(void)
83 {
84     int curpos;
85     HWND win;
86     int stat;
87
88     win = vcreat(21, 78, REVNORML, YES);
89     vlocate(win, 2, 1);
90     vframe(win, REVHIGHNORML, FRSINGLE);
91     vtitle(win, REVHIGHNORML, "< DOS Window - Type EXIT to quit >");
92     visible(win, YES, YES);
93
94
95     stat = vredir(ON, win);
96     if (stat < DWSUCCESS)
97     {
98         printf("Redirection Failure\n");
99         vdelete(win, NONE);
100     }
101     else
102     {
103         vratps(1, 1, NORML, " Type EXIT to return to RMRS ... ");
104         curdrag(win, ON);
105         vredir(ON, win);
106         system("command");
107     }
108     vredir(OFF, win);
109     vredir(0, 0);
110     vclear(win);
111     vdelete(win, NONE);
112
113     return (0);
114 }
115
116 int SystQuitFn(void)
117 {
118     HWND QuitFnWin;
119     char reply;
120
121     QuitFnWin = vcreat(7, 40, EMPHNORML, YES);
122     vlocate(QuitFnWin, 9, 20);
123     vframe(QuitFnWin, EMPHNORML, FRDOUBLE);
124     vshadow(QuitFnWin, CURRENT, SHADOW75, BOTTOMRIGHT);
125     visible(QuitFnWin, YES, YES);
126     vatputs(QuitFnWin, 3, 2, "Are you sure you want to quit (Y/N)?");
127     if (((reply = getkey()) == 'Y') || (reply == 'y'))
128     {
129         d4init_undo();
130         fcloseall();
131         /* reset default video mode */
132         vexit(0);
133     }
134     vdelete(QuitFnWin, NONE);
135     return (0);
136 }
137
138 int InitMatlFn(void)
139 {
140     long i;
141     HWND UserFnWin2;
142
143     UserFnWin2 = vcreat(7, 40, ERR, YES);
144     vlocate(UserFnWin2, 9, 20);
145     vframe(UserFnWin2, ERR, FRDOUBLE);
146     vshadow(UserFnWin2, CURRENT, SHADOW100, BOTTOMRIGHT);
147     vmttitle(UserFnWin2, _TOP, CENTERJUST, ERR, " none ");
148     visible(UserFnWin2, YES, YES);

```

```

149     for (i = 0; i < 3200000; i++);
150
151
152     vdelete(UserFnWin2, NONE);
153     return (0);
154 }
155
156 int InitEquiFn(void)
157 {
158     long i;
159     HWND UserFnWin3;
160
161     UserFnWin3 = vcreat(7, 40, ERR, YES);
162     vlocate(UserFnWin3, 9, 20);
163     vframe(UserFnWin3, ERR, FRDOUBLE);
164     vshadow(UserFnWin3, CURRENT, SHADOW100, BOTTOMRIGHT);
165     vmttitle(UserFnWin3, TOP, CENTERJUST, ERR, " none ");
166     visible(UserFnWin3, YES, YES);
167     for (i = 0; i < 3200000; i++);
168
169
170     vdelete(UserFnWin3, NONE);
171     return (0);
172 }
173
174
175 int ProcAutoFn(void)
176 {
177     int DoProcessAuto();
178     long i;
179     short x;
180     char str[6];
181     extern HWND FlHelpWin;
182     int repair_ref;
183     int done;
184     struct ffbk file;
185
186     win = vcreat(7, 50, EMPHNORML, YES);
187     vlocate(win, 9, 20);
188     vframe(win, EMPHNORML, FRDOUBLE);
189     vshadow(win, CURRENT, SHADOW75, BOTTOMRIGHT);
190     vmttitle(win, TOP, CENTERJUST, EMPHNORML, " Auto-mode Repair Processing ");
191     visible(win, YES, YES);
192
193     DoProcessAuto();
194
195     vdelete(win, NONE);
196     return (0);
197 }
198
199 int ProcFacIFn(void)
200 {
201     char * srchfor = "*.EQP";
202     MENUHDR * PDESDirHdr;
203     LISTITEM * file, * temp;
204     HWND win;
205     int num, row = 3, col = 3;
206
207     win = vopen(18, 60, REVNORML, REVNORML, FRDOUBLE, "< PDES files remaining to be processed >");
208     vatputs(win, 2, 3, "Select facility");
209     vatputs(win, 3, 3, " to process ->>");
210     vatputs(win, 16, 3, "Ctrl-Enter to end selection.");
211     pclrattrib(NORML);
212     pclrchar((char) 0xb0);
213     sclear();
214     PDESDirHdr = MNUCreateHdr(POPUP);
215     PDESDirHdr->toprow = 3;
216     PDESDirHdr->topcol = 30;
217     PDESDirHdr->maxrows = 9;
218     num = MNUSelectFiles(PDESDirHdr, srchfor, & file);
219     vclear(win);
220     vatputf(win, 1, 3, "%d items selected", num);
221     if (num)

```

```

221     {
222         temp = file;
223         do
224         {
225             vatputs(win, row, col, temp->listitem);
226             if (row++ > 17)
227             {
228                 row = 3;
229                 col += 15;
230             }
231             temp = temp->lnext;
232         } while (temp != file);
233     }
234     vatputs(win, 15, 3, "Hit any key to exit to main menu. ");
235     getkey();
236     MNUDeleteMenu(PDESDirHdr);
237     vdelete(win, NONE);
238     return (1);
239 }
240 int ProcCompFn(void)
241 {
242     long i;
243     HWND UserFnWin6;
244
245     UserFnWin6 = vcreat(7, 40, ERR, YES);
246     vlocate(UserFnWin6, 9, 20);
247     vframe(UserFnWin6, ERR, FRDOUBLE);
248     vshadow(UserFnWin6, CURRENT, SHADOW100, BOTTOMRIGHT);
249     vmtitle(UserFnWin6, TOP, CENTERJUST, ERR, " none ");
250     visible(UserFnWin6, YES, YES);
251     for (i = 0; i < 3200000; i++);
252
253     vdelete(UserFnWin6, NONE);
254     return (0);
255 }
256
257 int ProcCancFn(void)
258 {
259     long i;
260     HWND UserFnWin7;
261
262     UserFnWin7 = vcreat(7, 40, ERR, YES);
263     vlocate(UserFnWin7, 9, 20);
264     vframe(UserFnWin7, ERR, FRDOUBLE);
265     vshadow(UserFnWin7, CURRENT, SHADOW100, BOTTOMRIGHT);
266     vmtitle(UserFnWin7, TOP, CENTERJUST, ERR, " none ");
267     visible(UserFnWin7, YES, YES);
268     for (i = 0; i < 3200000; i++);
269
270     vdelete(UserFnWin7, NONE);
271     return (0);
272 }
273
274 int SchdFullFn(void)
275 {
276     SchedAuto(1);
277     return (0);
278 }
279
280 int SchdIncFn(void)
281 {
282     SchedAuto(0);
283     return (0);
284 }
285
286 int SchdPriorFn(void)
287 {
288     HWND InfoFnWin;
289     unsigned short int * screenbucket = NULL;

```

```
292     int curpos;
293     screenbucket = SaveScreen(screenbucket, & curpos);
294     pcuron();
295     open_repair(1);
296     b4browse(b4quick_browse, b4quick_edit);
297     d4close();
298     pcuroff();
299     RestoreScreen(screenbucket, & curpos);
300     free(screenbucket);
301     return (0);
302 }
303
304 int ChngCompFn(void)
305 {
306     long i;
307     HWND UserFnWin11;
308
309     UserFnWin11 = vcreat(7, 0, ERR, YES);
310     vlocate(UserFnWin11, 9, 20);
311     vframe(UserFnWin11, ERR, FRDOUBLE);
312     vshadow(UserFnWin11, CURRENT, SHADOW100, BOTTOMRIGHT);
313     vmtitle(UserFnWin11, TOP, CENTERJUST, ERR, " none ");
314     visible(UserFnWin11, YES, YES);
315     for (i = 0; i < 3200000; i++);
316
317     vdelete(UserFnWin11, NONE);
318     return (0);
319 }
320
321 int ChngCancFn(void)
322 {
323     long i;
324     HWND UserFnWin12;
325
326     UserFnWin12 = vcreat(7, 40, ERR, YES);
327     vlocate(UserFnWin12, 9, 20);
328     vframe(UserFnWin12, ERR, FRDOUBLE);
329     vshadow(UserFnWin12, CURRENT, SHADOW100, BOTTOMRIGHT);
330     vmtitle(UserFnWin12, TOP, CENTERJUST, ERR, " none ");
331     visible(UserFnWin12, YES, YES);
332     for (i = 0; i < 3200000; i++);
333
334     vdelete(UserFnWin12, NONE);
335     return (0);
336 }
337
338 int ChngRestFn(void)
339 {
340     long i;
341     HWND UserFnWin13;
342
343     UserFnWin13 = vcreat(7, 40, ERR, YES);
344     vlocate(UserFnWin13, 9, 20);
345     vframe(UserFnWin13, ERR, FRDOUBLE);
346     vshadow(UserFnWin13, CURRENT, SHADOW100, BOTTOMRIGHT);
347     vmtitle(UserFnWin13, TOP, CENTERJUST, ERR, " none ");
348     visible(UserFnWin13, YES, YES);
349     for (i = 0; i < 3200000; i++);
350
351     vdelete(UserFnWin13, NONE);
352     return (0);
353 }
354
355 int ChngDeLaFn(void)
356 {
357     long i;
358     HWND UserFnWin14;
359
360     UserFnWin14 = vcreat(7, 40, ERR, YES);
361     vlocate(UserFnWin14, 9, 20);
362     vframe(UserFnWin14, ERR, FRDOUBLE);
```

```

366     vshadow(UserFnWin14, CURRENT, SHADOW100, BOTTOMRIGHT);
367     vmtitle(UserFnWin14, _TOP, CENTERJUST, ERR, " none ");
368     visible(UserFnWin14, YES, YES);
369     for (i = 0; i < 3200000; i++);
370
371
372     vdelete(UserFnWin14, NONE);
373     return (0);
374 }
375
376
377 int EdFlsRepFn(void)
378 {
379     unsigned short int * screenbucket = NULL;
380     int curpos;
381     screenbucket = SaveScreen(screenbucket, & curpos);
382     pcuron();
383     open_repair(1);
384     b4browse(b4quick_browse, b4quick_edit);
385     d4close();
386     pcuroff();
387     RestoreScreen(screenbucket, & curpos);
388     free(screenbucket);
389     return (0);
390 }
391
392 int EdFlsEqSupFn(void)
393 {
394     unsigned short int * screenbucket = NULL;
395     int curpos;
396     screenbucket = SaveScreen(screenbucket, & curpos);
397     pcuron();
398     open_eqsup(1);
399     b4browse(b4quick_browse, b4quick_edit);
400     d4close();
401     pcuroff();
402     RestoreScreen(screenbucket, & curpos);
403     free(screenbucket);
404     return (0);
405 }
406 int EdFlsEqReqFn(void)
407 {
408     unsigned short int * screenbucket = NULL;
409     int curpos;
410     screenbucket = SaveScreen(screenbucket, & curpos);
411     pcuron();
412     open_eqpreq(1);
413     b4browse(b4quick_browse, b4quick_edit);
414     d4close();
415     pcuroff();
416     RestoreScreen(screenbucket, & curpos);
417     free(screenbucket);
418     return (0);
419 }
420 int EdFlsMaSupFn(void)
421 {
422     unsigned short int * screenbucket = NULL;
423     int curpos;
424     screenbucket = SaveScreen(screenbucket, & curpos);
425     pcuron();
426     open_matsup(1);
427     b4browse(b4quick_browse, b4quick_edit);
428     d4close();
429     pcuroff();
430     RestoreScreen(screenbucket, & curpos);
431     free(screenbucket);
432     return (0);
433 }
434
435 int EdFlsMaReqFn(void)
436 {
437     unsigned short int * screenbucket = NULL;
438     int curpos;
439     screenbucket = SaveScreen(screenbucket, & curpos);

```

```
440     pcuron();
441     open_matreq(1);
442     b4browse(b4quick_browse, b4quick_edit);
443     d4close();
444     pcuroff();
445     RestoreScreen(screenbucket, & curpos);
446     free(screenbucket);
447     return (0);
448 }
449
450 int EdFlsRepInfoFn(void)
451 {
452     unsigned short int * screenbucket = NULL;
453     int curpos;
454     screenbucket = SaveScreen(screenbucket, & curpos);
455     pcuron();
456     open_repinfo(1);
457     b4browse(b4quick_browse, b4quick_edit);
458     d4close();
459     pcuroff();
460     RestoreScreen(screenbucket, & curpos);
461     free(screenbucket);
462     return (0);
463 }
464
465 int EdFlsFacPrtyFn(void)
466 {
467     unsigned short int * screenbucket = NULL;
468     int curpos;
469     screenbucket = SaveScreen(screenbucket, & curpos);
470     pcuron();
471     open_facprt(1);
472     b4browse(b4quick_browse, b4quick_edit);
473     d4close();
474     pcuroff();
475     RestoreScreen(screenbucket, & curpos);
476     free(screenbucket);
477     return (0);
478 }
479
480 int EdFlsResetFn(void)
481 {
482     HWND win;
483
484     /* WINDOW STUFF */
485     win = vcreat(7, 40, EMPHNORML, YES);
486     vautoshd(win, CURRENT, SHADOW75, BOTTOMRIGHT);
487     vmtitle(win, _TOP, CENTERJUST, EMPHNORML, " Data Reset ");
488     vlocate(win, 9, 20);
489     vframe(win, EMPHNORML, FRDOUBLE);
490     visible(win, YES, YES);
491
492     vatputs(win, 1, 2, "Reseting Data Files ...");
493     vatputs(win, 3, 2, "Percent Complete : 0%");
494     d4close_all();
495     fcloseall();
496
497     open_repair(0);
498     d4zap(1L, d4rccount());
499     d4close_all();
500     vatputs(win, 3, 2, "Percent Complete : 20%");
501
502     open_matreq(0);
503     d4zap(1L, d4reccount());
504     d4close_all();
505     vatputs(win, 3, 2, "Percent Complete : 40%");
506
507     open_eqpreq(0);
508     d4zap(1L, d4reccount());
509     d4close_all();
510     vatputs(win, 3, 2, "Percent Complete : 60%");
511
512     open_repinfo(0);
513     d4zap(1L, d4reccount());
```

```

514     d4close_all();
515     vatputs(win, 3, 2, "Percent Complete : 80%");
516
517     open_matsup(0);
518     d4top();
519     while (! d4eof())
520     {
521         f4r_double(f4ref("QTY"), 1000.0);
522         d4skip(1);
523     }
524     d4pack();
525     d4close_all();
526
527     open_eqpsup(0);
528     d4pack();
529     d4close_all();
530     fcloseall();
531
532     vatputs(win, 3, 2, "Percent Complete : 100%");
533     vatputs(win, 5, 2, "press any key ...");
534     getch();
535     vdelete(win, NONE);
536     return (0);
537 }
538 int GnRpSchdFn(void)
539 {
540     ReprtSched();
541     return (0);
542 }
543
544 int GnRpGantFn(void)
545 {
546     ReprtGantt();
547     return (0);
548 }
549
550 int GnRpCompFn(void)
551 {
552     long i;
553     HWND UserFnWin17;
554
555     UserFnWin17 = vcreat(7, 40, ERR, YES);
556     vlocate(UserFnWin17, 9, 20);
557     vframe(UserFnWin17, ERR, FRDOUBLE);
558     vshadow(UserFnWin17, CURRENT, SHADOW100, BOTTOMRIGHT);
559     vmtitle(UserFnWin17, TOP, CENTERJUST, ERR, " none ");
560     visible(UserFnWin17, YES, YES);
561     for (i = 0; i < 3200000; i++);
562
563     vdelete(UserFnWin17, NONE);
564     return (0);
565 }
566
567
568 int GnRpCancFn(void)
569 {
570     long i;
571     HWND UserFnWin18;
572
573     UserFnWin18 = vcreat(7, 40, ERR, YES);
574     vlocate(UserFnWin18, 9, 20);
575     vframe(UserFnWin18, ERR, FRDOUBLE);
576     vshadow(UserFnWin18, CURRENT, SHADOW100, BOTTOMRIGHT);
577     vmtitle(UserFnWin18, TOP, CENTERJUST, ERR, " none ");
578     visible(UserFnWin18, YES, YES);
579     for (i = 0; i < 3200000; i++);
580
581     vdelete(UserFnWin18, NONE);
582     return (0);
583 }
584
585
586 int GnRpSuspFn(void)

```

```
587 {
588     long i;
589     HWND UserFnWin19;
590
591     UserFnWin19 = vcreat(7, 40, ERR, YES);
592     vlocate(UserFnWin19, 9, 20);
593     vframe(UserFnWin19, ERR, FRDOUBLE);
594     vshadow(UserFnWin19, CURRENT, SHADOW100, BOTTOMRIGHT);
595     vmttitle(UserFnWin19, TOP, CENTERJUST, ERR, " none ");
596     visible(UserFnWin19, YES, YES);
597     for (i = 0; i < 3200000; i++);
598
599
600     vdelete(UserFnWin19, NONE);
601     return (0);
602 }
603
604 int GnRpPossFn(void)
605 {
606     long i;
607     HWND UserFnWin20;
608
609     UserFnWin20 = vcreat(7, 40, ERR, YES);
610     vlocate(UserFnWin20, 9, 20);
611     vframe(UserFnWin20, ERR, FRDOUBLE);
612     vshadow(UserFnWin20, CURRENT, SHADOW100, BOTTOMRIGHT);
613     vmttitle(UserFnWin20, TOP, CENTERJUST, ERR, " none ");
614     visible(UserFnWin20, YES, YES);
615     for (i = 0; i < 3200000; i++);
616
617
618     vdelete(UserFnWin20, NONE);
619     return (0);
620 }
621
622 int GnRpAMatFn(void)
623 {
624     long i;
625     HWND UserFnWin21;
626
627     UserFnWin21 = vcreat(7, 40, ERR, YES);
628     vlocate(UserFnWin21, 9, 20);
629     vframe(UserFnWin21, ERR, FRDOUBLE);
630     vshadow(UserFnWin21, CURRENT, SHADOW100, BOTTOMRIGHT);
631     vmttitle(UserFnWin21, TOP, CENTERJUST, ERR, " none ");
632     visible(UserFnWin21, YES, YES);
633     for (i = 0; i < 3200000; i++);
634
635
636     vdelete(UserFnWin21, NONE);
637     return (0);
638 }
639
640 int GnRpUMatFn(void)
641 {
642     long i;
643     HWND UserFnWin22;
644
645     UserFnWin22 = vcreat(7, 40, ERR, YES);
646     vlocate(UserFnWin22, 9, 20);
647     vframe(UserFnWin22, ERR, FRDOUBLE);
648     vshadow(UserFnWin22, CURRENT, SHADOW100, BOTTOMRIGHT);
649     vmttitle(UserFnWin22, TOP, CENTERJUST, ERR, " none ");
650     visible(UserFnWin22, YES, YES);
651     for (i = 0; i < 3200000; i++);
652
653
654     vdelete(UserFnWin22, NONE);
655     return (0);
656 }
657
658 int GnRpABMtFn(void)
659 {
660     long i;
```

```
661     HWND UserFnWin23;
662
663     UserFnWin23 = vcreat(7, 40, ERR, YES);
664     vlocate(UserFnWin23, 9, 20);
665     vframe(UserFnWin23, ERR, FRDOUBLE);
666     vshadow(UserFnWin23, CURRENT, SHADOW100, BOTTOMRIGHT);
667     vmtitle(UserFnWin23, TOP, CENTERJUST, ERR, " none ");
668     visible(UserFnWin23, YES, YES);
669     for (i = 0; i < 3200000; i++);
670
671
672     vdelete(UserFnWin23, NONE);
673     return (0);
674 }
675
676 int GnRpABEqFn(void)
677 {
678     long i;
679     HWND UserFnWin24;
680
681     UserFnWin24 = vcreat(7, 40, ERR, YES);
682     vlocate(UserFnWin24, 9, 20);
683     vframe(UserFnWin24, ERR, FRDOUBLE);
684     vshadow(UserFnWin24, CURRENT, SHADOW100, BOTTOMRIGHT);
685     vmtitle(UserFnWin24, TOP, CENTERJUST, ERR, " none ");
686     visible(UserFnWin24, YES, YES);
687     for (i = 0; i < 3200000; i++);
688
689
690     vdelete(UserFnWin24, NONE);
691     return (0);
692 }
```

```

1  #include <stdio.h>
2  #include <string.h>
3  #include <stdlib.h>
4
5  #include "g4char.h"
6  #include "w4.h"
7  #include "d4base.h"
8  #include "PWRBASE.H"
9  #include "LKUmod.H"
10 #include "STR.H"
11
12 #define NO_ERROR
13
14 /* Local Prototypes *****/
15
16 long  LKUmodNextField(long, int);
17 void  LKUmodGetRefs(void);
18 int   LKUmodLookWindowWidth(void);
19 int   LKUmodfield_width(long);
20 int   LKUmodall_fields(int);
21 void  LKUmodDisplayRow(int, long);
22 void  LKUmodDisplayLookup(int);
23 void  LKUmodPositionLookup(long);
24
25 /* Globals *****/
26
27 extern copyright;
28 extern COPYRIGHT_NOTICE();
29
30 static LOOKUP LKUmodLookup;
31
32 static int     LKUmodfirstTime = 1;
33 static int     LKUmodshowFields;
34 static int     LKUmodcurrentRef;
35 static int     LKUmodtotalWndwWidth;
36 static int     LKUmodtotalFields;
37 static int     LKUmodforceCols;
38 static int     LKUmodallowExtra = 1;
39 static char    LKUmoddateFormat[30];
40 static int     LKUmodscrollFields;
41 static int     LKUmodtruncOutput;
42
43 static int     LKUmodwindowWidth;
44 static long    LKUmodfieldRefs[TOTALFIELDS];
45 static int     LKUmodwidths[TOTALFIELDS];
46
47
48 long LKUmodNextField(long fRef, int curIndex)
49 {
50     int found;
51     int i = 0;
52
53     if (! LKUmodshowFields)
54         return (long) fRef;
55     found = 0;
56     while (i < curIndex)
57     {
58         if (LKUmodfieldRefs[i] == fRef)
59         {
60             found = 1;
61             break;
62         }
63         i++;
64     }
65     if (! found)
66         return (long) fRef;
67     return (long) - 1;
68 }
69
70 int LKUmodset_columns(int columnWidth)
71 {
72     int i = 0;
73     if (columnWidth != -1)

```

```

72     {
73         if (columnWidth > CRTWIDTH || columnWidth < 0)
74             columnWidth = 0;
75
76         while (i < TOTALFIELDS)
77         {
78             LKUmodwidths[i] = columnWidth;
79             i++;
80         }
81         return (int) LKUmodwidths[i];
82     }
83
84 void LKUmodreset_columns(void)
85 {
86     LKUmodset_columns(0);
87 }
88
89 void LKUmodGetRefs(void)
90 {
91     int i = LKUmodshowFields;
92     int j = 0;
93     int startNum;
94
95     LKUmodtotalFields = f4num_fields();
96
97     if (LKUmodshowFields)
98         while (j < i)
99         {
100             LKUmodfieldRefs[j] = f4ref(LKUmodlookup.lField[j].fieldName);
101             if (! LKUmodwidths[j] || LKUmodwidths[j] == -2)
102                 if (f4type(LKUmodfieldRefs[j]) == 'M')
103                     LKUmodwidths[j] = 4;
104                 else
105                     if (f4type(LKUmodfieldRefs[j]) == 'D')
106                         LKUmodwidths[j] = strlen(LKUmoddateFormat);
107                     else
108                         if (LKUmodwidths[j] == -2)
109                             LKUmodwidths[j] = - (f4width(LKUmodfieldRefs[j]));
110                         else
111                             LKUmodwidths[j] = f4width(LKUmodfieldRefs[j]);
112             j++;
113         }
114     if (LKUmodtotalFields >= TOTALFIELDS)
115         LKUmodtotalFields = TOTALFIELDS - 1;
116     while (i < LKUmodtotalFields)
117     {
118         LKUmodfieldRefs[i] = LKUmodNextField(f4j_ref(i + 1), i);
119         if (LKUmodfieldRefs[i] == -1)
120         {
121             startNum = 0;
122             while (startNum < LKUmodtotalFields)
123             {
124                 if ((LKUmodfieldRefs[i] = LKUmodNextField((long) startNum, (int) LKUmodtotalFields)) != (long) - 1)
125                     break;
126                 startNum++;
127             }
128         }
129         if (! LKUmodwidths[i])
130             if (f4type(LKUmodfieldRefs[i]) == 'M')
131                 LKUmodwidths[i] = 4;
132             else
133                 if (f4type(LKUmodfieldRefs[i]) == 'D')
134                     LKUmodwidths[i] = strlen(LKUmoddateFormat);
135                 else
136                     LKUmodwidths[i] = f4width(LKUmodfieldRefs[i]);
137         if (! LKUmodshowFields || i < MAXLKUFIELDS)
138             LKUmodlookup.lField[i].attribute = LKUmodLookup.windowAttr;
139         i++;
140     }
141 }

```

```

140 int LKUModLookWindowWidth(void)
141 {
142     int i;
143     int totalWidth = 1, tWidth1, tWidth2, lastWidth, joinField;
144
145     for (i = 0; i < ((LKUModshowFields > 0) ? LKUModshowFields : MAXLKUFIELDS); i++)
146     {
147         tWidth1 = LKUModfield_width(LKUModfieldRefs[i]) + 1;
148         if (LKUModwidths[i] < 0)
149         {
150             tWidth2 = - (LKUModwidths[i]) + 1;
151             joinField = 1;
152             while ((LKUModwidths[i] < 0) &&
153                 (i < ((LKUModshowFields > 0) ? LKUModshowFields : MAXLKUFIELDS)))
154                 i++;
155         }
156         else
157         {
158             tWidth2 = LKUModwidths[i] + 1;
159             joinField = 0;
160         }
161
162         if (! LKUModfrceCols)
163         {
164             if (tWidth2 && joinField)
165             {
166                 lastWidth = tWidth2;
167             }
168             else
169             {
170                 if (tWidth1 < tWidth2)
171                 {
172                     lastWidth = tWidth1;
173                 }
174                 else
175                 {
176                     lastWidth = tWidth2;
177                 }
178             }
179         }
180         else
181             lastWidth = tWidth2;
182
183         totalWidth += lastWidth;
184
185         if (totalWidth > CRTWIDTH)
186         {
187             totalWidth -= ((totalWidth - CRTWIDTH) + 1);
188             break;
189         }
190     }
191
192     return (totalWidth + 1);
193 }
194
195 int LKUModfield_width(long fieldRef)
196 {
197     int length;
198
199     if (f4type(fieldRef) == 'M')
200         length = 4;
201     else
202     {
203         if (f4type(fieldRef) == 'D')
204             length = strlen(LKUModdateFormat);
205         else
206             length = f4width(fieldRef);
207     }
208
209     return length;
210 }
211
212 int LKUModall_fields(int refOffset)
213 {
214     int i = 0, tFields = 0, currntPos = 0, reachedEnd = 0;

```

```

201     int tWidth1, tWidth2, joinField;
202
203     while (1)
204     {
205         if ((refOffset + 1) >= LKumodtotalFields)
206         {
207             reachedEnd = 1;
208             break;
209         }
210         if (currntPos >= LKumodtotlWndwWidth)
211         {
212             break;
213         }
214         tWidth1 = LKumodfield_width(LKumodfieldRefs[refOffset + 1]) + 1;
215         if (LKumodwidths[refOffset + 1] < 0)
216         {
217             tWidth2 = - (LKumodwidths[refOffset + 1]) + 1;
218             joinField = 1;
219             while ((LKumodwidths[refOffset + 1] < 0) &&
220                 (i < ((LKumodshowFields > 0) ? LKumodshowFields : MAXLKUFIELDS)))
221             {
222                 i++;
223             }
224         }
225         else
226         {
227             tWidth2 = LKumodwidths[refOffset + 1] + 1;
228             joinField = 0;
229         }
230
231         if (! LKumodfrceCols)
232         {
233             if (tWidth2 && joinField)
234             {
235                 currntPos += tWidth2;
236             }
237             else
238             {
239                 if (tWidth1 < tWidth2)
240                 {
241                     currntPos += tWidth1;
242                 }
243                 else
244                 {
245                     currntPos += tWidth2;
246                 }
247             }
248         }
249         else
250             currntPos += tWidth2;
251
252         tFields++;
253         i++;
254     }
255     return reachedEnd;
256 }
257
258 void LKumodDisplayRow(int row, long highLight)
259 {
260     int i = 0;
261     int colPos = 0, prevPos;
262     int nextWidth;
263     /* int tWidth ; */
264     long fieldRef;
265     /* int frceWidth ; */
266     int padLength;
267     int joinField;
268     int nextOffset;
269     int attrib, trimLen;
270     int firstCombo;
271     char outputText[CRTWIDTH];
272     char holdText[CRTWIDTH];
273     char joinString[CRTWIDTH];
274     char join2String[CRTWIDTH];
275     char fieldText[MAXLKUFIELDWIDTH];
276     firstCombo = 0;
277     nextOffset = 0;
278     joinField = 0;

```

```

262 strcpy(joinString, "\0");
263 strcpy(join2String, "\0");
264 while (1)
264 {
265     prevPos = colPos;
266
267     if ((LKUmodcurrentRef + i) >= LKUmodtotalFields)
268         break;
269
270     if (! nextOffset && LKUmodcurrentRef > 0)
271         if (LKUmodwidths[LKUmodcurrentRef + i - 1] < 0)
271         {
272             i++;
273             continue;
274         }
275
276     fieldRef = LKUmodfieldRefs[LKUmodcurrentRef + i];
277     nextWidth = LKUmodwidths[LKUmodcurrentRef + i];
278
279     /* check for field combinations */
280     if (nextWidth < 0)
280     {
281         joinField = 1;
282         nextWidth *= -1;
283         strcpy(join2String, "More");
284         firstCombo++;
285     }
286     else
286     {
287         strcpy(join2String, "\0");
288     }
289
290     if (colPos >= (LKUmodtotlWndwWidth - 1))
290         break;
291
292     if (! nextOffset)
293     /* w4num_att( row, colPos++, " ", 1, (highLight == 0) ?
294        LKUmodLookup.lField[ (LKUmodshowFields && i >= LKUmodshowFields)
295           ? LKUmodshowFields-1 : i ].attribute
296           : highLight ) ;
297     */
298        w4num_att(row, colPos++, " ", 1, (highLight == 0) ?
299           LKUmodLookup.lField[i ].attribute
300           : highLight);
301
302
303     if (LKUmodshowFields > 0 && i >= LKUmodshowFields)
304         if (! LKUmodallowExtra)
305             break;
306     if ((i + 1) > ((LKUmodtotalFields < MAXLKUFIELDS) ? LKUmodtotalFields : MAXLKUFIELDS))
307         break;
308
309     if (LKUmodallowExtra && i >= LKUmodshowFields)
310         LKUmodLookup.lField[i].attribute = LKUmodLookup.windowAttr;
311
312     if (f4type(fieldRef) == 'D')
313         if (f4str(fieldRef:[0] == ' ')
314             strcpy(fieldText, STRspaces(strlen(LKUmoddateFormat)));
315         else
316             strcpy(fieldText, c4dt_format(f4str(fieldRef), LKUmoddateFormat));
317     else
318         if (f4type(fieldRef) == 'M')
319             strcpy(fieldText, "MEMO");
320         else
321             strcpy(fieldText, f4str(fieldRef));
322
323
324     if (joinField)
324     {
325         if (join2String[0] != '\0')
325         {
326             strcpy(join2String, STRrtrim(fieldText));
327             strcat(join2String, LKUmodLookup.lField[LKUmodcurrentRef + i].joinChars);
328             strcat(joinString, join2String);

```

```

329         if (strlen(joinString) > CRTWIDTH)
330             strcpy(holdText, STRleft(joinString, CRTWIDTH - 1));
331         else
332             strcpy(holdText, joinString);
333         nextOffset = strlen(holdText);
334         i++;
335         continue;
336     }
337     strcpy(joinString, "\0");
338
339     if ((strlen(fieldText) + strlen(holdText)) < CRTWIDTH)
340         strcat(holdText, STRrtrim(fieldText));
341     else
342         strcat(holdText,
343             STRleft(STRrtrim(fieldText), ((CRTWIDTH - 1) - strlen(holdText))));
344
345     strcpy(fieldText, holdText);
346
347     nextWidth = - (LKmodwidths[i - firstCombo]);
348     if (nextWidth < 0) nextWidth *= -1;
349
350     firstCombo = 0;
351     joinField = 0;
352     nextOffset = 0;
353 }
354
355 strcpy(outputText, "\0");
356 if (nextWidth)
357 {
358     padLength = nextWidth - strlen(fieldText);
359     if ((padLength > 0) && LKUmodfrceCols)
360     {
361         if (padLength < CRTWIDTH)
362             strcpy(outputText, STRspaces(padLength));
363         else
364             strcpy(outputText, STRleft(STRspaces(padLength), CRTWIDTH - 1));
365     }
366     else
367     {
368         nextWidth = strlen(fieldText);
369         if ((strlen(outputText) + strlen(fieldText)) < CRTWIDTH)
370             strcat(outputText, fieldText);
371         else
372             strcat(outputText, STRleft(fieldText, CRTWIDTH - 1));
373     }
374     if ((int) strlen(outputText) < (int) nextWidth)
375         strcat(outputText, STRspaces(nextWidth - strlen(outputText)));
376     else
377         outputText[nextWidth] = '\0';
378
379     trimLen = strlen(outputText) - (LKUmodtotlWndwWidth - colPos);
380     if (trimLen > 0)
381         trimLen = strlen(outputText) - trimLen;
382
383     if (highLight == 0)
384         attrib = (int) LKUmodLookup.lField[i].attribute;
385     else
386         attrib = (int) highLight;
387     if (trimLen > 0)
388     {
389         if (LKUmodtruncOutput)
390             w4num_att(row, colPos, STRleft(outputText, trimLen),
391                 trimLen, (long) attrib);
392         else
393         {
394             w4num_att(row, colPos, STRleft(STRspaces(trimLen + 1), trimLen),
395                 trimLen, (long) attrib);
396         }
397     }
398     else
399         w4num_att(row, colPos, outputText, nextWidth, (long) attrib);
400
401     colPos += nextWidth;
402     i++;
403 }

```

```

400
401     if (prevPos < LKUmodtotlWndwWidth)
402         w4num_att(LOW, prevPos, STRspaces((LKUmodtotlWndwWidth - prevPos) + 1),
403             (LKUmodtotlWndwWidth - prevPos),
404             (highLight == 0) ? LKUmodLookup.windowAttr : highLight);
405
406 }
407
408 void LKUmodDisplayLookup(int row)
409 {
410     long cr = d4recnc ();
411     int i;
412
413     x4skip((long) - row);
414     for (i = 0; i < LKUmodLookup.height; i++, x4skip(1L))
415     {
416         LKUmodDisplayRow(i, (long) 0);
417     }
418     d4go(cr);
419 }
420 void LKUmodPositionLookup(long records)
421 {
422     x4skip(records);
423 }
424
425 long LKUmodlookup(void)
426 {
427     long d4bof_BUG_FIX;
428
429     int lookWindow;
430     int lookWidth;
431     int row;
432     int returnKey = 0;
433     long SelectRecord;
434     char tempString1[81];
435
436     /* Ensure that the LKUmodinit function was called first */
437     if (LKUmodfirstTime)
438     {
439         u4error(1200, "Uninitialized Structure", "",
440             "LKUmodinit() must be called prior to LKUmodlookup()", (char *) 0);
441         return 0;
442     }
443
444     /* Open / Select the file(s) to lookup */
445     if (LKUmodLookup.lfile.dbfSelect >= 0)
446     {
447         if (d4select(LKUmodLookup.lfile.dbfSelect) < 0)
448             return 0;
449         if (LKUmodLookup.lfile.ndxSelect >= 0)
450             if (i4select(LKUmodLookup.lfile.ndxSelect) < 0)
451                 return 0;
452     }
453     else
454     {
455         if (d4use(LKUmodLookup.lfile.fileName) < 0)
456         {
457             #ifndef NO_ERROR
458             /* Problem occurred when closing the answer database */
459             sprintf(tempString1, "Error when opening %s",
460                 LKUmodLookup.lfile.fileName);
461             u4error(857, tempString1, "", (char *) 0);
462             #endif
463             return 0;
464         }
465         if (LKUmodLookup.lfile.indxName[0] != '\0')
466             if (i4open(LKUmodLookup.lfile.indxName) < 0)
467                 return 0;
468     }

```

```

468
469     if (LKUmodLookup.filterRoutine != NULL)
470         x4filter(LKUmodLookup.filterRoutine);
471
472     x4top();
473     d4bof_BUG_FIX = d4recno();
474
475     LKUmodGetRefs();
476     lookWidth = LKUmodtotalWndwWidth - (LKUmodwindowWidth) ? LKUmodwindowWidth
477         : LKUmodLookWindowWidth();
478     lookWindow = w4define(LKUmodLookup.startRow, LKUmodLookup.startCol,
479         LKUmodLookup.startRow + LKUmodLookup.height + 1, LKUmodLookup.startCol + lookWidth
;
480     w4attribute(LKUmodLookup.windowAttr);
481     /*
482         if( LKUmodLookup.shadow )
483             w4shadow( lo kWindow, ON );
484     */
485     w4popup();
486     w4border(DOUBLE_TOP, LKUmodLookup.windowAttr);
487
488     /* Check the title strings' length... if greater than the window
489        then cut it */
490     strcpy(tempString1, LKUmodLookup.title);
491     if ((int) strlen(tempString1) > (int) lookWidth)
492         tempString1[lookWidth] = '\0';
493     w4title((int) - 1, (int) - 1, (char *) tempString1, (long) LKUmodLookup.windowAttr);
494
495     w4activate(lookWindow);
496     w4clear(0);
497     w4cursor(-1, -1);
498
499
500     row = 0;
501     LKUmodDisplayLookup(row);
502
503     while ((returnKey != RETURN) && (returnKey != ESC) && (returnKey != F3))
504     {
505         LKUmodDisplayRow(row, LKUmodLookup.highLightBar);
506         returnKey = g4char();
507         LKUmodDisplayRow(row, (long) 0);
508         switch (returnKey)
509         {
510             case ESC : SelectRecord = -1L;
511                 break;
512             case F3 : SelectRecord = -2L;
513                 break;
514             case RETURN : SelectRecord = d4recno();
515                 break;
516             case F2 :
517                 /* get matid number to seek on */
518                 /* maybe incorporate string seek on matdesc */
519                 break;
520             case RIGHT :
521                 if (LKUmodscrollFields)
522                     if (! LKUmodall_fields(LKUmodcurrentRef))
523                     {
524                         while (1)
525                         {
526                             LKUmodcurrentRef++;
527                             if (LKUmodwidths[LKUmodcurrentRef - 1] >= 0 ||
528                                 LKUmodcurrentRef >= LKUmodtotalFields)
529                                 break;
530                         }
531                         LKUmodDisplayLookup(row);
532                     }
533                 break;
534             case LEFT :
535                 if (LKUmodscrollFields)
536                     if (LKUmodcurrentRef)
537                     {
538                         while (1)

```

```

535         {
536             LKUmodcurrentRef--;
537             if (LKUmodwidths[LKUmodcurrentRef - 1] >= 0 ||
538                 LKUmodcurrentRef == 0)
539                 break;
540         }
541         LKUmodDisplayLookup(row);
542     }
543     break;
544
545     case UP : if (d4recno() != d4bof_BUG_FIX)
546     {
547         LKUmodPositionLookup(-1L);
548         if (row > 0)
549             row--;
550         else
551             if (! d4bof())
552                 w4scroll(-1);
553     }
554     break;
555
556     case DOWN : LKUmodPositionLookup(1L);
557     if (++row == LKUmodLookup.height || d4eof())
558     {
559         if (! d4eof())
560         {
561             w4scroll(1);
562         }
563         else
564             x4bottom();
565         row--;
566     }
567     break;
568
569     case PGDN : LKUmodPositionLookup((long) (LKUmodLookup.height - 1));
570     if (d4eof())
571     {
572         LKUmodPositionLookup((long) (- LKUmodLookup.height));
573         if (d4bof())
574             x4top();
575     }
576     row = 0;
577     LKUmodDisplayLookup(row);
578     row = 0;
579     break;
580
581     case PGUP : LKUmodPositionLookup((long) (~ LKUmodLookup.height));
582     if (d4bof())
583         x4top();
584     row = 0;
585     LKUmodDisplayLookup(row);
586     row = 0;
587     break;
588
589     case HOME : x4top();
590     row = 0;
591     LKUmodDisplayLookup(row);
592     row = 0;
593     break;
594
595     case END : x4bottom();
596     x4skip((long) - (LKUmodLookup.height - 1));
597     row = 0;
598     LKUmodDisplayLookup(row);
599     row = 0;
600     break;
601 }
602
603 w4deactivate(lookWindow);
604 w4close(lookWindow);
605
606 if (LKUmodLookup.lfile.dbfSelect < 0)

```

```

604         if (d4close())
605         {
606             #ifndef NO_ERROR
607             /* Problem occurred when closing the query database */
608             sprintf(tempString1, "Error when closing %s",
609                     LKUModLookup.lfile.fileName );
610             u4error(858, tempString1, "", (char *)0 );
611             #endif
612         }
613     return SelectRecord;
614 }
615
616
617 void LKUmoddate_format(char * date_format)
618 {
619     if (strlen(date_format) < 30)
620         strcpy(LKUmoddateFormat, date_format);
621 }
622
623 void LKUmodfield(char * fName, long attr)
624 {
625     int fieldLength = strlen (fName);
626     char tempString1[81];
627     char tempString2[81];
628
629     if (fieldLength > 21 && fName != (char *) 0)
630     {
631         #ifndef NO_ERROR
632         u4error(1210, "Field name too long", "", fName, "",
633             "Must be 10 characters of less", "", tempString1, (char *)0 );
634         #endif
635     }
636     else
637     {
638         strcpy(LKUmodLookup.lField[LKUmodshowFields].fieldName, fName);
639         if (attr >= 0L && attr <= 255L)
640             LKUmodLookup.lField[LKUmodshowFields].attribute = attr;
641
642         #ifndef NO_ERROR
643         else {
644             sprintf(tempString1, "%d", (int)attr) ;
645             sprintf(tempString2, "Using the default field attribute value of %d",
646                 (int)DEFAULT_FATTR) ;
647             u4error(1220, "Invalid attribute value", "", tempString1, "",
648                 "Attribute must be between 0 and 255", "", tempString2, (char *)0 );
649         }
650         #endif
651
652         if (fName != (char *) 0)
653             LKUmodshowFields++;
654     }
655 }
656
657 void LKUmodinit(void)
658 {
659     strcpy(LKUmodLookup.title, DEFAULT_TITLE);
660     LKUmodLookup.windowAttr = (long) DEFAULT_WATTR;
661     LKUmodLookup.startRow = (char) STARTROW;
662     LKUmodLookup.startCol = (char) STARTCOL;
663     LKUmodLookup.height = (char) DEFAULT_HEIGHT;
664     LKUmodLookup.highLightBar = (long) DEFAULT_HATTR;
665     strcpy(LKUmodLookup.lfile.fileName, "\0");
666     strcpy(LKUmodLookup.lfile.indxName, "\0");
667     LKUmodLookup.lfile.dbfSelect = -1;
668     LKUmodLookup.lfile.ndxSelect = -1;
669     LKUmodLookup.filterRoutine = NULL;
670     LKUmoddate_format("MM/DD/YY");
671     LKUmodfrceCols = 0;
672     LKUmodwindowWidth = 0;
673     LKUmodallowExtra = 1;
674     LKUmodLookup.shadow = 0;
675     LKUmodset_columns(0);
676     LKUmodscrollFields = 1;

```

```

674     LKUModtruncOutput = 1;
675
676     LKUModcurrentRef = LKUModshowFields = LKUModfirstTime = 0;
677 /*
678     if( ! copyright )
679         COPYRIGHT_NOTICE() ;
680 */
681 }
682
683 void LKUModtitle(char * title)
684 {
685     int titleLength = strlen (title);
686     char tempString1[81];
687     char tempString2[81];
688
689     if (titleLength > TITLELENGTH)
690     {
691         #ifndef NO_ERROR
692             sprintf(tempString1,"Must be %d characters of less",TITLELENGTH) ;
693             sprintf(tempString2,"Lookup title will contain %s",DEFAULT_TITLE) ;
694             u4error(1230,"Title too long", "", title, "", tempString1, "",
695                 tempString2, (char *)0 ) ;
696         #endif
697     }
698     else
699         strcpy(LKUModLookup.title, title);
700 }
701 void LKUModposition(int startRow, int startCol, int windowHeight)
702 {
703     char tempString0[81];
704     char tempString1[81];
705     char tempString2[81];
706
707     if (startRow != -1)
708     {
709         if (startRow >= 0 && startRow <= MAXROWS)
710             LKUModLookup.startRow = (char) startRow;
711         else
712         {
713             #ifndef NO_ERROR
714                 sprintf(tempString1,"Must be between 0 and %d",MAXROWS) ;
715                 sprintf(tempString2,"Using default value of %d rows",
716                     (int)DEFAULT_ROWS) ;
717                 u4error(1240,"Row position outside screen limits", "",
718                     tempString1, "", tempString2, (char *)0 ) ;
719             #endif
720         }
721     }
722     if (startCol != -1)
723     {
724         if (startCol >= 0 && startCol <= MAXCOLS)
725             LKUModLookup.startCol = (char) startCol;
726         else
727         {
728             #ifndef NO_ERROR
729                 sprintf(tempString1,"Must be between 0 and %d",MAXCOLS) ;
730                 sprintf(tempString2,"Using default value of %d columns",
731                     (int)DEFAULT_COLS) ;
732                 u4error(1242,"Column position outside screen limits", "",
733                     tempString1, "", tempString2, (char *)0 ) ;
734             #endif
735         }
736     }
737     if (windowHeight != -1)
738     {
739         if (windowHeight >= 1 && windowHeight <= (MAXROWS - startRow))
740             LKUModLookup.height = (char) windowHeight;
741         else
742         {
743             #ifndef NO_ERROR
744                 sprintf(tempString0,"Height of %d will not fit in the window",
745                     windowHeight) ;
746                 sprintf(tempString1,"Must be between 1 and %d",

```

```

740         (int) (MAXROWS-startRow)) ;
741         sprintf(tempString2,"Using default value of %d rows",
742         (int)DEFAULT_HEIGHT) ;
743         u4error(1244,tempString0, "", tempString1, "", tempString2,
744         (char *)0 ) ;
745     #endif
746 }
747 }
748 }
749
750 void LKUmodselect(int dbfRef, int ndxRef)
751 {
752     char tempString1[81];
753
754     if (dbfRef != -1)
755     {
756         if (d4select(dbfRef) < 0)
757         {
758             #ifndef NO_ERROR
759             sprintf(tempString1,
760             "Database base reference %d not available.",dbfRef) ;
761             u4error(1250,"Invalid Base Reference", "", tempString1,
762             "", "Exiting lookup function", (char *)0 ) ;
763             #endif
764         }
765         else
766             LKUmodLookup.lfile.dbfSelect = dbfRef;
767     }
768     else
769     {
770         #ifndef NO_ERROR
771         sprintf(tempString1,"No database base reference given.",dbfRef) ;
772         u4error(1251,"Invalid Base Reference", "",
773         "No database base reference given.", (char *)0 ) ;
774         #endif
775     }
776
777     if (ndxRef != -1)
778     {
779         if (i4select(ndxRef) < 0)
780         {
781             #ifndef NO_ERROR
782             sprintf(tempString1,"Index reference %d not available.",ndxRef) ;
783             u4error(1252,"Invalid Index Reference", "", tempString1,
784             "", "Exiting lookup function", (char *)0 ) ;
785             #endif
786         }
787         else
788             LKUmodLookup.lfile.ndxSelect = ndxRef;
789     }
790     else
791         LKUmodLookup.lfile.ndxSelect = ndxRef;
792 }
793
794 void LKUmodfile(char * dbfName, char * ndxName)
795 {
796     char fileMatch[14];
797     char tempString1[81];
798
799     if (dbfName[0] != '\0')
800     {
801         if (u4file_first(dbfName, fileMatch))
802         {
803             #ifndef NO_ERROR
804             sprintf(tempString1,"Invalid %s database path or file name.",
805             dbfName) ;
806             u4error(1254,tempString1, "", "Exiting lookup function",
807             (char *)0 ) ;
808             #endif
809         }
810         else
811         {
812             strcpy(LKUmodLookup.lfile.fileName, dbfName);
813         }
814     }
815 }

```

```

801     }
801     else
801     {
802         #ifndef NO_ERROR
803             sprintf(tempString1,"No database path or file name given.",
804                     dbfName) ;
805             u4error(1255,tempString1, "", "Exiting lookup function",
806                     (char *)C ) ;
807         #endif
808     }
809
810     if (ndxName[0] != '\0')
810     {
811         if (u4file_first(ndxName, fileMatch))
811         {
812             #ifndef NO_ERROR
813                 sprintf(tempString1,"Invalid %s index path or file name.",
814                         ndxName) ;
815                 u4error(1256,tempString1, "", "Exiting lookup function",
816                         (char *)0 ) ;
817             #endif
818         }
818         else
818         {
819             strcpy(LKUmodLookup.lfile.indexName, ndxName);
820         }
821     }
822 }
823
824 void LKUmodattributes(long windowAttrib, long highlightBar)
825 {
826     char tempString1[81];
827     char tempString2[81];
828
829     if (windowAttrib != -1)
829     {
830         if (windowAttrib >= 0L && windowAttrib <= 255L)
831             LKUmodLookup.windowAttr = windowAttrib;
832
833         #ifndef NO_ERROR
834         else {
835             sprintf(tempString1,"%d", (int)windowAttrib) ;
836             sprintf(tempString2,
837                     "Using the default field attribute value of %d",
838                     (int)DEFAULT_FATTR) ;
839             u4error(1220,"Invalid attribute value", "", tempString1, "",
840                     "Attribute must be between 0 and 255", "", tempString2,
841                     (char *)0 ) ;
842         }
843         #endif
844     }
845     if (highlightBar != -1)
845     {
846         if (highlightBar >= 0L && highlightBar <= 255L)
847             LKUmodLookup.highLightBar = highlightBar;
848
849         #ifndef NO_ERROR
850         else {
851             sprintf(tempString1,"%d", (int)highlightBar) ;
852             sprintf(tempString2,"Using the default field attribute value of %d",
853                     (int)DEFAULT_FATTR) ;
854             u4error(1220,"Invalid attribute value", "", tempString1, "",
855                     "Attribute must be between 0 and 255", "", tempString2,
856                     (char *)0 ) ;
857         }
858         #endif
859     }
860 }
861
862 void LKUmodfilter(int (* fltrFunction) (void))
863 {
864     LKUmodLookup.filterRoutine = fltrFunction;
865 }
866

```

```

867
868 int LKUmodcolumn_width(int columnNum, int width)
869 {
870     char tempString0[81];
871     char tempString1[81];
872
873     if (columnNum >= TOTALFIELDS || columnNum < 0)
874     {
875         #ifndef NO_ERROR
876             sprintf(tempString0, "Invalid column number %d", (int)columnNum) ;
877             sprintf(tempString1, "Column number must be between 0 and %d",
878                 (int)TOTALFIELDS) ;
879             u4error(1260, tempString0, "", tempString1, "",
880                 "Using the field width as the default", (char *)0) ;
881         #endif
882         return - 1;
883     }
884     if (width != -1)
885     {
886         if (width <= 0 || width > CRTWIDTH)
887         {
888             #ifndef NO_ERROR
889                 sprintf(tempString0, "Invalid column width: %d", (int)width) ;
890                 sprintf(tempString1, "Column width must be between 1 and %d",
891                     (int)CRTWIDTH) ;
892                 u4error(1262, tempString0, "", tempString1, "",
893                     "Using the field width as the default", (char *)0) ;
894             #endif
895             return - 1;
896         }
897         if (LKUmodwidths[columnNum] < 0)
898             LKUmodwidths[columnNum] = - width;
899         else
900             LKUmodwidths[columnNum] = width;
901     }
902     if (LKUmodwidths[columnNum] < 0)
903         return - (LKUmodwidths[columnNum]);
904     else
905         return LKUmodwidths[columnNum];
906 }
907
908 int LKUmodforce_columns(int fcSwitch)
909 {
910     if (fcSwitch != -1)
911         LKUmodfrceCols = (fcSwitch) ? 1 : 0;
912     return LKUmodfrceCols;
913 }
914
915 int LKUmodallow_extra(int aeSwitch)
916 {
917     if (aeSwitch != -1)
918         LKUmodallowExtra = (aeSwitch) ? 1 : 0;
919     return LKUmodallowExtra;
920 }
921
922 int LKUmodwindow_width(int newWidth)
923 {
924     if (newWidth != -1)
925         if (newWidth > 0)
926             LKUmodwindowWidth = (newWidth <= CRTWIDTH) ? newWidth : 0;
927     return LKUmodwindowWidth;
928 }
929
930
931 int LKUmodscrolling(int scrollSwitch)
932 {
933     if (scrollSwitch != -1)
934         LKUmodscrollFields = (scrollSwitch) ? 1 : 0;
935     return LKUmodscrollFields;
936 }
937

```

```
938
939 int LKUmodtruncate(int truncSwitch)
940 {
941     if (truncSwitch != -1)
942         LKUmodtruncOutput = (truncSwitch) ? 1 : 0;
943     return LKUmodtruncOutput;
944 }
945
946 void LKUmodcombine_fields(char * joinchars)
947 {
948     if (LKUmodshowFields > 0 && LKUmodshowFields < MAXLKUFIELDS)
949     {
950         if (strlen(joinchars) > MAXJOINCHARS)
951             return;
952         if (LKUmodwidths[LKUmodshowFields - 1] == 0)
953             LKUmodwidths[LKUmodshowFields - 1] = -2;
954         else
955             LKUmodwidths[LKUmodshowFields - 1] *= -1;
956         strcpy(LKUmodLookup.lField[LKUmodshowFields - 1].joinChars, joinchars);
957     }
958 }
959
960 int LKUmodshadow(int ofSwitch)
961 {
962     if (ofSwitch != -1)
963         LKUmodLookup.shadow = (ofSwitch) ? 1 : 0;
964     return LKUmodLookup.shadow;
965 }
966 }
```

```
1 /*
2  * main.c
3  *
4  */
5
6 #include "dwmenu.h"
7 #include "rmrsinit.h"
8 #include "d4base.h"
9 #include "dw.h"
10
11 #define MAINMODULE
12 #include "rmrs.h"
13
14 void main(int argc, char * argv[])
15 {
16     if (argc > 1) setdwlib(-1, 7, -1, -1);
17     else setdwlib(-1, 3, -1, -1);
18     d4initialize(10, 15, 80, 3000, 0xFC00L);
19     IntroScreen();
20     SetupMenu();
21     while (1)
22     {
23         MNUDoSelect(mmain, NULLF);
24     }
25 }
```

```
1 /*
2  * miscutil.c
3  *
4  *      miscellaneous utility functions
5  */
6
7 #include <string.h>
8 #include <stdio.h>
9 #include <conio.h>
10 #include <ctype.h>
11 #include "dw.h"
12
13 void trimstr(char * s)
14 {
15     /*
16      * ARGUMENT
17      *      (char *) s - input string
18      *
19      * DESCRIPTION
20      *      Trim string of leading and trailing blanks
21      *
22      * RETURNS
23      *
24      */
25     int len, cnt = 0, i;
26
27     /* remove white space at beginning of string */
28     for (cnt = 0; isspace(*(s + cnt)); cnt++);
29     len = strlen(s);
30     for (i = 0; i <= (len - cnt); i++)
31         *(s + i) = *(s + i + cnt);
32     *(s + len + 1) = '\0';
33
34     /* remove white space at end of string */
35     len = strlen(s);
36     len--;
37     while (isspace(*(s + len)))
38         len--;
39     *(s + len + 1) = '\0';
40 }
41
42 void dmpxtrakeys(void)
43 {
44     /*
45      * ARGUMENT
46      *
47      * DESCRIPTION
48      *      Dumps extra keystrokes from input buffer
49      *
50      * RETURNS
51      *
52      */
53     while (kbhit()) getch();
54 }
55
56 char * uppercase(char * str)
57 {
58     /*
59      * ARGUMENT
60      *      (char *) str - input string
61      *
62      * DESCRIPTION
63      *      Converts all lowercase characters of string to uppercase
64      *
65      * RETURNS
66      *      (char *) converted string
67      *
68      */
69     int i, len;
70
71     len = strlen(str);
```

```
72     for (i = 0; i <= (len - 1); i++)
73         str[i] = toupper(str[i]);
74     return (str);
75 }
```

```

1  /*
2  *   procauto.c
3  *
4  *       contains DoProcessAuto()
5  *
6  *       by Chris Sawyer (last rev: 1/4/91)
7  */
8
9  /*
10 *       Rev 1.0 ?? Jan 1992
11 *       Original release
12 *
13 *
14 */
15
16 #include <stdio.h>
17 #include <conio.h>
18 #include <stdlib.h>
19 #include <dir.h>
20 #include <string.h>
21 #include <dw.h>
22 #include <dwmenu.h>
23 #include <dwsystem.h>
24 #include <color.h>
25 #include <d4base.h>
26
27 // custom header files
28 #include "sysdefs.h" // general typedefs and defs for procauto.c & comprom.c
29 #include "dbutil.h" // database access fns (only used by procauto.c & comprom.c presently)
30 #include "miscutil.h" // miscellaneous fns (only used by procauto.c & comprom.c presently)
31 #include "procutil.h"
32 #include "rmrsutil.h"
33 int DoCompromise(long, REQLSTNODE *);
34
35 #define RMRSError -99
36 #define NOFILES -1
37 #define FOUND 0
38 #define NOTFOUND 1
39
40
41 // Assume all databases exist
42 // Handle for empty case
43
44 // SETUP RMRS SYSTEM DIRECTORY VARIABLES
45 // TO BE DEFINED IN "INIT/SETUP" FUNCTION
46 #ifdef RMRSSYSTEMDIRS
47 extern char pdesinputdirpath;
48 extern char pdesarchvdirpath;
49 #else
50     char pdesinputdirpath[50] = "";
51     char pdesarchvdirpath[50] = "archive\\";
52 #endif
53
54 extern HWND win; // Process | Auto mode comand window handle
55
56 int LoadSingFacFiles(char *);
57 long GetFacnum(char *);
58 long GetNxtAvlRepid(void);
59 int LoadEqpReqs(char *, long, long, int);
60 int LoadMatReqs(char *, long);
61 int LoadRepInfo(char *, long);
62 int TryResAlloc(void);
63 int AddRepRec(long, long, int);
64 int AddEqpRec(EQPREQREC);
65 int AddMatRec(char *, float, char *, long, char);
66 int AddRepInfoRec(REPINFOREC *);
67 void ArchiveFile(char *, char *, long);
68 long HrToSec(float);
69 long ChkOvrddPoss(long, float, long);
70 int AnyConflicts(REQLSTNODE *);
71 int SkipLines(FILE *, int, char *);
72 void ParseLine(char *, char *);
73 void PrintChkLst(REQLSTNODE *);
74 void ClearChkLst(REQLSTNODE *);

```

```

75 int AllocMatls(REQLSTNODE *, long);
76 int setrepairstatus(int, long);
77 int GetRepairPrty(long);
78 void cleanwin(void);
79
80 #ifdef RMRSPROCSTATS
81 // processing info variable declarations
82 typedef struct faclistnode {
83     char *facnumstr[10]; // facility number string (e.g. B4058)
84     int numofreps; // number of repairs in that submission
85     struct faclistnode *next;
86 } FACLISTNODE;
87 int numofrepsproc; // # of repairs processed
88 int numofconflreps; // # of repairs w/ conflicts
89 FACLISTNODE *faclist; // LL of facilities processed
90 #endif
91
92 // local statistic var
93 static int fileloadcnt;
94
95 /*
96  * int DoProcessAuto()
97  *
98  * ARGUMENTS: none
99  *
100 * DESCRIPTION
101 * Loads ALL new PDES facility repair files and checks for resource
102 * availability conflicts, allocating material resources to a repair
103 * if no conflicts arise or upon succesful resolution of those conflicts
104 * Repairs go to one of three states: (P)ossible
105 *                                     (S)uspended
106 *                                     (X)anceled
107 *
108 * RETURNS
109 *
110 * AUTHOR
111 * Applied Research Associates, Inc.
112 *
113 * MODIFICATIONS
114 *
115 */
116
117 int DoProcessAuto(void)
118 {
119     struct ffbk fblkptr;
120     int done;
121     char facnumstr[10];
122     char pdesinputfilestr[65];
123
124     // compose custom colors
125 {
126     int attr;
127     // colors used in COMPROMISE ENVIRONMENT
128     attr = bldattr(LTGREY, RED);
129     tblattr(SELBLINK, attr | GINTENSITY | GBLNK);
130     attr = bldattr(LTGREY, RED);
131     tblattr(SELNORM, attr | GINTENSITY);
132     attr = bldattr(BLACK, LTGREY);
133     tblattr(SELQTYBLINK, attr | GBLINK);
134     attr = bldattr(WHITE, BLUE);
135     tblattr(HILITE, attr);
136
137     // colors used in miscellaneous MESSAGE BOXES
138     attr = bldattr(WHITE, LTGREY);
139     tblattr(GREYBOX, attr | GINTENSITY);
140     attr = bldattr(BLACK, LTGREY);
141     tblattr(GRYBTXT, attr);
142     tblattr(GRYBTXTBLINK, attr | GBLINK);
143     attr = bldattr(CYAN, BLUE);
144     tblattr(KEYALERT, attr | GBLINK);
145     attr = bldattr(CYAN, BLUE);
146     tblattr(KEYMESS, attr);
147     attr = bldattr(BLACK, GREEN);

```

```

148     tblattr(FLSHCHR, attr | GBLINK);
149     )
150     // open required databases (old method)
151     repair_dbf = d4use_excl("REPAIR.DBF");
152     matreq_dbf = d4use_excl("MATREQ.DBF");
153     eqpreq_dbf = d4use_excl("EQPREQ.DBF");
154     eqpsup_dbf = d4use_excl("EQPSUP.DBF");
155     matsup_dbf = d4use_excl("MATSUP.DBF");
156     repinfo_dbf = d4use_excl("REPINFO.DBF");
157
158     fileloadcnt = 0; // init file load count
159     // RETRIEVE NEW PDES FILES FROM INPUT DIRECTORY
160     // Since files are removed from this directory after processing,
161     // any .EQP, .MAT, .OUT files found there are considered to be new.
162     strcpy(pdesinputfilestr, pdesinputdirpath);
163     strcat(pdesinputfilestr, "*.EQP");
164     done = findfirst(pdesinputfilestr, & fblkptr, 0);
165     if (done == NOFILES)
166     {
167         vatputs(win, 1, 2, "WARNING: NO FILES FOUND IN ");
168         vatputs(win, 2, 2, "SPECIFIED DIRECTORY ");
169         vatputs(win, 4, 2, "*** Check PDES Input directory setting ");
170         vatputs(win, 6, 5, "-> press any key to continue ");
171         getkey();
172         return (0);
173     }
174     // Load facility repair file sets one at a time
175     while (done != NOFILES)
176     {
177         strcpy(facnumstr, fblkptr.ff_name);
178         facnumstr[strlen(facnumstr) - 4] = '\0';
179         vatprintf(win, 2, 2, "Loading facility files for: %5s ", facnumstr);
180         modattr(win, 2, 30, 7, HELP);
181
182         LoadSingFacFiles(facnumstr);
183
184         done = findnext(& fblkptr);
185     }
186     d4close_all(); //
187     // reindex files that changed
188     open_matreq(1); // 3 files open
189     open_eqpreq(1); // 6 files open
190     open_repair(1); // 3 files open
191     open_repinfo(1); // 2 files open
192     d4close_all(); //
193     // reopen files to use for resource allocation
194     repair_dbf = d4use_excl("REPAIR.DBF");
195     matreq_dbf = d4use_excl("MATREQ.DBF");
196     eqpreq_dbf = d4use_excl("EQPREQ.DBF");
197     eqpsup_dbf = d4use_excl("EQPSUP.DBF");
198     matsup_dbf = d4use_excl("MATSUP.DBF");
199
200     cleanwin();
201     TryResAlloc();
202
203     d4close_all();
204     open_matreq(1); // 3 files open
205     open_eqpreq(1); // 6 files open
206     open_repair(1); // 3 files open
207     d4close_all();
208     return (1);
209 }
210 int LoadSingFacFiles(char * facnumstrptr)
211 /*
212  * ARGUMENT
213  * (char *) facnumstrptr - pointer to facility number string (e.g. "B4058")
214  *
215  * DESCRIPTION
216  * Loads the repair-specific databases (REPAIR.DBF, EQPREQ.DBF, MATREQ.DBF,
217  * and REPINFO.DBF) with the repair data from a single facility repair file
218  * set (i.e. the .EQP, .MAT, and .OUT files) specified by 'facnumstrptr'
219  */

```

```

220 * RETURNS
221 *   (int) status code: SUCCESS or FAILURE
222 *
223 * DATABASES AFFECTED (whether direct or indirect)
224 *   facprt_dbf (facprt_ndx1 - FACNUM index) - d4seek used
225 *   repair_dbf (no index used)
226 *   eqpreq_dbf (no index used)
227 *   matreq_dbf (no index used)
228 *   repinfo_dbf (no index used)
229 *
230 */
231 {
232     long facnum = 0L, indxrepid = 0L, currrepid = 0L;
233     int priority = 0, err = 0;
234
235     currrepid = GetNxtAvlRepId();      // get next avail repair id
236     indxrepid = currrepid;
237     facnum = GetFacnum(facnumstrptr);
238     // look up facility priority in FACPTY.DBF
239     if (! (priority = GetFacPty(facnum)))
240     {
241         u4error(-1, "WARNING: Facility not listed in",
242             "FACPTY.DBF",
243             ".. processing will be aborted",
244             "for this facility", (char *) 0);
245         return (FAILURE);              // abort
246     }
247     err = LoadEqpReqs(facnumstrptr, indxrepid, facnum, priority);
248     if (! err) err = LoadMatReqs(facnumstrptr, indxrepid);
249     if (! err) err = LoadRepInfo(facnumstrptr, indxrepid);
250     return (err);
251 }
252 int LoadEqpReqs(char * facnumstrptr, long indxrepid, long facnum, int priority)
253 {
254     /*
255     * ARGUMENT
256     *   (char *) facnumstrptr - pointer to facility number string (e.g. "B4058")
257     *   (long)   indxrepid    - index repid
258     *   (long)   facnum       - facility number
259     *   (int)    priority     - AB mission-critical facility priority assignment
260     *
261     * DESCRIPTION
262     *   Loads the repair equipment requirements into EQPREQ.DBF from the .EQP
263     *   file specified by 'facnumstrptr'; also, because each repair is
264     *   guaranteed to have at least 1 equipment requirement, namely a "repair
265     *   team", this fact is exploited within this function to produce a repair
266     *   data record (REPAIR.DBF) for each repair
267     *
268     * RETURNS
269     *   (int) error status code:  < 0 - error
270     *                               = 0 - success
271     *
272     * DATABASES AFFECTED (whether direct or indirect)
273     *   eqpreq_dbf (no index used)
274     */
275     char eqpfilename[50];
276     FILE * eqpfileptr;
277     long currrepid;
278     char prev[6], * currptr;
279     char fileline[80];
280     int eqpqty;
281     int err;
282     EQPREQREC eqprecbuf;
283
284     // load equipment requirements from .EQP file into EQPREQ.DBF *
285     strcpy(eqpfilename, pdesinputdirpath);
286     strcat(eqpfilename, facnumstrptr);
287     strcat(eqpfilename, ".EQP");
288     if ((eqpfileptr = fopen(eqpfilename, "r")) != NULL)
289     {
290         currrepid = indxrepid;
291         AddRepRec(facnum, currrepid, priority);

```

```

291     strcpy(prev, "1");
292     fgets(fileline, 80, eqpfileptr);
293     strtok(fileline, "\\");
294     currptr = strtok(NULL, "\\");
295
296     while (!feof(eqpfileptr))
297     {
298         // check to see this is eqp for diff repair
299         // if yes, create new repair record with unique id #
300         if (atoi(currptr) > atoi(prev))
301         {
302             strcpy(prev, currptr);
303             currrepid = indxrepid + (atoi(currptr) - 1);
304             err = AddRepRec(facnum, currrepid, priority);
305         }
306
307         // initialize eqp record structure
308         eqprecbuf.eqpid = 0L;
309         strcpy(eqprecbuf.eqpdscstr, "");
310         eqprecbuf.start = 0L;
311         eqprecbuf.duration = 0L;
312         eqprecbuf.repid = currrepid;
313         eqprecbuf.status = 'N';
314
315         // parse equipment record for information
316         strtok(NULL, "\\");
317         strcpy(eqprecbuf.eqpdscstr, strtok(NULL, "\\"));
318         strtok(NULL, "\\");
319         eqpqty = atoi(strtok(NULL, "\\"));
320         strtok(NULL, "\\"); strtok(NULL, "\\"); strtok(NULL, "\\");
321         eqprecbuf.duration = HrToSec(atoi(strtok(NULL, "\\")));
322
323         // add separate equipment requirement record
324         // for each individual piece of equipment
325         for (; eqpqty; eqpqty--)
326         {
327             err = AddEqpRec(eqprecbuf);
328             SkipLines(eqpfileptr, 1, fileline);
329             strtok(fileline, "\\");
330             currptr = strtok(NULL, "\\");
331         }
332         fclose(eqpfileptr);
333         ArchiveFile(eqpfilename, ".EQP", indxrepid);
334         return (RMRSSUCCESS);
335     }
336     else
337     {
338         u4error(-1, "FILE ACCESS RMRSError!", "can't .EQP open file:", eqpfilename, (char *) 0)
339
340         return (-1);
341     }
342 }
343
344 int LoadMatReqs(char * facnumstrptr, long indxrepid)
345 {
346     /*
347     * ARGUMENT
348     * (char *) facnumstrptr - pointer to facility number string (e.g. "B4058")
349     * (long) indxrepid - index repid
350     *
351     * DESCRIPTION
352     * Loads the repair material requirements into MATREQ.DBF from the .MAT
353     * file specified by 'facnumstrptr'
354     *
355     * RETURNS
356     * (int) error status code: < 0 - error
357     *                          = 0 - success
358     *
359     * DATABASES AFFECTED (whether direct or indirect)
360     * matreq_dbf (no index used)
361     */
362     char matfilename[50];

```

```

360 FILE * matfileptr;
361 long currrepid;
362 char * currptr;
363 char fileline[80];
364 char * matdescptr, * matunitptr;
365 float matqty;
366 int err;
367
368 // load material requirements from .MAT file into MATREQ.DBF **
369 strcpy(matfilename, pdesinputdirpath);
370 strcat(matfilename, facnumstrptr);
371 strcat(matfilename, ".MAT");
372 if ((matfileptr = fopen(matfilename, "r")) != NULL)
373 {
374     fgets(fileline, 80, matfileptr);
375     strtok(fileline, "\\");
376     currptr = strtok(NULL, "\\");
377     currrepid = indxrepid + (atoi(currptr) - 1);
378     while (!feof(matfileptr))
379     {
380         strtok(NULL, "\\");
381         matdescptr = strtok(NULL, "\\");
382         strtok(NULL, "\\");
383         matqty = atof(strtok(NULL, "\\"));
384         strtok(NULL, "\\");
385         matunitptr = strtok(NULL, "\\");
386
387         err = AddMatRec(matdescptr, matqty, matunitptr, currrepid, 'N');
388
389         fgets(fileline, 80, matfileptr);
390         strtok(fileline, "\\");
391         currptr = strtok(NULL, "\\");
392         currrepid = indxrepid + (atoi(currptr) - 1);
393     }
394     fclose(matfileptr);
395     ArchiveFile(matfilename, ".MAT", indxrepid);
396     return (RMRSSUCCESS);
397 }
398 else
399 {
400     u4error(-2, "FILE ACCESS RMRSError!",
401            "can't open .MAT file:", matfilename, (char *) 0);
402     return (-2);
403 }
404
405
406 int LoadRepInfo(char * facnumstrptr, long indxrepid)
407 {
408     /*
409     * ARGUMENT
410     * (char *) facnumstrptr - pointer to facility number string (e.g. "B4058")
411     * (long) indxrepid - indexrepid
412     * DESCRIPTION
413     * Loads the repair information into REPINFO.DBF from the .OUT
414     * file specified by 'facnumstrptr'
415     * RETURNS
416     * (int) error status code: < 0 - error
417     *                          = 0 - success
418     * DATABASES AFFECTED (whether direct or indirect)
419     * repinfo_dbf (no index used)
420     */
421
422     REPINFOREC buffer;
423     char outfilename[50];
424     FILE * outfileptr;
425     char linestr[80];
426     char elenumstr[50];

```

```

430     char remstr[80];
431     int cnt = 0, err;
432
433     // load facility repair info from .OUT file into REPINFO.DBF **
434     strcpy(outfilename, pdesinputdirpath); // build
435     strcat(outfilename, facnumstrptr); // filename
436     strcat(outfilename, ".OUT"); // string
437     if ((outfileptr = fopen(outfilename, "r")) != NULL)
438     {
439         // get facility header information
440         SkipLines(outfileptr, 1, linestr);
441         while (linestr[0] != '\n')
442             SkipLines(outfileptr, 1, linestr);
443         ParseLine(linestr, buffer.facfunc);
444         SkipLines(outfileptr, 2, linestr);
445         ParseLine(linestr, buffer.facdesc);
446         SkipLines(outfileptr, 8, linestr);
447         // loop through each repair report
448         while (!feof(outfileptr))
449         {
450             buffer.repid = indxrepid + cnt++;
451             ParseLine(linestr, elenumstr);
452             buffer.elenum = atoi(elenumstr);
453             SkipLines(outfileptr, 1, linestr);
454             ParseLine(linestr, buffer.eledesc);
455             SkipLines(outfileptr, 1, linestr);
456             ParseLine(linestr, buffer.dammode);
457             SkipLines(outfileptr, 1, linestr);
458             strcpy(buffer.damw, "x");
459             strcpy(buffer.daml, "x");
460             strcpy(buffer.damh, "x");
461             // compensate for TAB
462             while ((linestr[9] == 'D') || (linestr[2] == 'D'))
463             {
464                 // check to see which data var to load
465                 switch (linestr[16])
466                 {
467                     case ('W') :
468                         ParseLine(linestr, buffer.damw);
469                         break;
470                     case ('L') :
471                         ParseLine(linestr, buffer.daml);
472                         break;
473                     case ('H') :
474                         ParseLine(linestr, buffer.damh);
475                         break;
476                     default :;
477                 }
478                 // handle case in which TAB spacing is used
479                 switch (linestr[9])
480                 {
481                     case ('W') :
482                         ParseLine(linestr, buffer.damw);
483                         break;
484                     case ('L') :
485                         ParseLine(linestr, buffer.daml);
486                         break;
487                     case ('H') :
488                         ParseLine(linestr, buffer.damh);
489                         break;
490                     default :;
491                 }
492                 SkipLines(outfileptr, 1, linestr);
493             }
494             ParseLine(linestr, buffer.repstgy);
495             SkipLines(outfileptr, 1, linestr);
496
497             // retrieve remarks entry
498             // locate " 4.) Remarks : " line in .OUT file
499             while ((atoi(strtok(linestr, ","))) != 4)
500                 SkipLines(outfileptr, 1, linestr);
501             SkipLines(outfileptr, 2, linestr);
502         }
503     }

```

```

499         strcpy(buffer.remline, linestr);
500         trimstr(buffer.remline);
501         strcpy(remstr, "\0");
502         // if non-empty remarks section, build text lines
503         // into one continuous string -> "remstr"
504         while (strlen(buffer.remline) > 1)
505         {
506             strcat(remstr, buffer.remline);
507             strcpy(buffer.remline, "\n");
508             SkipLines(outfileptr, 1, linestr);
509             strcpy(buffer.remline, linestr);
510             trimstr(buffer.remline);
511         }
512         // (CSS) { long *memofilestats;
513         memofilestats=m4check(f4ref("REMARKS"));
514         // REMARKS LOADING IS NOT OPERATIONAL: will fix soon
515         { int i;
516             for(i=0;i<=5;i++)
517                 vfprintf(win,4,1+(6*i),"<%ld>",*(memofilestats+(i*sizeof(long))));
518         }
519
520         err = AddRepInfoRec(& buffer);
521
522         while ((linestr[0] != '\0') && (! feof(outfileptr)))
523             SkipLines(outfileptr, 1, linestr);
524         if (! feof(outfileptr))
525             SkipLines(outfileptr, 8, linestr);
526     }
527     fclose(outfileptr);
528     ArchiveFile(outfilename, ".OUT", indxrepid);
529     return (RMRSSUCCESS);
530 }
531 else
532 {
533     u4error(-3, "FILE ACCESS RMRSError!", "can't open .OUT file:", outfilename, (char *) 0)
534 }
535 return (-3);
536 }
537 void ArchiveFile(char * oldfilename, char * fileext, long indxrepid)
538 {
539     /*
540     * ARGUMENT
541     * (char *) oldfilename - present path-specified filename
542     * (char *) fileext     - filename extension
543     * (long)   indxrepid   - index repair id (unique for each file set)
544     *
545     * DESCRIPTION
546     * Transfers a RMRS-loaded file from the PDES Input directory to
547     * the PDES Archive directory; the new file gets the prefix "ARF"
548     * (for Archived Repair File) followed by the reference repair id number
549     * (left zero-padded to 5 digits) corresponding to the first repair in the
550     * file set; for example, if the file set designated B4058 was loaded, and
551     * the first repair from that set was mapped to repair id #303, then the
552     * corresponding archived files would be named: ARF00303.EQP, ARF00303.MAT,
553     * and ARF00303.OUT
554     *
555     * RETURNS
556     *
557     * NOTES
558     * (int) interrupt 56H status codes : 3 - Path not found
559     *                                     5 - Access denied
560     *                                     17 - Path not found
561     *
562     * DATABASES AFFECTED (whether direct or indirect)
563     * none
564     */
565     char archivefilename[50], str[25], errmess[50];
566     int status, len, i;
567     vfprintf(win, 4, 2 + fileloadcnt++, "");

```

```

569 strcpy(archivefilename, pdesarchvdirpath);
570 strcat(archivefilename, "ARF");
571
572 // convert indxrepid to ascii and pad left with zeros
573 ltoa(indxrepid, str, 10);
574 len = strlen(str);
575 for (i = 1; i <= len; i++)
576     str[5 - i] = str[len - i];
577 for (i = 0; i <= (4 - len); i++)
578     str[i] = '0';
579 str[5] = '\0';
580
581 strcat(archivefilename, str);
582 strcat(archivefilename, fileext);
583 status = move_file(oldfilename, archivefilename);
584 if (status)
585 {
586     switch (status)
587     {
588     case (3) :
589     case (17) : strcpy(errmess, "Path not found");
590     case (5) : strcpy(errmess, "Access denied");
591     default : strcpy(errmess, "Undefined error");
592     }
593     u4error(status, "File archive error"
594             , errmess
595             , " "
596             , oldfilename
597             , archivefilename, (char *) 0);
598 }
599 return;
600
601 int AddEqpRec(EQPREQREC eqprecbuf)
602 {
603     /*
604     * ARGUMENT
605     * (EQPREQREC) eqprecbuf - equipment requirement record buffer
606     *
607     * DESCRIPTION
608     * Populates an equipment requirement record and appends it to the
609     * EQPREQ.DBF database
610     *
611     * RETURNS
612     * (int) d4append() return status code
613     *
614     * DATABASES AFFECTED (whether direct or indirect)
615     * eqpreq_dbf (no index used)
616     */
617     d4select(eqpreq_dbf);
618     f4r_long(f4ref("EQPID"), 0L);
619     f4r_str(f4ref("EQPDESC"), eqprecbuf.eqpdescstr);
620     f4r_long(f4ref("START"), 0L);
621     f4r_long(f4ref("DURATION"), eqprecbuf.duration);
622     f4r_long(f4ref("REPID"), eqprecbuf.repid);
623     f4r_char(f4ref("STATUS"), eqprecbuf.status);
624     return (d4append());
625 }
626
627
628 int AddMatRec(char * matdescptr, float matqty, char * matunitptr, long repid, char status)
629 {
630     /*
631     * ARGUMENT
632     * (MATREQREC) matrecbuf - material requirement record buffer
633     *
634     * DESCRIPTION
635     * Populates a material requirement record and appends it to the
636     * MATREQ.DBF database
637     *
638     * RETURNS
639     * (int) d4append() return status code

```

```

639 *
640 * DATABASES AFFECTED (whether direct or indirect)
641 *   matreq_dbf (no index used)
642 *
643 */
644
645 // add material requirement record to MATREQ.DBF
646 d4select(matreq_dbf);
647 f4r_long(f4ref("MATID"), 0);
648 f4r_str(f4ref("MATDESC"), matdescptr);
649 f4r_double(f4ref("QTY"), matqty);
650 f4r_str(f4ref("UNIT"), matunitptr);
651 f4r_long(f4ref("REPID"), repid);
652 f4r_char(f4ref("STATUS"), status);
653 return (d4append());
654 }
655
656
657 int AddRepInfoRec(REPINFOREC * rec)
658 {
659 /*
660 * ARGUMENT
661 *   (REPINFOREC) repinfo_dbf - repair information record buffer
662 *
663 * DESCRIPTION
664 *   Populates a repair information record and appends it to the
665 *   REPINFO.DBF database
666 *
667 * RETURNS
668 *   (int) d4append() return status code
669 *
670 * DATABASES AFFECTED (whether direct or indirect)
671 *   repinfo_dbf (no index used)
672 */
673
674 // add repair .OUT file info record to REPINFO.DBF
675 d4select(repinfo_dbf);
676 f4r_long(f4ref("REPID"), rec->repid);
677 f4r_str(f4ref("FACFUNC"), rec->facfunc);
678 f4r_str(f4ref("FACDESC"), rec->facdesc);
679 f4r_int(f4ref("ELENUM"), rec->elenum);
680 f4r_str(f4ref("ELEDESC"), rec->eledesc);
681 f4r_str(f4ref("DAMMODE"), rec->dammode);
682 f4r_str(f4ref("DAMW"), rec->damw);
683 f4r_str(f4ref("DAML"), rec->daml);
684 f4r_str(f4ref("DAMH"), rec->damh);
685 f4r_str(f4ref("REPSTGY"), rec->repstgy);
686 #ifdef DEBUG
687 (CS$) printf("\nnumber written >%d<", m4write(f4ref("REMARKS"), d4recno(), rec->remline, 80));
688 #endif
689
690 return (d4append());
691 }
692
693 int SkipLines(FILE * fptr, int numoflines, char * returnline)
694 {
695 /*
696 * ARGUMENT
697 *   (FILE *) fptr - file pointer
698 *   (int) numoflines - number of lines to skip
699 *   (char*) returnline - string containing current line of file
700 *
701 * DESCRIPTION
702 *   Skips the specified number of lines in the specified file and
703 *   returns the current line
704 *
705 * RETURNS
706 *   (int) status code
707 *
708 * DATABASES AFFECTED
709 *   none
710 */

```

```

711     int i;
712
713     for (i = 1; i <= numoflines; i++)
714         fgets(returnline, 80, fptr);
715     return (1);
716 }
717
718 void ParseLine(char * linestr, char * returnstr)
719 {
720     /*
721     * ARGUMENT
722     * (char *) linestr - line to be parsed
723     * (char *) returnstr - return string
724     *
725     * DESCRIPTION
726     * Parses data lines within .OUT text file
727     *
728     * RETURNS
729     * DATABASES AFFECTED
730     * none
731     */
732     /*
733     strtok(linestr, ":");
734     strcpy(returnstr, strtok(NULL, "\n"));
735     */
736
737 long HrToSec(float hours)
738 {
739     /*
740     * ARGUMENT
741     * (float) hours - hours value to convert
742     *
743     * DESCRIPTION
744     * Converts hours to seconds
745     *
746     * RETURNS
747     * (long) number of seconds
748     */
749     return ((long) (hours * 3600.0));
750 }
751
752 int TryResAlloc(void)
753 {
754     /*
755     * ARGUMENT
756     *
757     * DESCRIPTION
758     * Prioritizes (N)ew and (O)verriden repairs, analyzes overall
759     * resource availability for each repair, and acts appropriately
760     * according to the results of the analysis. Thus, materials are
761     * allocated to the repair if no conflicts persist.
762     *
763     * Equipment conflicts result in (X)cancel-ation of repair.
764     * Compromise mode is available for a repair with only material
765     * conflicts, otherwise those repair is (S)uspended.
766     *
767     * RETURNS
768     * (int) status code
769     *
770     * DATABASES AFFECTED (whether direct or indirect)
771     * repair_dbf (repair_ndx1 - REPID index ) - d4seek used
772     * repair_ndx2 - prioritized index ) - d4skip used
773     * eqpsup_dbf (eqpsup_ndx1 - EQPDESC index ) - d4seek used
774     * eqpreq_dbf (eqpreq_ndx1 - REPID index ) - d4seek, d4skip used
775     * matsup_dbf (matsup_ndx1 - MATDESC index ) - d4seek used
776     * (matsup_ndx2 - MATID index ) - (CS$) to be added
777     * matreq_dbf (matreq_ndx1 - REPID index ) - d4seek, d4skip used
778     * matreq_ndx2 - MATID index ) - d4seek, d4skip used
779     */
780
781     long curr;

```

```
782 char eqpdescstr[26], matdescstr[26], repstatus;
783 REQSTNODE * reqchklist, * lastnode;
784 float matqtyneeded, matqtyavail;
785 long matid, ovrddrecno;
786 int status;
787 CONTEXT def1, def2;
788
789 vatputs(win, 2, 2, "Allocating material resources to repairs...");
790
791
792 // Index REPAIR.DBF for searching on "REPID"
793 // Used in GetRepairPrty() to retrieve priority
794 d4select(repair_dbf);
795 repair_ndx1 = i4open("REPRNDX1");
796 if (repair_ndx1 < 0)
797 {
798     u4error(-99, "repair_ndx1", (char *) 0);
799 }
800 // Index REPAIR.DBF for sequencing through (N)ew repairs
801 repair_ndx2 = i4open("REPRNDX2");
802 if (repair_ndx2 < 0)
803 {
804     u4error(-99, "repair_ndx2", (char *) 0);
805 }
806
807 // index EQPSUPP.DBF for searching on "EQPDESC"
808 // used for locating equipment requirement pieces in Air Base supply
809 d4select(eqpsup_dbf);
810 eqpsup_ndx1 = i4open("EQSUNDX1");
811 if (eqpsup_ndx1 < 0)
812 {
813     u4error(-99, "eqpsup_ndx1", (char *) 0);
814 }
815
816 // index EQPREQ.DBF for searching on "REPID"
817 // used to find all equipment requirements for a repair
818 d4select(eqpreq_dbf);
819 eqpreq_ndx1 = i4open("EQRENDX1");
820 if (eqpreq_ndx1 < 0)
821 {
822     u4error(-99, "eqpreq_ndx1", (char *) 0);
823 }
824
825 // index MATSUPP.DBF for searching on "MATDESC"
826 // used for locating material requirement pieces in Air Base supply
827 d4select(matsup_dbf);
828 matsup_ndx1 = i4open("MASUNDX1");
829 if (matsup_ndx1 < 0)
830 {
831     u4error(-99, "matsup_ndx1", (char *) 0);
832 }
833
834 // NOTICE: this index file must be added (CS 12/30)
835 // index MATSUPP.DBF for searching on "MATID"
836 // used for in AllocMats() for finding material listing in supply
837 matsup_ndx2 = i4open("MASUNDX2");
838 if (matsup_ndx2 < 0) {
839     u4error(-99, "matsup_ndx2", (char *) 0);
840 }
841
842 // index MATREQ.DBF for searching on "REPID"
843 // used to find all material requirements for a repair
844 d4select(matreq_dbf);
845 matreq_ndx1 = i4open("MARENDX1");
846 if (matreq_ndx1 < 0)
847 {
848     u4error(-99, "matreq_ndx1", (char *) 0);
849 }
850
851 // index MATREQ.DBF for searching on "MATID"
852 // used to search all instances of a particular material type
853 // for priority-override candidates
854 matreq_ndx2 = i4open("MARENDX2");
855 if (matreq_ndx2 < 0)
856 {
857     u4error(-99, "matreq_ndx2", (char *) 0);
858 }
```

```

849     }
850
851     // create and init data structure for resource requirements check list
852     reqchk1st = (REQLISTNODE *) malloc(sizeof (REQLISTNODE));
853     reqchk1st->type = 'H';
854     reqchk1st->recno = 0;
855     reqchk1st->availstatus = 'X';
856     reqchk1st->next = (REQLISTNODE *) NULL;
857     lastnode = reqchk1st;
858
859     // PROCESS ALL (NEW REPAIRS
860     d4select (repair_dbf);
861     d4top();
862     repstatus = f4char(f4ref("STATUS"));
863     // x4list();
864     // getkey();
865     // exit(0);
866     // while (((repstatus=f4char(f4ref("STATUS")))) == 'N') || (repstatus == 'O')) {
867     while ((repstatus = f4char(f4ref("STATUS"))) == 'N')
868     {
869         curr = f4long(f4ref("REPID"));
870
871         // CHECK EQP REQUIREMENTS FOR AVAILABILITY
872         d4select (eqpreq_dbf);
873         i4select (eqpreq_ndx1);
874         if (d4seek_double(curr) == FOUND)
875         {
876             while ((f4long(f4ref("REPID"))) == curr)
877             {
878                 // create check list node for eqp req
879                 lastnode->next = (REQLISTNODE *) malloc(sizeof (REQLISTNODE));
880                 lastnode = lastnode->next;
881                 lastnode->type = 'E';
882                 lastnode->recno = d4recno();
883                 lastnode->availstatus = 'x';
884                 lastnode->resrecno = 0;
885                 lastnode->resqty = 0.0;
886                 lastnode->ovrd = FALSE;
887                 lastnode->next = (REQLISTNODE *) NULL;
888
889                 // search EQPSUPP.DBF for eqp piece
890                 strcpy (eqpdescstr, f4str(f4ref("EQPDESC")));
891                 d4select (eqpsup_dbf);
892                 i4select (eqpsup_ndx1);
893                 uppercase (eqpdescstr);
894                 status = d4seek_str (eqpdescstr);
895                 if (status == FOUND)
896                 {
897                     // leave marked as (N)ew (in EQPREQ.DBF)
898                     lastnode->availstatus = 'F';
899                 }
900                 else
901                 {
902                     // mark as (M)issing (in EQPREQ.DBF)
903                     lastnode->availstatus = 'M';
904                     d4select (eqpreq_dbf);
905                     f4r_char (f4ref("STATUS"), 'M');
906                 }
907                 d4select (eqpreq_dbf);
908                 d4skip(1);
909             }
910
911             // CHECK MAT REQUIREMENTS FOR AVAILABILITY
912             d4select (matreq_dbf);
913             i4select (matreq_ndx1);
914             if (d4seek_double(curr) == FOUND)
915             {
916                 while ((f4long(f4ref("REPID"))) == curr)
917                 {
918                     lastnode->next = (REQLISTNODE *) malloc(sizeof (REQLISTNODE));
919                     lastnode = lastnode->next;
920                     lastnode->type = 'M';

```

```

916      lastnode->recno = d4recno();
917      lastnode->availstatus = 'x';
918      lastnode->resrecno = 0;
919      lastnode->resqty = 0.0;
920      lastnode->ovrd = FALSE;
921      lastnode->next = (REQLISTNODE *) NULL;
922      matqtyneeded = f4double(f4ref("QTY"));
923      // search MATSUPP.DBF for material
924      strcpy(matdescstr, f4str(f4ref("MATDESC")));
925      d4select(matsup_dbf);
926      i4select(matsup_ndx1);
927      uppercase(matdescstr);
928      status = d4seek_str(matdescstr);
929      if (status == FOUND)
930      {
931          matid = f4long(f4ref("MATID"));
932          matqtyavail = f4double(f4ref("QTY"));
933          d4select(matreq_dbf);
934          f4r_long(f4ref("MATID"), matid);
935          d4write(d4recno());
936          if (matqtyavail >= matqtyneeded)
937          {
938              lastnode->availstatus = 'E';
939          }
940          else
941          {
942              d4select(repair_dbf);
943              context_save(& def1);
944              d4select(matreq_dbf);
945              context_save(& def2);
946              /* (I)nadequate supply to meet needs, check override poss */
947              ovrddrecno = ChkOvrddPoss(matid, (matqtyneeded - matqtyavail), curr);
948              context_restore(& def1);
949              context_restore(& def2);
950              if (ovrddrecno)
951              {
952                  // (O)verride is possible, save record number
953                  lastnode->availstatus = 'O';
954                  lastnode->resrecno = ovrddrecno;
955              }
956              // (I)nadequate quantity in AB material supply
957              else
958              {
959                  lastnode->availstatus = 'I';
960              }
961          }
962          // matching entry not found in AB mat supply: (M)issing
963          else
964          {
965              lastnode->availstatus = 'M';
966          }
967          d4select(matreq_dbf);
968          d4skip(1);
969      }
970      }
971      else
972      {
973          // no materials associated with repair
974      }
975      // see if any resource conflicts arose
976      if (AnyConflicts(reqchk1st))
977      {
978          /* check to see if one was an equipment conflict */
979          if (iseqpconf(reqchk1st))
980          {
981              /* notify user compromise not possible & cancel repair */
982              alerttoeqpconf(); // (CS$) to be added
983              setrepairstatus(XCANCEL, curr);
984          }
985          else
986          {
987              /* ask user if he would like to attempt a compromise */
988              /* to resolve material conflicts */

```

```

981         if (askcomp())
981         {
982             d4select (repair_dbf);
983             i4select (repair_ndx2);
984             context_save (& def1);
985             if (DoCompromise (curr, reqchk1st) == SUCCESS)
986             {
987                 AllocMatls (reqchk1st, curr);
988                 /* leave eqp reqs marked (N)ew */
989             }
990             else
991             {
992                 /* unsuccessful compromise attempt */
993                 setrepairstatus (SUSPEND, curr);
994             }
995             context_restore (& def1);
996         }
997         else
998         {
999             /* user chose not to attempt compromise at this time */
1000             setrepairstatus (SUSPEND, curr);
1001         }
1002     }
1003     else
1004     {
1005         AllocMatls (reqchk1st, curr);
1006         /* leave eqp reqs marked (N)ew */
1007     }
1008     // get next repair
1009     d4select (repair_dbf);
1010     d4top();
1011     d4top();
1012     ClearChkLst (reqchk1st);
1013     iastnode = reqchk1st;
1014 }
1015 free ((REQLISTNODE *) reqchk1st);
1016 d4close_all();
1017 return (1);
1018 }
1019
1020 long ChkOvrD Poss (long matid, float matqtyneeded, long repid)
1021 {
1022     /*
1023     * ARGUMENT
1024     * (long)    matid        - RMRS system-assigned material id number
1025     *                of mat resource under conflict
1026     * (float)   matqtyneeded - quantity of material needed by the conflicting
1027     *                material requirement
1028     * (long)    repid        - RMRS system assigned repair id of repair to
1029     *                which the conflicting mat requirement belongs
1030     *
1031     * DESCRIPTION
1032     * This function searches through the (A)llocated material records
1033     * corresponding to 'matid' in an attempt to find the lowest priority
1034     * repair that is holding the least qty of sufficient resources to
1035     * statisfy the particular conflicting material need of the higher priority
1036     * repair specified by 'repid'. If one is found, the corresponding record
1037     * number is returned. Otherwise, zero is returned.
1038     *
1039     * RETURNS
1040     * (long)    n - record number of priority override candidate
1041     *                NOTE: This is the record number of the material requirement
1042     *                record in the MATREQ.DBF from which the materials will
1043     *                be taken.
1044     *                0 - none found
1045     *
1046     * Databases affected
1047     * repair_dbf (repair_ndx1 - REPID index) - d4seek used
1048     * matreq_dbf (matreq_ndx2 - MATID index) - d4seek, d4skip used
1049     */

```

```

1049     long same, selrecno = 0L, xrecno, xrepid, xmatid;
1050     int minprty, xprty, flgprty;
1051     char xstatus;
1052     float xqty, currminqty;
1053
1054     // Get priority of the conflict-owning repair
1055     flgprty = GetRepairPrty(repid);
1056     minprty = flgprty;
1057
1058     // MATREQ.DBF: indexed on "MATID"
1059     d4select(matreq_dbf);
1060     i4select(matreq_ndx2);
1061     // this index file groups like materials together, so find first
1062     // and skip sequentially through the list
1063     same = matid;
1064     d4seek_double(matid);
1065     d4recno();
1066     xmatid = f4long(f4ref("MATID"));
1067     // check all matching (A)llocated material records for
1068     // quantity available and associated priority
1069     // NOTE: vars w/ prefix 'x' hold data belonging to current
1070     // mat requirement under scrutiny
1071     do
1072     {
1073         xstatus = f4char(f4ref("STATUS"));
1074         xqty = f4double(f4ref("QTY"));
1075         xrepid = f4long(f4ref("REPID"));
1076         xrecno = d4recno();
1077         xprty = GetRepairPrty(xrepid);
1078         if (xstatus == 'A')
1079         {
1080             // does this mat req belong to a repair with a LOWER priority
1081             // (higher number) and does the it have at least the qty needed ??
1082             if ((xprty > minprty) && (xqty >= matqtyneeded))
1083             {
1084                 minprty = xprty;
1085                 currminqty = xqty;
1086                 selrecno = xrecno;
1087             }
1088             // does this mat req belong to a repair with the SAME priority
1089             // as the current minimum selected mat req and does the it have a
1090             // LOWER qty than the current minimum selected mat req
1091             else if ((xprty == minprty) && (xprty != flgprty) &&
1092                     (xqty >= matqtyneeded) && (xqty < currminqty))
1093             {
1094                 currminqty = xqty;
1095                 selrecno = xrecno;
1096             }
1097         }
1098         d4select(matreq_dbf);
1099         i4select(matreq_ndx2);
1100         d4skip(1);
1101         xmatid = f4long(f4ref("MATID"));
1102     }
1103     while ((xmatid == same) && (! d4eof()));
1104     return (selrecno);          /* return subordinate matreq record number, 0 for none*/
1105 }
1106
1107 int AnyConflicts(REQLSTNODE * lstptr)
1108 {
1109     /*
1110     * ARGUMENT
1111     * (REQLSTNODE *) lstptr - pointer to resource availability analysis list
1112     *
1113     * DESCRIPTION
1114     * Checks the completed resource availability analysis list for any
1115     * conflicts.
1116     *
1117     * Non-conflict statuses: for equipment .... (F)ound
1118     *                          for material..... (E)xistent
1119     *
1120     * note: this availability status refers to the codes used in the
1121     * resource availability analysis linked list and not the STATUS fields
1122     */

```

```
1117 *   used in the databases
1118 *
1119 * RETURNS
1120 *   (int) number of resource availability conflicts for the repair
1121 *
1122 * DATABASES AFFECTED
1123 *   none
1124 */
1125
1126   int numofconflicts;
1127
1128   numofconflicts = 0;
1129   while (lstptr->next != (REQLSTNODE *) NULL)
1129   {
1130       lstptr = lstptr->next;
1131       switch (lstptr->availstatus)
1131       {
1132           case ('F') :
1133           case ('E') : break;
1134           default :
1135               numofconflicts++;
1136       }
1137   }
1138   return (numofconflicts);
1139 }
1140
1141 void ClearChkLst (REQLSTNODE * lstptr)
1141 {
1142 /*
1143  * ARGUMENT
1144  *   (REQLSTNODE *) lstptr - pointer to resource availability analysis list
1145  *
1146  * DESCRIPTION
1147  *   Clears list, leaves headnode
1148  *
1149  * RETURNS
1150  *
1151  * DATABASES AFFECTED
1152  *   none
1153  *
1154  */
1155
1156   REQLSTNODE * temp;
1157
1158   temp = lstptr;
1159   lstptr = lstptr->next;
1160   temp->next = (REQLSTNODE *) NULL;
1161   while (lstptr != (REQLSTNODE *) NULL)
1161   {
1162       temp = lstptr->next;
1163       lstptr->next = (REQLSTNODE *) NULL;
1164       free((REQLSTNODE *) lstptr);
1165       lstptr = temp;
1166   }
1167 }
1168
1169 void PrintChkLst (REQLSTNODE * lstptr)
1169 {
1170 /*
1171  * ARGUMENT
1172  *   (REQLSTNODE *) lstptr - pointer to resource availability analysis list
1173  *
1174  * DESCRIPTION
1175  *   Prints list to stdout, used for debug
1176  *
1177  * RETURNS
1178  *
1179  * DATABASES AFFECTED
1180  *   none
1181  *
1182  */
1183
1184   printf("\n");
1185   while (lstptr != (REQLSTNODE *) NULL)
```

```

1185     {
1186         printf("\n TYPE = >%c< recno = >%d< ", lstptr->type, lstptr->recno);
1187         printf(" AVAILSTATUS = >%c<", lstptr->availstatus);
1188         lstptr = lstptr->next;
1189     }
1190 }
1191
1192 int AllocMatls(REQLSTNODE * lstptr, long repid)
1193 {
1194     /*
1195     * ARGUMENT
1196     * (REQLSTNODE *) lstptr - pointer to resource availability analysis list
1197     * (long) repid - repair id
1198     *
1199     * DESCRIPTION
1200     * Allocates materials according to allocation condition information
1201     * within the resource availability analysis list
1202     *
1203     * RETURNS
1204     * (int) status
1205     */
1206     long matid, xmatid, xrepid, smatid;
1207     char smatdesc[30], sunit[5];
1208     float qntyneeded, xqty, sqty;
1209
1210     // loop until no more materials left to allocate
1211     while (lstptr->next != (REQLSTNODE *) NULL)
1212     {
1213         lstptr = lstptr->next;
1214         switch (lstptr->type)
1215         {
1216             case ('M') :
1217                 /* PRIORITY OVERRIDE CASE */
1218                 if ((lstptr->availstatus == 'O') && (lstptr->ovrd == TRUE))
1219                 {
1220                     /* DUMP MATERIALS OF OVERRIDEN REPAIR BACK INTO SUPPLY */
1221                     if (d4select(matreq_dbf) < 0)
1222                     {
1223                         u4error(-1, "", (char *) 0);
1224                         return (RMRSError);
1225                     }
1226                     if (i4select(matreq_ndx1) < 0)
1227                     {
1228                         u4error(-2, "", (char *) 0);
1229                         return (RMRSError);
1230                     }
1231                     d4go(lstptr->resrecno);
1232                     xrepid = f4long(f4ref("REPID"));
1233                     d4top();
1234                     if (d4seek_double(xrepid))
1235                         u4error(-9999, "SEEK REPID RMRSError", "repid not found", "FN: AllocateMate
1236 rials", (char *) 0);
1237                     // skip sequentially through mat reqs for subordinate repair
1238                     while ((f4long(f4ref("REPID")) == xrepid) && (! d4eof()))
1239                     {
1240                         xmatid = f4long(f4ref("MATID"));
1241                         xqty = f4double(f4ref("QTY"));
1242                         // search material supply on MATID
1243                         if (d4select(matsup_dbf) < 0)
1244                         {
1245                             u4error(-3, "", (char *) 0);
1246                             return (RMRSError);
1247                         }
1248                         i4select(-1); // to be replaced (CS$)
1249                         d4top(); // to be replaced (CS$)
1250                         i4select(matsup_ndx2); /* to be indexed on MATID (CS$) */
1251                         d4seek_double(xmatid); /* $$(CS$) */
1252                         while ((xmatid != f4long(f4ref("MATID"))) && (! d4eof()))
1253                         {
1254                             d4skip(1);
1255                             f4r_double(f4ref("QTY"), (double) (f4double(f4ref("QTY")) + xqty));
1256                             d4write(d4recno());
1257                             if (d4select(matreq_dbf) < 0)
1258                             {

```

```

1249         u4error(-5, "", (char *) 0);
1250         return (RMRSError);
1251     }
1252     if (i4select(matreq_ndx1) < 0)
1253     {
1254         u4error(-6, "", (char *) 0);
1255         return (RMRSError);
1256     }
1257     d4skip(1);
1258     /* SET AVAIL STATUS TO 'O' (OVERRIDEN) */
1259     setrepairstatus(OVERRIDE, xrepid);
1260 }
1261 // MATERIAL SUBSTITUTION CASE
1262 else if ((lstptr->ovrd == FALSE) && (lstptr->availstatus != 'E'))
1263 {
1264     /* SUBSTITUTION OF MATERIAL for conflicting requirement */
1265     /* get mat info for substitute */
1266     d4select(matsup_dbf);
1267     d4go(lstptr->resrecno);
1268     sqty = lstptr->resqty;
1269     smatid = f4long(f4ref("MATID"));
1270     strcpy(smatdesc, f4str(f4ref("MATDESC")));
1271     strcpy(sunit, f4str(f4ref("UNIT")));
1272     /* replace matreq with the substitution material */
1273     d4select(matreq_dbf);
1274     d4go(lstptr->recno);
1275     f4r_long(f4ref("MATID"), smatid);
1276     f4r_str(f4ref("MATDESC"), smatdesc);
1277     f4r_double(f4ref("QTY"), sqty);
1278     f4r_str(f4ref("UNIT"), sunit);
1279     d4write(lstptr->recno);
1280 }
1281 // non-conflicting material requirements begin execution here
1282 /* ALLOCATE MATERIALS FROM SUPPLY AS USUAL */
1283 d4select(matreq_dbf);
1284 d4go(lstptr->recno);
1285 qntyneeded = f4double(f4ref("QTY"));
1286 matid = f4long(f4ref("MATID"));
1287 d4select(matsup_dbf);
1288 i4select(-1); // to be replaced (CS$)
1289 d4top(); // to be replaced (CS$)
1290 /* i4select(matsup_ndx2); */ /* MATID $$$ (CS$) */
1291 /* d4seek_double(xmatid); */ /* $$$ (CS$) */
1292 while (matid != f4long(f4ref("MATID"))) /* (CS$) $$$ MATID */
1293 {
1294     d4skip(1);
1295     f4r_double(f4ref("QTY"), (f4double(f4ref("QTY")) - qntyneeded));
1296     break;
1297 }
1298 case ('E') : break; // do nothing
1299 default : u4error(-7, "Invalid node TYPE"
1300             , "in resource requirement"
1301             , "linked list"
1302             , lstptr->type, (char *) 0);
1303 }
1304 }
1305 // set STATUS of repair id record to (P)ossible
1306 d4select(repair_dbf);
1307 d4seek_double(repid);
1308 f4r_char(f4ref("STATUS"), 'P');
1309 // set STATUS of material records to (A)llocated
1310 d4select(matreq_dbf);
1311 d4seek_double(repid);
1312 while (f4long(f4ref("REPID")) == repid)
1313 {
1314     f4r_char(f4ref("STATUS"), 'A');
1315     d4skip(1);
1316 }
1317 return (RMRSSUCCESS);
1318 }
1319 int setrepairstatus(int tostatus, long repid)
1320 {
1321     /*
1322     * ARGUMENT

```

```

1319 * (int) tostatus - predefined mnemonic constant corresponding to
1320 * the status to which the repair is changed to
1321 * (e.g. SUSPEND, XCANCEL, OVERRIDE)
1322 * (long) repid - repair id
1323 *
1324 * DESCRIPTION
1325 * Changes STATUS fields of both the repair id record and associated
1326 * resource requirement records to the appropriate code
1327 *
1328 * RETURNS
1329 * (int) status code
1330 *
1331 * DATABASES AFFECTED (whether direct or indirect)
1332 * repair_dbf (repair_ndx1 - REPID index) - d4seek used
1333 * eqpreq_dbf (eqpreq_ndx1 - REPID index) - d4seek, d4skip used
1334 * matreq_dbf (matreq_ndx1 - REPID index) - d4seek, d4skip used
1335 *
1336 */
1337
1338 float matqtytoreturn, supplyqty;
1339 long matid;
1340 char status;
1341
1342 switch (tostatus)
1343 {
1344 case (XCANCEL) : status = 'X';
1345 break;
1346 case (SUSPEND) : status = 'S';
1347 break;
1348 case (OVERRIDE) : status = 'O';
1349 break;
1350 default ;;
1351 }
1352 d4select(repair_dbf);
1353 i4select(repair_ndx1);
1354 d4seek_double(repid);
1355 f4r_char(f4ref("STATUS"), status);
1356
1357 d4select(eqpreq_dbf);
1358 i4select(eqpreq_ndx1);
1359 d4seek_double(repid);
1360 while (f4long(f4ref("REPID")) == repid)
1361 {
1362 f4r_char(f4ref("STATUS"), status);
1363 d4skip(1);
1364 }
1365 d4select(matreq_dbf);
1366 i4select(matreq_ndx1);
1367 d4seek_double(repid);
1368 while (f4long(f4ref("REPID")) == repid)
1369 {
1370 if (f4char(f4ref("STATUS")) == 'A')
1371 {
1372 matqtytoreturn = f4double(f4ref("QTY"));
1373 matid = f4long(f4ref("MATID")); // $$ to be used w/ indexed search
1374 f4r_char(f4ref("STATUS"), status);
1375 d4select(matsup_dbf);
1376 i4select(-1);
1377 d4top();
1378 while ((matid != f4long(f4ref("MATID"))) && (! d4eof()))
1379 d4skip(1); // $$ change to index search (ndx on "MATID")
1380 if (! d4eof())
1381 {
1382 supplyqty = f4double(f4ref("QTY"));
1383 f4r_double(f4ref("QTY"), supplyqty + matqtytoreturn);
1384 }
1385 }
1386 d4select(matreq_dbf);
1387 i4select(matreq_ndx1);
1388 f4r_char(f4ref("STATUS"), status);
1389 d4skip(1);
1390 }
1391 return (SUCCESS);
1392 }

```

```

1388
1389 //-----
1390 long GetFacnum(char * facnumstrptr)
1391 {
1392 /*
1393  * ARGUMENT
1394  *   (char *) facnumstrptr - pointer to facility number string (e.g. "B4058")
1395  * DESCRIPTION
1396  *   Accepts facility number string and returns a long representation
1397  *   ex. "B4058" -> (long)4058
1398  *
1399  * RETURNS
1400  *   facility number designation (long)
1401  *
1402  */
1403 /*
1404   char tempstr[25];
1405   int i;
1406
1407   // strip 'B' off of facility designation and convert to int
1408   strcpy(tempstr, facnumstrptr);
1409   for (i = 1; tempstr[i] != '\0'; i++)
1410       tempstr[i - 1] = tempstr[i];
1411   tempstr[i - 1] = '\0';
1412
1413   return (atol(tempstr));
1414 }
1415
1416 long GetNxtAvlRepid(void)
1417 {
1418 /*
1419  * ARGUMENT: none
1420  * DESCRIPTION
1421  *   Gets the next available RMRS repair id number; checks for
1422  *   empty database situation
1423  *
1424  * RETURNS
1425  *   (long) repair id number
1426  *
1427  * DATABASES AFFECTED (whether direct or indirect)
1428  *   repair_dbf (no index used)
1429  */
1430 /*
1431   // assign unique RMRS id # to repair
1432   d4select(repair_dbf);
1433   i4select(-1);
1434   if (d4bottom() == 3)
1435       return (1);
1436   else
1437       return (f4long(f4ref("REPID")) + 1);
1438 }
1439
1440 int AddRepRec(long facnum, long repid, int priority)
1441 {
1442 /*
1443  * ARGUMENT
1444  *   (long) facnum - facility number
1445  *   (long) repid - repair id
1446  *   (int) priority - AB mission-critical facility priority assignment
1447  *
1448  * DESCRIPTION
1449  *   Adds a repair data record to the end of REPAIR.DBF
1450  *
1451  * RETURNS
1452  *   (int) status of d4append() operation
1453  *
1454  * DATABASES AFFECTED (whether direct or indirect)
1455  *   repair_dbf (no index used)
1456  */
1457 /*
1458

```

```
1459 // add repair record to REPAIR.DBF
1460 d4select(repair_dbf);
1461 f4r_long(f4ref("FACNUM"), facnum);
1462 f4r_long(f4ref("REPID"), repid);
1463 f4r_int(f4ref("PRIORITY"), priority);
1464 f4r_int(f4ref("SCHDPRTY"), priority);
1465 f4r_char(f4ref("STATUS"), 'N');
1466
1467 return (d4append());
1468 }
1469
1470 void cleanwin(void)
1471 {
1472     int i;
1473     for (i = 1; i <= 7; i++)
1474         vatputf(win, i, 0, "
");
1475 }
1476
1477 int GetRepairPrty(long repid)
1478 {
1479     /*
1480      * ARGUMENT
1481      * (long) repid - repair id
1482      *
1483      * DESCRIPTION
1484      * Gets priority of repair
1485      *
1486      * RETURNS
1487      * (int) priority
1488      *
1489      * DATABASES AFFECTED (whether direct or indirect)
1490      * repair_dbf (repair_ndx1 - REPID index ) - d4seek used
1491      */
1492
1493     int prty;
1494
1495     d4select(repair_dbf);
1496     i4select(repair_ndx1);
1497     d4seek_double(repid);
1498     prty = f4int(f4ref("PRIORITY"));
1499     return (prty);
1500 }
```

```

1  /*
2  *  Procutil.c
3  *
4  *      functions used by the PROCESS menu selections
5  *
6  */
7
8
9  #include <stdio.h>
10 #include <stdlib.h>
11 #include <string.h>
12 #include <dos.h>
13
14 #include "sysdefs.h"
15 #include "d4base.h"
16 #include "u4error.h"
17 #include "miscutil.h"
18
19 #include "dw.h"
20
21 HWND xdsplymessbox(void);
22
23 int move_file(char * oldname, char * newname)
24 {
25  /*
26  *  ARGUMENT
27  *      (char *) oldname - old filename string
28  *      (char *) newname - new filename string
29  *
30  *  DESCRIPTION
31  *      Changes the filename specification by modifying the FCB
32  *
33  *  RETURNS
34  *      (int) status code:
35  *          interrupt 56H status codes : 3 - Path not found
36  *                                       5 - Access denied
37  *                                       17 - Path not found
38  *
39  */
40
41  union REGS regs; struct SREGS sregs;
42  int ret;
43  regs.h.ah = 0x56;
44  sregs.ds = FP_SEG(oldname);
45  regs.x.dx = FP_OFF(oldname);
46  sregs.es = FP_SEG(newname);
47  regs.x.di = FP_OFF(newname);
48  ret = intdosx(& regs, & regs, & sregs);
49
50  /* if carry flag is set, there was an error */
51  ret = regs.x.ax;
52  return (regs.x.cflag ? ret : 0);
53 }
54
55 int iseqpconf(REQLSTNODE * lstptr)
56 {
57  /*
58  *  ARGUMENT
59  *      (REQLSTNODE *) lstptr - pointer to resource availability analysis list
60  *
61  *  DESCRIPTION
62  *      Checks the resource availability analysis list to see if any equipment
63  *      conflicts exist
64  *
65  *  RETURNS
66  *      (int) TRUE or FALSE
67  *
68  */
69  if (! lstptr)
70  {
71      u4error(EMPTYLST, "Unexpected empty list error", (char *) 0);
72      return (FALSE);
73      /* error */
74  }
75  */

```

```

72     for (lstptr = lstptr->next; lstptr->type == 'E'; lstptr = lstptr->next)
73         if (lstptr->availstatus != 'F')
74             return (TRUE);          /* eqp conf found      */
75     return (FALSE);                /* no conflicts        */
76 }
77
78 HWND xdsplymessbox(void)
79 {
80     /*
81     * ARGUMENT
82     * DESCRIPTION
83     *   Sets up and displays a standard message box
84     *
85     * RETURNS
86     *   (HWND) Handle to window of message box
87     *
88     */
89
90     HWND messwin;
91
92     messwin = vcreat(8, 30, GREYBOX, YES);
93     vlocate(messwin, 8, 24);
94     vframe(messwin, GREYBOX, FRDOUBLE);
95     vshadow(messwin, CURRENT, SHADOW100, BOTTOMRIGHT);
96
97     visible(messwin, YES, YES);
98     return (messwin);
99 }
100
101
102 int askcomp(void)
103 {
104     /*
105     * ARGUMENT
106     * DESCRIPTION
107     *   Asks the user if a COMPROMISE should be attempted
108     *
109     * RETURNS
110     *   (int) TRUE or FALSE
111     *
112     */
113
114     HWND win;
115     unsigned int key;
116
117     win = xdsplymessbox();
118     vratps(1, 0, GRYBTXT, " // REPAIR HAS MATERIAL .. ");
119     modattr(win, 1, 2, 2, SELQTYBLINK);
120     modattr(win, 1, 27, 2, SELQTYBLINK);
121     vratps(2, 0, GRYBTXT, "          CONFLICTS          ");
122     vratps(4, 0, GRYBTXT, " substitution may be possible ");
123     vratps(6, 0, KEYMESS, "   Do you wish to enter   ");
124     vratps(7, 0, KEYMESS, "   COMPROMISE mode (N): ?   ");
125     modattr(win, 7, 25, 3, FLSHCHR);
126     while (1)
127     {
128         key = getkey();
129         switch (key)
130         {
131             case ('N') :
132             case ('n') :
133             case (ENTR) : vratps(7, 26, HELP, "N");
134                         delay(2);
135                         vdelete(win, NONE);
136                         dmpxtrakeys();
137                         return (FALSE);
138             case ('Y') :
139             case ('y') : vratps(7, 26, HELP, "Y");
140                         delay(2);
141                         vdelete(win, NONE);
142                         dmpxtrakeys();
143                         return (TRUE);

```

```
142      default : vbeep();          /* INVALID keystroke      */
143      }
144  }
145 }
146
147
148
```

```
1  /*****
2  *
3  *   reset.c
4  *
5  *   This program will ZAP all the records in the
6  *   databases that are populated by processed
7  *   repairs and reset all material supply
8  *   quantities to 1000.0.
9  *
10 *   by Chris Sawyer 10/6/91
11 *****/
12
13 #include <stdio.h>
14 #include <stdlib.h>
15 #include "d4base.h"
16 #include "rmrsutil.h"
17
18 int reset(void)
19 {
20
21     open_repair(0);
22     d4zap(1L, d4reccount());
23     d4close_all();
24
25     open_matreq(0);
26     d4zap(1L, d4reccount());
27     d4close_all();
28
29     open_eqpreq(0);
30     d4zap(1L, d4reccount());
31     d4close_all();
32
33     open_repinfo(0);
34     d4zap(1L, d4reccount());
35     d4close_all();
36
37     open_matsup(0);
38     d4top();
39     while (! d4eof())
40     {
41         f4r_double(f4ref("QTY"), 1000.0);
42         d4skip(1);
43     }
44     d4pack();
45     d4close_all();
46
47     open_eqpsup(0);
48     d4pack();
49     d4close_all();
50     fcloseall();
51 }
```

```

1 #include "d4all.h"
2 #include "w4.h"
3 #include "g4char.h"
4 #include "p4misc.h"
5 #include "rmrs.h"
6
7 #include <string.h>
8 #include <ctype.h>
9
10 #include "rmrsclr.h"
11
12 static void b4set_row(int, long);
13 static int b4modify(int);
14 static int b4skip_one(int);
15 static int b4page(int);
16 static int b4top(int);
17 static int b4find(int);
18 static int b4bottom(int);
19 static int b4char(void);
20 static void b4edit_setup(void);
21 static void b4browse_setup(void);
22 static int b4setup(int (*) (void), int (*) (void));
23 static void b4deact_menu(void);
24 static int b4add_copy(int);
25 static int b4select_index(int);
26 static int b4delete(int);
27 static int b4undelete(int);
28 static int b4go_rec(void); /* Goes to the record at 'b4cursor_row'. */
29 static int b4empty_check(void);
30
31 static int (* user_edit_setup) (void) = 0;
32 static int (* user_browse_setup) (void) = 0;
33 static int b4on_browse = -1;
34 static int b4cur_ref = -1;
35
36 static int b4pull_ref = -1;
37 static int b4select_ref = -1;
38
39 static int b4page_size = -1;
40 static int b4top_margin = 3;
41 static int b4cur_top_margin = -1;
42 static int b4bottom_margin = 1;
43 static int b4cursor_col = -1;
44 static int last_read_char = 0;
45 static int is_empty = 0;
46 static int do_escape = 0;
47
48 /* Action routines assume that the record at 'b4cursor_row' is
49 'b4cursor_rec'. */
50 static int b4cursor_row;
51 static long b4cursor_rec; /* The record at 'b4cursor_row'. This info
52 is filled in by 'b4display_recs'.
53 It becomes '-1L' if there was no record
54 at the cursor. */
55 static long b4start_rec; /* Saved by 'b4display_assume'.
56 This is the starting record number. */
57 static int b4num_displayed; /* Saved by 'b4display_recs'. */
58
59 static int b4display_one(int);
60 static int b4display_assume(void); /* Assumes that 'b4cursor_rec'
61 is at 'b4cursor_row'. */
62 static int b4display_recs(long);
63 static void b4display_cursor(void);
64
65 extern CB_WINDOW * v4window_ptr;
66 extern GET * v4get;
67 extern INDEX * v4index;
68
69
70 static long natt1 = KW, natt2 = WK, matt1 = KW, matt2 = WK;
71
72
73 static int b4go_rec()
74 {

```

```

75     if (! is_empty)
76         if (b4cursor_rec != d4recno())
77             d4go(b4cursor_rec);
78     return 0;
79 }
80
81 static int  b4empty_check()
82 {
83     if (d4eof())
84         d4top();
85     else
86     {
87         if (d4skip(1L) == -3)
88             d4top();
89         else
90             d4skip(-1L);
91     }
92     is_empty = 1;
93     b4cursor_rec = 0L;
94     if (! d4eof())
95     {
96         is_empty = 0;
97         b4cursor_rec = d4recno();
98     }
99     return 0;
100 }
101
102 static int (* verify_routine) (int) = 0;
103 #ifdef KR
104     void b4verify( verify_rou )
105     int (*verify_rou)() ;
106 #else
107     void b4verify(int (* verify_rou) (int));
108 #endif
109 {
110     verify_routine = verify_rou;
111 }
112
113 static int (* call_routine) (int, int) = 0;
114 #ifdef KR
115     void b4call( call_rou )
116     int (*call_rou)() ;
117 #else
118     void b4call(int (* call_rou) (int, int));
119 #endif
120 {
121     call_routine = call_rou;
122 }
123
124
125 static int  temp_ref = -1;
126
127 static void  b4deact_menu()
128 {
129     if (w4select(-1) != b4pull_ref && w4select(-1) != b4cur_ref)
130     {
131         temp_ref = w4select(-1);
132         n4refresh(temp_ref);
133         w4deactivate(temp_ref);
134     }
135 }
136
137 static void  b4set_row(int new_row, long new_attribute)
138 {
139     int get_on;
140     GET * get_ptr;
141
142     new_row += b4cur_top_margin;
143
144     for (get_on = v4window_ptr->first_get; get_on >= 0; get_on = get_ptr->next)
145     {
146         get_ptr = v4get + get_on;
147         if (b4on_browse)
148             get_ptr->row = new_row;

```

```
149     get_ptr->attribute = new_attribute;
150   )
151 }
152
153 static int b4modify(int is_modify)
154 {
155     int rc;
156
157     if (is_modify)
158         if (!is_empty) return 0;
159
160     b4deact_menu();
161     if (is_modify)
162         b4go_rec();
163
164     w4activate(b4cur_ref);
165     b4set_row(b4cursor_row, KW);
166
167     for (;;)
168     {
169         if (is_modify)
170             d4unlock(-1L);
171
172         last_read_char = g4read();
173         if (last_read_char == ESC || last_read_char == CTRL_Q)
174         {
175             last_read_char = 0;
176             if (is_modify)
177                 d4go(b4cursor_rec);
178             else
179                 return ESC;
180
181             break;
182         }
183         else
184         {
185             if (!u4ptr_equal((void *) 0, (void *) verify_routine))
186                 rc = (*verify_routine) (b4cur_ref);
187             else
188                 rc = 0;
189
190             if (rc == 0)
191             {
192                 if (is_modify) d4lock(d4recno(), 1);
193                 if (!u4ptr_equal((void *) call_routine, (void *) 0))
194                     (*call_routine) (b4cur_ref, b4cursor_row + b4cur_top_margin);
195
196                 if (is_modify)
197                     rc = d4write(d4recno());
198                 else
199                     rc = d4append();
200                 if (rc == -3) continue;
201
202                 d4flush(d4select(-1));
203                 if (last_read_char == RETURN || last_read_char == DOWN)
204                     last_read_char = 0;
205                 break;
206             }
207         }
208     }
209
210     if (is_modify)
211     {
212         b4display_assume();
213         b4display_cursor();
214     }
215
216     d4unlock(-1L);
217     return 0;
218 }
219
220 static int b4add_copy(int is_blank)
221 {
222     int rc, save_row;
```

```

223
224     if (is_blank)
225         d4go(0L);
226     else
227         b4go_rec();
228
229     save_row = b4cursor_row;
230     b4cursor_row = b4num_displayed;
231     if (b4cursor_row >= b4page_size)
232         b4cursor_row--;
233
234     rc = b4modify(0);
235     if (rc != 0)
236         b4cursor_row = save_row;
237     else
238     {
239         b4cursor_rec = d4recno();
240         b4go_rec();
241         b4empty_check();
242     }
243
244     b4display_assume();
245     return 0;
246 }
247
248
249 static void b4display_cursor()
250 {
251     w4select(b4cur_ref);
252     w4cursor(b4cur_top_margin + b4cursor_row, b4cursor_col);
253 }
254
255 static int b4display_one(int disp_row)
256 {
257     int first_get;
258
259     b4set_row(disp_row, KC);
260     first_get = v4window_ptr->first_get;
261
262     if (! u4ptr_equal((void *) call_routine, (void *) 0))
263         (* call_routine) (b4cur_ref, disp_row + b4cur_top_margin);
264
265     if (d4deleted())
266         w4(v4get[first_get].row, v4get[first_get].col - 1, "");
267     else
268         w4(v4get[first_get].row, v4get[first_get].col - 1, " ");
269
270     g4display();
271     return 0;
272 }
273
274 /* Returns the number of records displayed.
275    Stores record at cursor in 'b4cursor_rec'.
276 */
277
278 static int b4display_recs(long start_rec)
279 {
280     int i;
281
282     b4num_displayed = 0;
283     w4activate(b4cur_ref);
284
285     b4cursor_rec = -1L;
286     if (is_empty)
287         d4go(0L);
288     else
289         d4go(start_rec);
290
291     for (i = 0; i < b4page_size; i++)
292     {
293         b4display_one(i);
294
295         if (! d4eof())
296             ;

```

```
297         b4num_displayed++;
298         if (i == b4cursor_row)
299             b4cursor_rec = d4recno();
300
301         d4skip(1L);
302     }
303 }
304
305     return 0;
306 }
307
308 static int b4display_assume()
309 {
310     b4start_rec = d4recno();
311     b4go_rec();
312     d4skip((long) ~ b4cursor_row);
313     b4display_recs(d4recno());
314     b4display_cursor();
315
316     return 0;
317 }
318
319 static int b4top(int junk_parm)
320 {
321     d4top();
322     b4cursor_row = 0;
323     b4cursor_rec = d4recno();
324
325     b4deact_menu();
326     b4display_assume();
327
328     d4unlock(-1L);
329     return 0;
330 }
331
332 static int b4bottom(int junk_parm)
333 {
334     long bot_rec;
335
336     d4bottom();
337     bot_rec = d4recno();
338     d4skip((long) (1 - b4page_size));
339
340     b4deact_menu();
341     b4display_recs(d4recno());
342     b4cursor_rec = bot_rec;
343     b4cursor_row = b4num_displayed - 1;
344     b4display_cursor();
345
346     d4unlock(-1L);
347     return 0;
348 }
349
350 static int b4skip_one(int n)
351 {
352     int rc;
353
354     b4go_rec();
355     rc = d4skip((long) n);
356     if (rc < 0) return -1;
357     if (rc == 1) return 0;
358     if (rc == 3) return 0;
359
360     b4cursor_row += n;
361     b4cursor_rec = d4recno();
362     if (b4cursor_row < 0)
363     {
364         b4cursor_row = 0;
365         b4deact_menu();
366         b4display_assume();
367         return 0;
368     }
369
370     if (b4cursor_row >= b4page_size)
```

```

371     {
372         b4cursor_row = b4page_size - 1;
373         b4deact_menu();
374         b4display_assume();
375         return 0;
376     }
377
378     b4display_cursor();
379     d4unlock(-1L);
380     return 0;
381 }
382
383 static int b4page(int no_recs)
384 {
385     b4go_rec();
386     if (no_recs < 0 && b4cursor_row > 0)
387     {
388         d4skip((long) - b4cursor_row);
389         b4cursor_rec = d4recno();
390         b4cursor_row = 0;
391         b4display_cursor();
392         return 0;
393     }
394
395     if (no_recs > 0 && b4cursor_row < b4num_displayed - 1)
396     {
397         d4skip((long) (b4num_displayed - 1 - b4cursor_row));
398         b4cursor_rec = d4recno();
399         b4cursor_row = b4num_displayed - 1;
400         b4display_cursor();
401         return 0;
402     }
403
404     d4skip((long) no_recs);
405     if (d4eof()) d4skip(-1L);
406     b4cursor_rec = d4recno();
407     b4deact_menu();
408     b4display_assume();
409     d4unlock(-1L);
410     return 0;
411 }
412
413 void b4margin(int top_margin, int bottom_margin)
414 {
415     b4top_margin = top_margin;
416     b4bottom_margin = bottom_margin;
417 }
418
419 static int b4help(int junk_parm)
420 {
421     int w_ref, startrow = 1, startcol = 1;
422
423     w_ref = w4define(2, 5, 23, 65);
424     w4popup();
425     w4attribute(natt1);
426     w4border(DOUBLE, natt1);
427     w4title(0, 2, " Help ", natt1);
428     w4activate(w_ref);
429
430     w4(startrow++, startcol, "You can use the menu to select an option or you can press");
431     w4(startrow++, startcol, "one of the following command keys:");
432     w4(startrow++, startcol, " ");
433     w4(startrow++, startcol, "A Add a blank record.");
434     w4(startrow++, startcol, "B Move to the bottom database record.");
435     w4(startrow++, startcol, "C Add a record by copying the current record.");
436     w4(startrow++, startcol, "D Mark the current record for deletion.");
437     w4(startrow++, startcol, "E Switch to the edit screen if possible.");
438     w4(startrow++, startcol, "F Find a record.");
439     w4(startrow++, startcol, "H Display this help screen.");
440     w4(startrow++, startcol, "M Modify the current record.");
441     w4(startrow++, startcol, "R Enter a record to move to.");
442     w4(startrow++, startcol, "S Select a record ordering index.");
443     w4(startrow++, startcol, "T Move to the top database record.");
444     w4(startrow++, startcol, "U Remove the deletion mark from the current record.");

```

```

445     w4(startrow++, startcol, "Z Switch to the browse screen if possible.");
446     w4(startrow++, startcol, "X Exit.");
447     w4(startrow++, startcol, "<PgUp> or <PgDn> Move up/down one screen of records.");
448     w4(startrow++, startcol, "<Up or Down Arrow> Move to the previous/next record.");
449     getch();
450     w4deactivate(w_ref);
451     w4close(w_ref);
452     return 0;
453 }
454
455 static int b4record(int junk_parm)
456 {
457     long rec;
458     int rc, w_ref;
459
460     rec = b4cursor_rec;
461     if (rec < 0L) rec = 0L;
462
463     w_ref = w4define(9, 20, 14, 58);
464     w4attribute((long) KW);
465     w4border(DOUBLE, (long) KW);
466     w4popup();
467     w4title(0, -1, " Record Number Command ", B_WHITE);
468     w4activate(w_ref);
469
470     w4(1, 2, "Enter Record Number: ");
471     g4attribute((long) WK);
472     g4long(w4row(), w4col(), & rec);
473     w4(2, 2, "Records in Database:");
474     w4long(w4row(), w4col(), d4reccount(), 9);
475     rc = g4read(); /* Read the Record */
476
477     w4deactivate(w_ref);
478     w4close(w_ref);
479
480     if (rc != ESC)
481     {
482         if (rec > 0 && rec <= d4reccount())
483         {
484             if (is_empty)
485             {
486                 w4display(" Message ",
487                     "To display a record, there will be an immediate",
488                     "switch to record number ordering.", (char *) 0);
489                 i4unselect();
490                 b4empty_check();
491             }
492             b4cursor_rec = rec;
493             b4cursor_row = 0;
494
495             b4deact_menu();
496             b4display_assume();
497         }
498     }
499
500     b4display_cursor();
501     d4unlock(-1L);
502     return 0;
503 }
504
505 static int b4find(int junk_parm)
506 {
507     int index_ref, rc, w_ref;
508     char seek_data[101];
509
510     index_ref = i4seek_ref();
511     if (index_ref < 0) return 0;
512
513     w_ref = w4define(9, 21, 16, 58);
514     w4attribute((long) KW);
515     w4popup();
516     w4border(DOUBLE, (long) KW);
517     w4title(0, -1, " Find ", (long) KW);
518     w4activate(w_ref);

```

```
519
520     w4(1, 3, "Find Index: ");
521     w4(1, w4col(), i4name(index_ref));
522
523     memset(seek_data, 0, (size_t) sizeof (seek_data));
524     w4(3, 3, "Enter the Find Information:");
525     g4attribute((long) WK);
526     g4(4, 3, seek_data);
527     if (i4type(index_ref) == 'N')
528         g4width(24, 24);
529     else
530         g4width(24, 100);
531     rc = g4read();
532
533     w4deactivate(w_ref);
534     w4close(w_ref);
535
536     if (rc == ESC)
537         b4display_cursor();
538     else
539     {
540         if (i4type(index_ref) == 'N')
541         {
542             if ((rc = d4seek_double(c4atod(seek_data, 24))) < 0) return - 1;
543         }
544         else
545             if ((rc = d4seek_str(seek_data)) < 0) return - 1;
546
547         if (rc == 3)
548         {
549             c4trim_n(seek_data, (int) sizeof (seek_data));
550             w_ref = w4define(6, 5, 10, (46 + strlen(seek_data) + 8));
551             w4popup();
552             w4border(DOUBLE, natt1);
553             w4attribute(natt1);
554             w4activate(w_ref);
555             w4(1, 1, "The end of the file was reached finding data: ");
556             w4(1, w4col(), seek_data);
557             g4char();
558             w4deactivate(w_ref);
559             w4close(w_ref);
560             b4display_cursor();
561
562             d4unlock(-1L);
563             return 0;
564         }
565
566         b4cursor_rec = d4recno();
567         b4cursor_row = 0;
568         b4deact_menu();
569         b4display_assume();
570     }
571
572     d4unlock(-1L);
573     return 0;
574 }
575
576
577 static int  b4select_index(int item_ref)
578 {
579     i4select(n4int_get(item_ref));
580     n4start_item(item_ref);
581     last_read_char = -2;
582     do_escape = 1;
583     return 0;
584 }
585
586 static int  b4delete(int junk_parm)
587 {
588     if (b4cursor_rec > 0L)
589         d4delete(b4cursor_rec);
590
591     b4deact_menu();
592     w4select(b4cur_ref);
```

```
593     b4display_one(b4cursor_row);
594     d4unlock(-1L);
595     return 0;
596 }
597
598 static int b4undelete(int junk_parm)
599 {
600     if (b4cursor_rec > 0L)
601         d4recall(b4cursor_rec);
602
603     b4deact_menu();
604     w4select(b4cur_ref);
605     b4display_one(b4cursor_row);
606     d4unlock(-1L);
607     return 0;
608 }
609
610 static int b4browse_flip(int junk_parm)
611 {
612     if (u4ptr_equal((void *) user_edit_setup, (void *) 0)) return 0;
613     if (u4ptr_equal((void *) user_browse_setup, (void *) 0)) return 0;
614
615     b4deact_menu();
616     n4refresh(b4pull_ref);
617
618     w4close(b4cur_ref);
619
620     if (b4on_browse)
621         b4edit_setup();
622     else
623         b4browse_setup();
624
625     return 0;
626 }
627
628 #ifdef KR
629     int b4edit( browse_setup, edit_setup )
630     int (*browse_setup)();
631     int (*edit_setup)();
632 #else
633     int b4edit(int (* browse_setup) (void), int (* edit_setup) (void))
634 #endif
635 {
636     b4on_browse = 0;
637     return (b4setup(browse_setup, edit_setup));
638 }
639
640 #ifdef KR
641     int b4browse( browse_setup, edit_setup )
642     int (*browse_setup)();
643     int (*edit_setup)();
644 #else
645     int b4browse(int (* browse_setup) (void), int (* edit_setup) (void))
646 #endif
647 {
648     b4on_browse = 1;
649     return (b4setup(browse_setup, edit_setup));
650 }
651
652 static char browse_or_edit[] = "BROWSE";
653
654 static void b4edit_setup()
655 {
656     b4cur_ref = (* user_edit_setup) ();
657     b4on_browse = 0;
658     b4cur_top_margin = v4get[v4window_ptr->first_get].row;
659     b4page_size = 1;
660     b4cursor_row = 0;
661     b4cursor_col = v4get[v4window_ptr->first_get].col;
662     strcpy(browse_or_edit, "Browse");
663     b4display_assume();
664     d4unlock(-1L);
665 }
666
```

```

667 static void b4browse_setup()
668 {
669     b4cur_ref = (* user_browse_setup) ();
670     b4on_browse = 1;
671     b4cur_top_margin = b4top_margin;
672     b4page_size = w4height(-1) - b4top_margin - b4bottom_margin;
673     b4cursor_row = 0;
674     b4cursor_col = v4get[v4window_ptr->first_get].col;
675     strcpy(browse_or_edit, "Edit ");
676     b4display_assume();
677     d4unlock(-1L);
678 }
679
680 #ifdef KR
681     static int b4setup( browse_setup, edit_setup )
682     int (*browse_setup) () ;
683     int (*edit_setup) () ;
684 #else
685     static int b4setup(int (* browse_setup) (void), int (* edit_setup) (void))
686 #endif
687 {
688     int srch_ref, pos_ref, add_ref, ch_ref, fillwin;
689     int i_ref, item_ref;
690     int start_lock_code;
691
692     /* w4clear(-1) ;*/
693
694     user_edit_setup = edit_setup;
695     user_browse_setup = browse_setup;
696     start_lock_code = d4lock_code(2);
697
698     b4empty_check();
699
700     if (b4on_browse)
701         b4browse_setup();
702     else
703         b4edit_setup();
704
705     b4select_ref = -1;
706
707     /* Define the index file selection menu. */
708     if (d4ptr()->index_ref >= 0)
709     {
710         i_ref = h4first((char **) & v4index, d4ptr()->index_ref);
711
712         b4select_ref = w4define(-1, -1, -1, -1);
713         w4attribute(natt1);
714         w4border(SINGLE, natt1);
715         n4attribute(natt1, natt2);
716         n4("Record Number Ordering");
717         n4action(b4select_index);
718         n4int_save(-1);
719         for (; i_ref >= 0; i_ref = v4index[i_ref].next)
720         {
721             item_ref = n4(v4index[i_ref].name);
722             if (i4seek_ref() == i_ref)
723                 n4start_item(item_ref);
724             n4action(b4select_index);
725             n4int_save(i_ref);
726         }
727     }
728
729     /* Define the main pulldown menu. */
730     n4key_special(-1, CTRL_C, -1, -1);
731     b4pull_ref = w4define(0, 0, 0, 79);
732     w4attribute(matt1);
733     n4attribute(matt1, matt2);
734     n4key_special(ESC, CTRL_C, -1, -1);
735
736     n4(" Help ");
737     n4key(0, 0, -1);
738     n4action(b4help);
739
740     n4(" Modify ");

```

```
741     n4key(0, 0, -1);
742     n4action(n4sub_menu);
743     n4ptr_save(& ch_ref);
744
745     n4(" Add ");
746     n4key(0, 0, -1);
747     n4action(n4sub_menu);
748     n4ptr_save(& add_ref);
749
750     n4(" Position ");
751     n4key(0, 0, -1);
752     n4action(n4sub_menu);
753     n4ptr_save(& pos_ref);
754
755     if (b4select_ref >= 0)
756     {
757         n4(" Find ");
758         n4key(0, 0, -1);
759         n4action(n4sub_menu);
760         n4ptr_save(& srch_ref);
761     }
762
763     if (! u4ptr_equal((void *) user_edit_setup, (void *) 0) &&
764         ! u4ptr_equal((void *) user_edit_setup, (void *) 0))
765     {
766         n4(browse_or_edit);
767         n4key(0, 0, -1);
768         n4action(b4browse_flip);
769     }
770
771     n4(" Exit ");
772     n4key(0, 0, -1);
773     n4parm(-1);
774
775     ch_ref = w4define(-1, -1, -1, -1);
776     w4attribute(natt1);
777     w4border(SINGLE, natt1);
778     n4attribute(natt1, natt2);
779     n4(" Modify Record "); n4action(b4modify); n4parm(1);
780     n4(" Delete Record "); n4action(b4delete);
781     n4(" Undelete Record "); n4action(b4undelete);
782
783     if (b4select_ref >= 0)
784     {
785         srch_ref = w4define(-1, -1, -1, -1);
786         w4attribute(natt1);
787         w4border(SINGLE, natt1);
788         n4attribute(natt1, natt2);
789         n4(" Find "); n4action(b4find);
790         if (b4select_ref >= 0)
791         {
792             n4(" Select Record Ordering Index ");
793             n4action(n4sub_menu);
794             n4ptr_save(& b4select_ref);
795         }
796     }
797
798     pos_ref = w4define(-1, -1, -1, -1);
799     w4attribute(natt1);
800     w4border(SINGLE, natt1);
801     n4attribute(natt1, natt2);
802     n4(" Record "); n4action(b4record);
803     n4(" Top "); n4action(b4top);
804     n4(" Bottom "); n4action(b4bottom);
805
806     add_ref = w4define(-1, -1, -1, -1);
807     w4attribute(natt1);
808     w4border(SINGLE, natt1);
809     n4attribute(natt1, natt2);
810     n4(" Add Blank "); n4action(b4add_copy); n4parm(1);
811     n4(" Add Copy "); n4action(b4add_copy); n4parm(0);
812     n4key((int) 'C', 1, 4);
813
814     n4pulldown(b4pull_ref);
```

```
815     w4select(b4pull_ref);
816     w4memory();
817     fillwin = w4define(0, 0, 0, 79);
818     w4attribute(matt1);
819     w4activate(fillwin);
820
821     n4char_routine(b4char);
822     n4activate(b4pull_ref);
823
824     w4close(b4cur_ref);
825
826     w4deactivate(b4pull_ref);
827     w4close(b4pull_ref);
828     w4close(fillwin);
829     n4char_routine(0);
830
831     if (b4select_ref >= 0)
832     {
833         w4close(b4select_ref);
834         w4close(srch_ref);
835     }
836
837     w4close(pos_ref);
838     w4close(add_ref);
839     w4close(ch_ref);
840
841     d4lock_code(start_lock_code);
842     w4clear(-1);
843
844     return 0;
845 }
846
847 static int  b4char()
848 {
849     int  rc;
850
851     if (do_escape)
852     {
853         do_escape = 0;
854         return ESC;
855     }
856
857     if (last_read_char != 0)
858     {
859         rc = last_read_char;
860         last_read_char = 0;
861     }
862     else
863         rc = g4char();
864
865     if (rc > 0 && rc < 0xFF)
866         rc = u4toupper(rc);
867
868     if (w4select(-1) != b4pull_ref)
869         if (rc >= (int) 'A' && rc <= (int) 'Z') return rc;
870
871     switch (rc)
872     {
873     case PGUP :
874         b4page(- b4page_size);
875         break;
876
877     case PGDN :
878         b4page(b4page_size);
879         break;
880
881         #ifndef UNIX
882         case ALT_A :
883             #endif
884         case 'A' :
885             b4add_copy(1);
886             break;
887
888         #ifndef UNIX
```

```
889     case ALT_B :
890     #endif
891     case 'B' :
892     b4bottom(0);
893     break;
894
895     #ifndef UNIX
896     case ALT_C :
897     #endif
898     case 'C' :
899     b4add_copy(0);
900     break;
901
902     #ifndef UNIX
903     case ALT_D :
904     #endif
905     case 'D' :
906     b4delete(0);
907     break;
908
909     #ifndef UNIX
910     case ALT_E :
911     #endif
912     case 'E' :
913     if (b4on_browse)
914         b4browse_flip(0);
915     break;
916
917     #ifndef UNIX
918     case ALT_F :
919     #endif
920     case 'F' :
921     b4find(0);
922     break;
923
924     #ifndef UNIX
925     case ALT_H :
926     #endif
927     case 'H' :
928     b4help(0);
929     break;
930
931     #ifndef UNIX
932     case ALT_M :
933     #endif
934     case 'M' :
935     b4modify(1);
936     break;
937
938     #ifndef UNIX
939     case ALT_R :
940     #endif
941     case 'R' :
942     b4record(0);
943     break;
944
945     #ifndef UNIX
946     case ALT_S :
947     #endif
948     case 'S' :
949     if (b4select_ref >= 0)
950         n4activate(b4select_ref);
951     break;
952
953     case - 2 :                                /* After 'b4select_index' to refresh the display.*/
954     b4deact_menu();
955     b4go_rec();
956     b4empty_check();
957     b4display_assume();
958     if (b4cursor_rec == -1L)
959     {
960         b4cursor_rec = b4start_rec;
961         b4cursor_row = 0;
962     }
```

```

963         d4unlock(~1L);
964         break;
965
966         #ifndef UNIX
967         case ALT_T :
968         #endif
969     case 'T' :
970         b4top(0);
971         break;
972
973         #ifndef UNIX
974         case ALT_U :
975         #endif
976     case 'U' :
977         b4undelete(0);
978         break;
979
980         #ifndef UNIX
981         case ALT_Z :
982         #endif
983     case 'Z' :
984         if (! b4on_browse)
985             b4browse_flip(0);
986         break;
987
988         #ifndef UNIX
989         case ALT_X :
990         #endif
991     case 'X' :
992         return CTRL_C;
993
994     case UP :
995         if (v4window_ptr->horizontal)
996             b4skip_one(-1);
997         return rc;
998
999     case DOWN :
1000         if (v4window_ptr->horizontal)
1001             b4skip_one(1);
1002         return rc;
1003
1004     default :
1005         return rc;
1006     }
1007
1008     return 0;
1009 }
1010
1011 int b4quick_browse()
1012 {
1013     int c, j, i, next_c, w, fillwin;
1014     long ref;
1015     int ur = 1, lc = 0, lr = 24, rc = 79;
1016     char vl[2] =
1017     {
1018         179, 0
1019     };
1020     char vt[2] =
1021     {
1022         194, 0
1023     };
1024
1025     w4define(ur, lc, lr, rc);
1026     w4attribute(KC);
1027     w4memory();
1028
1029     w4activate(-1);
1030     g4release(0);
1031
1032     next_c = 2;
1033
1034     w4repeat(2, lc + 2, 196, (rc - lc - 3));
1035     w4(0, 2, d4name());

```

```
1031     for (j = 1; j <= f4num_fields(); j++)
1032     {
1033         c = next_c;
1034
1035         ref = f4j_ref(j);
1036         if (f4type(ref) == 'M') continue;
1037         if (f4width(ref) >= MAX_GET_WIDTH) continue;
1038
1039         w = f4width(ref);
1040         if (w <= 10) w = 10;
1041
1042         if (c > 2 && c < (rc - w))
1043         {
1044             w4(ur + 1, c - 2, vt);
1045             for (i = ur + 2; i < lr - 1; i++) w4(i, c - 2, vl);
1046         }
1047
1048         next_c = c + w + 2;
1049         if (next_c >= w4width(-1)) break;
1050
1051         w4(1, c, f4name(ref));
1052         g4field(-1, c, ref);
1053     }
1054     return (w4select(-1));
1055 }
1056
1057 int b4quick_edit()
1058 {
1059     int r, j;
1060     long ref;
1061
1062     w4define(1, 0, 24, 79);
1063     w4attribute(KC);
1064     w4(0, 1, d4name());
1065
1066     w4memory();
1067     w4activate(-1);
1068     g4release(0);
1069
1070     r = 1;
1071     for (j = 1; j <= f4num_fields(); j++)
1072     {
1073         ref = f4j_ref(j);
1074         if (f4type(ref) == 'M') continue;
1075         if (f4width(ref) >= MAX_GET_WIDTH) continue;
1076
1077         if (r >= w4height(-1) - 1) break;
1078         w4(r, 2, f4name(ref));
1079         g4field(r, 14, ref);
1080         if (f4width(ref) > 64) g4width(f4width(ref), 64);
1081         r++;
1082     }
1083     return (w4select(-1));
1084 }
1085
1086 }
1087 }
```

```

1  /* u4error.c   modified for RMRS and DATA WINDOWS
2
3  MDS 11-21-91
4  see original code base version of u4error.c to port to UNIX
5
6  */
7
8  #include <stdarg.h>
9  #include "rmrserr.h"
10 #include "p4misc.h"
11 #include "dw.h"
12 #include "d4all.h"
13 #include "rmrs.h"
14
15 HWND errwin;
16
17 int v4error = -1;
18
19 typedef struct error_data_st
20 {
21     int    error_num;
22     char * error_data;
23 }
24     ERROR_DATA;
25 ERROR_DATA error_data[] =
26 {
27     /* General Disk Access Errors */
28     {
29         E_CREATE, "Creating File"
30     },
31     {
32         E_OPEN, "Opening File"
33     },
34     {
35         E_READ, "Reading from File"
36     },
37     {
38         E_LSEEK, "Seeking to File Position"
39     },
40     {
41         E_WRITE, "Writing to File"
42     },
43     {
44         E_CLOSE, "Closing File"
45     },
46     {
47         E_REMOVE, "Removing File"
48     },
49     /* Database Specific Errors */
50     {
51         E_BAD_DBF, "File is not a Database:"
52     },
53     {
54         E_D_MISSING, "No Open Database"
55     },
56     {
57         E_REC_LENGTH, "Record Length is Too Large"
58     },
59     {
60         E_FIELD, "Unrecognized Field"
61     },
62     /* Index File Specific Errors */
63     {
64         E_INDEX, "Building Index File"
65     },
66     {
67         E_I_CLOSE, "Closing Index File"
68     },
69     {
70         E_BAD_NDX, "File is not an Index File"
71     },
72 }

```

```
46      {
46          E_I_DATE, "Index File is out of Date"
46      },
47      {
47          E_I_RECORD, "Index File Record does not Exist"
47      },
48      {
48          E_UNIQUE, "Key is not Unique"
48      },
49      {
49          E_I_TYPE, "Key Evaluates to Logical Result"
49      },
50      {
50          E_I_CHANGED, "Key Length or Type has Changed"
50      },
51      {
51          E_KEY_LEN, "Key Length over 100 Characters"
51      },
52      {
52          E_NO_INDEX, "Seek on Database with no Index File"
52      },
53      {
53          E_NUM_PARMS, "Wrong Number of Parameters in Expression"
53      },
54      /* Multi-User Errors */
55      {
56          E_LOCK, "Locking a File"
56      },
57      {
57          E_UNLOCK, "Unlocking a File"
57      },
58      /* Expression Evaluation Errors */
59      {
60          E_BASE_NAME, "Database not Located while Evaluating Expression"
60      },
61      {
61          E_COMPILE_NULL, "Executing Null Expression"
61      },
62      {
62          E_EXPECT, "Expecting \",\" or \")\" while Evaluating Expression"
62      },
63      {
63          E_COMPLETE, "Expression is not Complete"
63      },
64      {
64          E_DATE, "Illegal Date"
64      },
65      {
65          E_OVERFLOW, "Overflow while Evaluating Expression"
65      },
66      {
66          E_TYPE, "Parameter or Operator has the Wrong Type"
66      },
67      {
67          E_RIGHT, "Right Bracket Missing in Expression"
67      },
68      {
68          E_FUNCTION, "Unrecognized Function in Expression"
68      },
69      {
69          E_OPERATOR, "Unrecognized Operator in Expression"
69      },
70      {
70          E_VALUE, "Unrecognized Value in Expression"
70      },
71      {
71          E_STRING_LONG, "Unterminated String in Expression"
71      },
72      /* Memo File Errors */
73
```

```

74      {
74          E_EDITOR, "Editing Memo File with Editor"
74      }
74      ,
75      {
75          E_MEMO_NAME, "Memo File Name Inconsistency"
75      }
75      ,
76      {
76          E_MEMO_SIZE, "Memo File Entry is over 32767 Bytes"
76      },
77
78      /* Windowing and Menuing Errors */
79      {
79          E_WINDOW_REF, "Illegal Window Reference Number"
79      },
80
81      /* Extended Routine Errors */
82      {
82          E_RELATING, "Relating Databases"
82      }
82      ,
83      {
83          E_CONTROL, "No Controlling Database"
83      }
83      ,
84      {
84          E_RELATED, "Illegal Related Database"
84      }
84      ,
85
86      /* Memory Error */
87      {
87          E_MEMORY, "Out of Memory"
87      }
87      ,
88      {
88          E_ALLOCATE, "Memory Allocation Error"
88      }
88      ,
89
90      /* Internal Error */
91      {
91          E_INTERNAL, "Overwritten Memory"
91      }
91      ,
92
93      /* Sorting Errors */
94      {
94          E_SORT, "Not Enough Memory to Sort"
94      }
94      ,
95      {
95          E_SORT_ADD, "Too Many Records in Sort"
95      },
96      };
97
98 static int col, row;
99
100 void error_out(char *);
101 void error_out(char * ptr)
102 {
103     row++;
104     vatputs(errwin, row, col, ptr);
105 }
106
107 void u4error(int error_num, char * msg, ...)
108 {
109     int keyboard, i;
110     va_list arg_marker;
111     char * ptr;
112     char buffer[40];
113     row = 0;
114     col = 2;

```

```
115
116     errwin = vcreat(15, 40, ERR, YES);
117     vlocate(errwin, 5, 20);
118     vframe(errwin, ERR, FRDOUBLE);
119     vshadow(errwin, CURRENT, SHADOW75, BOTTOMRIGHT);
120     visible(errwin, YES, YES);
121     vmtitle(errwin, _TOP, CENTERJUST, ERR, " ERROR ");
122
123     v4error = error_num;
124
125     error_out("Error Number: ");
126     c4ltoa((long) error_num, buffer, 10);
127     buffer[10] = '\000';
128     error_out(buffer);
129     error_out("");
130
131     for (i = 0; i < sizeof (error_data) / sizeof (ERROR_DATA); i++)
132         if (error_data[i].error_num == error_num)
133             {
134                 error_out(error_data[i].error_data);
135                 break;
136             }
137
138     va_start(arg_marker, msg);
139     ptr = msg;
140
141     for (; ptr != (char *) 0; ptr = va_arg(arg_marker, char *))
142     {
143         error_out(ptr);
144     }
145
146     error_out("Press a key ...");
147     keyboard = getch();
148
149     if (error_num == E_MEMORY || error_num == E_INTERNAL || error_num == E_ALLOCATE)
150     {
151         vdelete(errwin, NONE);
152         exit(1);
153     }
154
155     vdelete(errwin, NONE);
156     return (keyboard);
157 }
```

```

1 /*
2  *  rmrsinit.c
3  *
4  */
5
6 #define SYSTDSCRP "System commands and description"
7 #define SABTDSCRP "Show version "
8 #define SDOSDSCRP "Exit to DOS temporarily (typing EXIT will return to RMRS)"
9 #define SQUTDSCRP "Quit the RMRS program "
10 #define INITDSCRP "Change setup or initialize new AB supply files"
11 #define PROCDSCRP "Check repairs for resource availability conflicts"
12 #define SCHDDSCRP "Schedule the repairs that are POSSIBLE"
13 #define CHNGDSCRP "Change the status of a repair"
14 #define GREPDSCRP "Generate a report"
15 #define QUITDSCRP "Exit program"
16 #define ILNSDSCRP "Load a new AB resource supply file from disk"
17 #define ILEQDSCRP "Load a new AB equipment supply file from disk"
18 #define IIMTDSCRP "Load a new AB materials supply file from disk"
19 #define PAUTDSCRP "Process ALL new PDES facility repair files in priority order"
20 #define PSELDSCRP "Select a single PDES facility repair file for processing"
21 #define PCOMDSCRP "Select a SUSPENDED repair and reattempt a resource compromise"
22 #define SFULDSCRP "Schedule all repairs (i.e. new AND previously scheduled repairs)"
23 #define SINCDSCRP "Schedule only newly processed repairs"
24 #define SPRIDSCRP "Set the scheduling order by adjusting repair priority"
25 #define CCOMDSCRP "Change the status of a QUEUED repair to COMPLETED"
26 #define CCANDSCRP "Change the status of a POSSIBLE or QUEUED repair to CANCELED"
27 #define GGANDSCRP "Generate a gantt chart"
28 #define GSCHDSCRP "Generate a tabular schedule listing"
29 #define ED "Edit RMRS data files"
30
31 #include <stdio.h>
32 #include <string.h>
33 #include "color.h"
34 #include "dw.h"
35 #include "dwmenu.h"
36 #include "dwsystem.h"
37
38 #include "itemfns.h"
39 #include "rmrs.h"
40
41 HWND * mmain;
42
43 void globhelp(MENUITEM * item);
44 void IntroScreen(void);
45 void SetupMenu(void);
46
47 void SetupMenu(void)
48 {
49     HWND FlHelpWin;
50
51     MENUITEM * System;
52     MENUITEM * SystAbot, * SystXDOS, * SystCOMM, * SystQuit;
53
54     MENUITEM * EdFiles;
55     MENUITEM * EdFlsRep, * EdFlsEqSup, * EdFlsEqReq, * EdFlsMaSup, * EdFlsMaReq;
56     MENUITEM * EdFlsReset, * EdFlsRepInfo, * EdFlsFacPrty;
57
58     MENUITEM * InitSetup;
59     MENUITEM * InitNwFl, * InitMatl, * InitEqui;
60
61     MENUITEM * Process;
62     MENUITEM * ProcAuto, * ProcFacI, * ProcComp;
63
64     MENUITEM * Schedule;
65     MENUITEM * SchdFull, * SchdInc, * SchdPrior;
66
67     MENUITEM * ChangeStatus;
68     MENUITEM * ChngComp, * ChngCanc;
69
70     MENUITEM * GenerateReport;
71     MENUITEM * GnRpSum, * GnRpDet, * GnRpGant;
72
73     /*** set up help micro-window "F1 - HELP" ***/
74     FlHelpWin = vcreat(1, 80, EMPHNORML, NO);

```

```

75  vlocate(FlHelpWin, 24, 0);
76  vratputs(FlHelpWin, 0, 0, RED_ON_WHITE, " F1 ");
77  vatputs(FlHelpWin, 0, 4, "Help 3 ");
78  visible(FlHelpWin, YES, NO);
79
80  /** define main menu **/
81  mmain = MNUCreateHdr(PULLDOWN);
82  MNUSetGlobalHelp(mmain, globhelp);
83  MNUSetShadow(mmain, SHADOW75, BOTTOMRIGHT);
84  MNUShowHotKey(mmain, YES);
85  MNUSetDescPosition(mmain, 24, 11);
86  MNUSetDownIndicator(mmain, NO);
87  MNUSetSpaces(mmain, 2);
88  MNUSetDropDown(mmain, YES);
89  MNUSetAttributes(mmain, EMPHNORML, EMPHNORML, EMPHNORML, REVNORML/* HELP*/, HIGHNORML,
90  EMPHNORML, REVERR, EMPHNORML);
91
92  /** define menu bar items **/
93  System = MNUAddItem(" System ", SYSTDSCR, 'Y', ALTY, NULL, mmain, NULLF);
94  InitSetup = MNUAddItem(" Init/Setup ", INITDSCR, 'I', ALTI, NULL, mmain, NULLF);
95  Process = MNUAddItem(" Process ", PROCDSCR, 'P', ALTP, NULL, mmain, NULLF);
96  Schedule = MNUAddItem(" Schedule ", SCHDDSCR, 'S', ALTS, NULL, mmain, NULLF);
97  ChangeStatus = MNUAddItem(" Change Status ", CHNGDSCR, 'C', ALTC, NULL, mmain, NULLF);
98  GenerateReport = MNUAddItem(" Gen Report ", GREPDSCR, 'R', ALTR, NULL, mmain, NULLF);
99
100 /** set up pull down menus **/
101 SystAbot = MNUAddItem("About", SABTDSCR, ' ', 0, System, mmain, SystAbotFn);
102
103 EdFiles = MNUAddItem("Edit Data", ED, ' ', ALTE, System, mmain, NULLF);
104 EdFlsRep = MNUAddItem("Repair File", ED, ' ', 0, EdFiles, mmain, EdFlsRepFn);
105 EdFlsEqSup = MNUAddItem("Equip Supply", ED, ' ', 0, EdFiles, mmain, EdFlsEqSupFn);
106 EdFlsEqReq = MNUAddItem("Equip Required", ED, ' ', 0, EdFiles, mmain, EdFlsEqReqFn);
107 EdFlsMaSup = MNUAddItem("Material Supply", ED, ' ', 0, EdFiles, mmain, EdFlsMaSupFn);
108 EdFlsMaReq = MNUAddItem("Material Required", ED, ' ', 0, EdFiles, mmain, EdFlsMaReqFn);
109 EdFlsRepInfo = MNUAddItem("Repair Info", ED, ' ', 0, EdFiles, mmain, EdFlsRepInfoFn);
110 EdFlsFacPrty = MNUAddItem("Facility Priority", ED, ' ', 0, EdFiles, mmain, EdFlsFacPrtyFn);
111 EdFlsReset = MNUAddItem("Reset All Data", ED, ' ', 0, EdFiles, mmain, EdFlsResetFn);
112
113 SystXDOS = MNUAddItem("DOS shell", SDOSDSCR, ' ', ALTD, System, mmain, SystXDOSFn);
114
115 MNUSetSeparatorBefore(SystXDOS);
116
117 SystQuit = MNUAddItem("Quit", SQUTDSCR, ' ', ALTQ, System, mmain, SystQuitFn);
118
119 InitNwFi = MNUAddItem("Load new AB supply file", ILNSDSCR, ' ', 0, InitSetup, mmain, NULLF
);
120 InitMatl = MNUAddItem("AB materials supply file", ILMTDSCR, ' ', 0, InitNwFi, mmain, InitM
atlFn);
121 InitEqui = MNUAddItem("AB equipment supply file", ILEQDSCR, ' ', 0, InitNwFi, mmain, InitE
quiFn);
122
123 ProcAuto = MNUAddItem("Auto Mode", PAUTDSCR, ' ', 0, Process, mmain, ProcAutoFn);
124 ProcFacI = MNUAddItem("Select facility", PSELDSCR, ' ', 0, Process, mmain, ProcFacIFn);
125 ProcComp = MNUAddItem("Compromise Retry", PCOMDSCR, ' ', 0, Process, mmain, ProcCompFn);
126
127 SchdFull = MNUAddItem("Full Schedule", SFULDSCR, ' ', 0, Schedule, mmain, SchdFullFn);
128 SchdInc = MNUAddItem("Update Schedule", SINCSCR, ' ', 0, Schedule, mmain, SchdIncFn);
129 SchdPrior = MNUAddItem("Set Priority", SPRIDSCR, ' ', 0, Schedule, mmain, SchdPriorFn);
130
131 ChngComp = MNUAddItem("Completed", CCOMDSCR, ' ', 0, ChangeStatus, mmain, ChngCompFn);
132 ChngCanc = MNUAddItem("Canceled", CCANDSCR, ' ', 0, ChangeStatus, mmain, ChngCancFn);
133
134 GnRpSum = MNUAddItem("Summary Schedule", GSCHDSCR, ' ', 0, GenerateReport, mmain, GnRpSchd
Fn);
135 GnRpDet = MNUAddItem("Detailed Schedule", GSCHDSCR, ' ', 0, GenerateReport, mmain, GnRpSch
dFn);
136 GnRpGant = MNUAddItem("Gantt Chart", GGANDSCR, ' ', 0, GenerateReport, mmain, GnRpGantFn);
137
138 }
139
140 #define HELPFILNAME "RMRS.HLP"
141 #define HWLR 25
142 #define HWLC 80
143 #define HWPR 7

```

```

144 #define HWPC 40
145 #define HWROW 9
146 #define HWCOL 20
147
148 void globhelp(MENUITEM * item)
149 {
150     HWND hwinptr;
151     char * helpstr;
152
153     /* DEFINE HELP WINDOW */
154     hwinptr = vcreat(HWLR, HWLC, HELP, NO);
155     vwind(hwinptr, HWPR, HWPC, 0, 0);
156     vlocate(hwinptr, HWROW, HWCOL);
157     vframe(hwinptr, REVHIGHHELP, FRSSINGLE);
158     vmtitle(hwinptr, TOP, CENTERJUST, REVHIGHHELP, " RMRS Help (ESC to quit) ");
159     visible(hwinptr, YES, YES);
160
161     helpstr = ExtractFromHelpFile(HELPPFILENAME,
162     (item->strtoggle ? item->item1 : item->item0));
163     if (helpstr)
164     {
165         vdispstr(hwinptr, helpstr);
166         scroll_win_keys(hwinptr, HWLR, HWLC, HWPR, HWPC);
167     }
168     else
169     {
170         /* NO HELP FILE */
171         vatputs(hwinptr, 2, 1, "<Cannot find help file '");
172         vputs(hwinptr, HELPPFILENAME);
173         vputs(hwinptr, ">");
174         getch();
175     }
176
177     /* DELETE HELP WINDOW */
178     vdelete(hwinptr, NONE);
179 }
180
181 void IntroScreen(void)
182 {
183     HWND IntroScreen;
184
185     /** set background */
186     pclrchar(SHADOW25);
187
188     /** display intro screen */
189     IntroScreen = vcreat(17, 70, REVNORML, YES);
190     vlocate(IntroScreen, 3, 5);
191     vshadow(IntroScreen, NULL, SHADOW75, BOTTOMRIGHT);
192     vframe(IntroScreen, REVNORML, FRDOUBLE);
193     vatputas(IntroScreen, 1, 8, HELP, " Automated RESOURCE MANAGER / REPAIR SCHEDULER (RMRS)");
194
195     vatputs(IntroScreen, 2, 9, "Automated RESOURCE MANACER / REPAIR SCHEDULER (RMRS)");
196     vatputas(IntroScreen, 3, 8, HELP, " Automated RESOURCE MANAGER / REPAIR SCHEDULER (RMRS)");
197
198     vatputs(IntroScreen, 5, 9, " for the ");
199     vatputs(IntroScreen, 7, 9, " Post-Attack Facility Damage Assessment System ");
200     vatputs(IntroScreen, 9, 9, " <<< POST - DAM >>> ");
201     vatputs(IntroScreen, 10, 28, "Version ");
202     vputs(IntroScreen, VERSION);
203     vputs(IntroScreen, ".");
204     vputs(IntroScreen, REVISION);
205     vatputs(IntroScreen, 13, 9, " AFCESA/RACS Tyndall AFB, FL");
206     vatputs(IntroScreen, 14, 9, " Applied Research Associates, Inc.");
207     vatputs(IntroScreen, 16, 9, " Hit any key to continue ... ");
208     visible(IntroScreen, YES, YES);
209
210     /** wait for keystroke and return */
211     getch();
212     vdelete(IntroScreen, NONE);
213     return;
214 }

```

```

1 #include <stdio.h>
2 #include <math.h>
3 #include <conio.h>
4 #include <stdlib.h>
5 #include <dir.h>
6 #include <time.h>
7 #include <string.h>
8 #include <alloc.h>
9 #include "d4base.h"
10 #include "dw.h"
11 #include "dwmenu.h"
12 #include "dwsystem.h"
13 #include "rmrsutil.h"
14 #include "rmrs.h"
15
16 #define BUFFLENGTH 300
17
18 int ReprtSched(void)
19 {
20     int status, elenum, cnt = 0;
21     int r = 0, c = 1, key, lwinr, lwinc, pwinr, pwinc;
22     long repid, facnum;
23     time_t start, dur, end, currttime;
24     char eledescstr[53], dammodestr[53], repstgystr[53], schedstatus, * timeptr;
25     char facfuncstr[53];
26     HWND win;
27
28     lwinr = 200;
29     lwinc = 78;
30     pwinr = 21;
31     pwinc = 78;
32     /* WINDOW STUFF */
33     win = vcreat(lwinr, lwinc, bldattr(LIGHTBLUE, 0), YES);
34     if (win < 1) u4error(-1, "Error creating window", (char *) 0);
35     vwind(win, pwinr, pwinc, 0, 0);
36     vlocate(win, 2, 1);
37     vframe(win, REVNORML, FRDOUBLE);
38     vmtitle(win, _TOP, CENTERJUST, REVNORML, " Report - Repair Schedule ");
39
40     /* OPEN DBF AND INDEX FILES */
41     if (open_eqpreq(1))
42     {
43         u4error(-1, "BAD RETURN CODE", "OPEN_EQPREQ", (char *) 0);
44         vdelete(win, NONE);
45         return (-1);
46     }
47     if (open_repinf(1))
48     {
49         u4error(-2, "BAD RETURN CODE", "OPEN_REPINF", (char *) 0);
50         vdelete(win, NONE);
51         return (-2);
52     }
53     if (open_repair(1))
54     {
55         u4error(-3, "BAD RETURN CODE", "OPEN_REPAIR", (char *) 0);
56         vdelete(win, NONE);
57         return (-3);
58     }
59
60     d4select(eqpreq_dbf);
61     i4select(eqpreq_ndx2);
62     status = x4seek_str("REPAIR TEAM");
63     switch (status)
64     {
65     case (0) :
66     case (1) :
67         while (! d4eof()) && (! is_repteam()))
68         {
69             switch (f4char(f4ref("STATUS")))
70             {
71             case ('A') :
72             case ('L') :
73                 repid = f4long(f4ref("REPID"));
74                 start = f4long(f4ref("START"));

```

```

75     dur = f4long(f4ref("DURATION"));
76     schedstatus = f4char(f4ref("STATUS"));
77     end = (start + dur);
78
79     d4select(repair_dbf);
80     i4select(repair_ndx1);
81     x4seek_double(repid);
82     facnum = f4long(f4ref("FACNUM"));
83
84     d4select(repinfo_dbf);
85     i4select(repinfo_ndx1);
86     x4seek_double(repid);
87     elenum = f4int(f4ref("ELENUM"));
88     strcpy(eledescstr, f4str(f4ref("ELEDESC")));
89     strcpy(facfuncstr, f4str(f4ref("FACFUNC")));
90     strcpy(dammodestr, f4str(f4ref("DAMMODE")));
91     strcpy(repstgystr, f4str(f4ref("REPSTGY")));
92
93     if ((cnt == 0))
94     {
95         vatputs(win, r++, c, "Schedule Listing");
96         currttime = time(NULL);
97         timeptr = ctime(& currttime);
98         timeptr[strlen(timeptr) - 1] = '\0';
99         vatprintf(win, r++, c, "Generated : %s", timeptr);
100        vatprintf(win, r++, c, "-----");
101    }
102    d4select(eqpreq_dbf);
103    r++;
104    vatprintf(win, r++, c, "Repair ID      : %-19d  ", repid);
105    timeptr = ctime(& start);
106    timeptr[strlen(timeptr) - 1] = '\0';
107    vatprintf(win, r++, c, "START TIME      : %s", timeptr);
108    timeptr = ctime(& end);
109    timeptr[strlen(timeptr) - 1] = '\0';
110    vatprintf(win, r++, c, "END TIME        : %s", timeptr);
111    vatprintf(win, r++, c, "Facility Num    : B%-ld", facnum);
112    vatprintf(win, r++, c, "Facility Func   : %-s", facfuncstr);
113    vatprintf(win, r++, c, "Element #       : %d", elenum);
114    vatprintf(win, r, c, "elem desc      : %-19s", eledescstr);
115    vatprintf(win, r++, c + 60, "SCHD STATUS: %c", schedstatus);
116    vatprintf(win, r++, c, "damage mode    : %s", dammodestr);
117    vatprintf(win, r++, c, "rep strategy   : %s", repstgystr);
118    cnt++;
119    break;
120    default :; /* do nothing */
121    }
122    d4select(eqpreq_dbf);
123    d4skip(1);
124
125    }
126    d4close_all();
127    visible(win, YES, YES);
128    scroll_win_keys(win, lwinc, lwinc, pwinc, pwinc);
129    cnt = vfputwin(win, "SCHED.OUT", 1);
130    if (cnt != win) u4error(cnt, "DID NOT WRITE FILE", (char *) 0);
131    vdelete(win, NONE);
132    break;
133    case (2) :
134    case (3) :
135    default :; /* no repairs in EQPREQ.DBF */
136        u4error(0, "REPORT GENERATION ERROR", (char *) 0);
137    }
138    return (1);
139 }
140
141 int is_repteam(void)
142 {
143
144     if (strstr(f4str(f4ref("EQPDESC")), "repair team") != NULL)
145         return (1);
146     else
147         return (0);

```

```

148 )
149
150 /* ReprtGant
151 *
152 *
153 */
154 int ReprtGantt(void)
155 {
156     int done, i, thisone, r = 0, c = 1;
157     char status, linebuff[BUFFLENGTH], curdesc[26], * tmpdesc, * timeptr;
158     long start, duration, curepid, minstart, delta = 900L, curtime, maxend;
159     long cureqid, eqpidfld, eqpdescfld, eqpid;
160     long statusfld, startfld, durationfld, repidfld, repid;
161     HWND win;
162     unsigned int key = 0;
163     int lwinr, lwinc, pwinr, pwinc;
164
165     /* WINDOW STUFF */
166     lwinr = 100; /* logical window rows */
167     lwinc = 300; /* logical window columns */
168     pwinr = 21; /* physical window rows */
169     pwinc = 78; /* physical window columns */
170     win = vcreat(lwinr, lwinc, bldattr(LIGHTBLUE, 0), YES);
171     vwind(win, pwinr, pwinc, 0, 0);
172     vlocate(win, 2, 1);
173     vframe(win, REVNORML, FRDOUBLE);
174     vmtitle(win, _TOP, CENTERJUST, REVNORML, " Report - Gantt Chart ");
175
176     /* OPEN DBF AND INDEX FILES */
177     if (open_eqpreq(1))
178     {
179         u4error(-1, "BAD RETURN CODE", "OPEN_EQPREQ", (char *) 0);
180         vdelete(win, NONE);
181         return (-1);
182     }
183     i4select(eqpreq_ndx4); /* key = START */
184
185     statusfld = f4ref("STATUS");
186     startfld = f4ref("START");
187     durationfld = f4ref("DURATION");
188     repidfld = f4ref("REPID");
189     eqpidfld = f4ref("EQPID");
190     eqpdescfld = f4ref("EQPDESC");
191
192     /* GET MINSTART & MAXEND */
193     minstart = 2147000000;
194     maxend = 0L;
195     x4top();
196     while (! d4eof())
197     {
198         status = f4char(statusfld);
199         start = f4long(startfld);
200         duration = f4long(durationfld);
201         if ((status == 'A' || status == 'L') && start < minstart)
202             minstart = start;
203         if ((status == 'A' || status == 'L') && start + duration > maxend)
204             maxend = start + duration;
205         x4skip(1L);
206     }
207
208     /* BUILD SCHEDULE */
209     curtime = time(NULL);
210     timeptr = ctime(& curtime);
211     timeptr[strlen(timeptr) - 1] = '\0';
212     vatputf(win, r++, c, "Schedule as of :%s", timeptr);
213     vatputf(win, r++, c, "Each character represents %ld minutes", delta / 60);
214     vatputf(win, r++, c, "Total repair time is %.2f hours", ((float) (maxend - minstart)) / 360
0.);
215     vatputf(win, r++, c, "Repair Schedule");
216     vatputf(win, r++, c, "%20s|", " ");
217     x4top();
218     while (! d4eof())
219     {
220         for (i = 0; i < 299; i++) linebuff[i] = 32;

```

```

221     linebuff[299] = '\0';
222     status = f4char(statusfld);
223     while (! (status == 'A' || status == 'L'))
224     {
225         x4skip(1L);
226         status = f4char(statusfld);
227     }
228     curepid = f4long(repidfld);
229     start = f4long(startfld);
230     duration = f4long(durationfld);
231     Add2Sch(linebuff, minstart, start, duration, delta, status, curepid);
232     TrimSch(linebuff);
233     vatputf(win, r++, c, "REPID %-4ld%10s:%s", curepid, " ", linebuff);
234     while (curepid == f4long(repidfld))
235     {
236         x4skip(1L);
237     }
238 }
239
240 i4select(eqpreq_ndx5);
241 r++;
242 vatputf(win, r++, c, "Equipment Allocation Schedule");
243 vatputf(win, r++, c, "%20s|", " ");
244 x4top();
245 while (! d4eof())
246 {
247     for (i = 0; i < 299; i++) linebuff[i] = 32;
248     linebuff[299] = '\0';
249     curepid = f4long(epidfld);
250     strcpy(curdesc, f4str(eqpdescfld));
251     strupr(curdesc);
252     c4trim_n(curdesc, 25);
253     eqpid = f4long(epidfld);
254     thisone = 0;
255     while (curepid == eqpid && ! d4eof())
256     {
257         status = f4char(statusfld);
258         if ((status == 'A' || status == 'L' || status == 'C'))
259         {
260             thisone = 1;
261             start = f4long(startfld);
262             duration = f4long(durationfld);
263             tmpdesc = f4str(eqpdescfld);
264             strupr(tmpdesc);
265             c4trim_n(tmpdesc, 255);
266             repid = f4long(f4ref("REPID"));
267             Add2Sch(linebuff, minstart, start, duration, delta, status, repid);
268         }
269         x4skip(1L);
270         eqpid = f4long(epidfld);
271     }
272     TrimSch(linebuff);
273     if (thisone)
274         vatputf(win, r++, c, "%-12s ID %-4ld:%s", curdesc, curepid, linebuff);
275 }
276 d4close_all();
277 visible(win, YES, YES);
278 scroll_win_keys(win, lwinc, lwinc, pwinc, pwinc);
279 vfputwin(win, "GANTT.OUT", 1);
280 vdelete(win, NONE);
281 return (0);
282 }
283
284 static int TrimSch(char * linebuff)
285 {
286     int i;
287     for (i = 298; i > -1; i--)
288     {
289         if (linebuff[i] != 32 || i == 0)
290         {
291             if (i) linebuff[i + 1] = '\0';
292             else linebuff[0] = '\0';
293             break;
294         }
295     }

```

```
295     )
296 }
297
298 static int Add2Sch(char * linebuff, long minstart, long start, long duration,
299                  long delta, char marker, long repid)
300 {
301     double dbeg, dend;
302     int beg, end, i, buffstart, buffend;
303
304     dbeg = ((double) (start - minstart)) / ((double) delta);
305     dend = dbeg + ((double) duration / ((double) delta));
306     dbeg = (((ceil(dbeg) - dbeg) < .5) ? ceil(dbeg) : floor(dbeg));
307     dend = (((ceil(dend) - dend) < .5) ? ceil(dend) : floor(dend));
308
309     beg = (int) dbeg;
310     end = (int) dend - 1;
311
312     buffstart = 0;
313     buffend = 298;
314
315     if (beg < 0 || beg > 298) return (-1);
316     if (end > 298) end = 298;
317     for (i = beg; i <= end; i++) linebuff[i] = (char) (repid + 64);
318     return (1);
319 }
```

```

1 #include <stdio.h>
2 #include <conio.h>
3 #include <stdlib.h>
4 #include <dir.h>
5 #include <time.h>
6 #include <string.h>
7 #include <alloc.h>
8 #include "d4base.h"
9 #include "dw.h"
10 #include "dwmenu.h"
11 #include "dwsystem.h"
12 #include "rmrs.h"
13
14
15 typedef struct
16 {
17     int dbf;
18     int ndx;
19     long rec;
20 }
21     CONTEXT;
22
23 typedef struct llll
24 {
25     long val;
26     struct llll *next;
27 }
28     LLLL;
29
30 int CountLLLLNodes(LLLL * ptr);
31 void context_save(CONTEXT *);
32 void context_restore(CONTEXT *);
33 LLLL * CreateLLLLNode(void);
34 int scroll_win_keys(int, int, int, int, int);
35 int InLLLL(LLLL * ptr, long);
36 int DumpLLLLNodes(LLLL * ptr, FILE * debug, char * spaces);
37 LLLL * FreeLLLL(LLLL * ptr);
38
39 void context_save(CONTEXT * def)
40 {
41     def->rec = d4recno();
42     def->ndx = i4select(-1);
43     def->dbf = d4select(-1);
44 }
45
46 void context_restore(CONTEXT * def)
47 {
48     if (def->dbf > -1)
49     {
50         d4select(def->dbf);
51         i4select(def->ndx);
52         if (d4eof()) d4skip(-1L);
53         if (d4bof()) d4skip(1L);
54         d4go(def->rec);
55     }
56 }
57
58 int scroll_win_keys(HWND win, int lwinr, int lwinc, int pwinc, int pwinc)
59 {
60     unsigned int key = 13;
61     while (key != ESC)
62     {
63         switch (key = getkey())
64         {
65             case CTRLCURRT :
66                 if (vmovedir(win, 0, pwinc) != DWSUCCESS)
67                     vloc(win, 0, 0);
68                 break;
69             case CTRLCURLF :
70                 if (vmovedir(win, 0, - pwinc) != DWSUCCESS)
71                     vloc(win, 0, 0);
72             case CURUP :
73                 vmovedir(win, -1, 0);
74                 break;
75         }
76     }
77 }

```

```

70     case CURDN :
71         vmovedir(win, 1, 0);
72         break;
73     case CURLF :
74         vmovedir(win, 0, -1);
75         break;
76     case CURRT :
77         vmovedir(win, 0, 1);
78         break;
79     case CTRLUP :
80         if (vmovedir(win, - pwinr, 0) != DWSUCCESS)
81             vloc(win, 0, 0);
82         break;
83     case PGUP :
84         if (vmovedir(win, - pwinr, 0) != DWSUCCESS)
85             vloc(win, 0, 0);
86         break;
87     case CTRLCURDN :
88         if (vmovedir(win, pwinr, 0) != DWSUCCESS)
89             vloc(win, lwinr - pwinr, 0);
90         break;
91     case PGDN :
92         if (vmovedir(win, pwinr, 0) != DWSUCCESS)
93             vloc(win, lwinr - pwinr, 0);
94         break;
95     case HOME :
96         vloc(win, 0, 0);
97         break;
98     case ENDKEY :
99         vloc(win, lwinr - pwinr, 0);
100        break;
101    default :
102        break;
103    }
104    }
105    return (0);
106 }
107
108 int InLLLL(LLLL * ptr, long val)
109 {
110     if (ptr == NULL) return (0);
111
112     if (val == ptr->val) return (1);
113
114     if (ptr->val == -1000000L)
115     {
116         ptr->val = val;
117         return (0);
118     }
119
120     while (ptr->next)
121     {
122         ptr = ptr->next;
123         if (val == ptr->val) return (1);
124     }
125     ptr->next = CreateLLLLNode();
126     ptr = ptr->next;
127     ptr->val = val;
128     return (0);
129 }
130
131
132 LLLL * CreateLLLLNode()
133 {
134     LLLL * ptr;
135     ptr = (LLLL *) malloc(sizeof (LLLL));
136     ptr->val = -1000000L;
137     ptr->next = NULL;
138     return (ptr);
139 }
140
141 int CountLLLLNodes(LLLL * ptr)
142 {
143     int i = 0;

```

```

144     if (ptr == NULL)
145     {
146         return (i);
147     }
148     while (ptr->next)
149     {
150         i++;
151         ptr = ptr->next;
152     }
153     i++;
154     return (i);
155 }
156
157 int DumpLLLLNodes(LLLL * ptr, FILE * deblog, char * spaces)
158 {
159     int i = 0;
160     if (ptr == NULL)
161     {
162         return (i);
163     }
164     while (ptr->next)
165     {
166         i++;
167         fprintf(deblog, "%snode %d: %ld\n", spaces, i, ptr->val);
168         ptr = ptr->next;
169     }
170     i++;
171     fprintf(deblog, "%snode %d: %ld\n", spaces, i, ptr->val);
172     return (i);
173 }
174
175 LLLL * FreeLLLL(LLLL * ptr)
176 {
177     LLLL * tmp;
178
179     while (ptr->next != NULL)
180     {
181         tmp = ptr;
182         ptr = ptr->next;
183         free(tmp);
184     }
185     free(ptr);
186     return (NULL);
187 }
188
189 /* FUNCTIONS TO OPEN DBF FILES AND ATTACH ALL INDEX FILES
190 *
191 * DBFfname      NDXREF      NDXfname      KEY
192 * -----
193 * repair        repair_ndx1  reprndx1    REPID
194 *                repair_ndx2  reprndx2    upper (STATUS)+str (PRIORITY,2,0)
195 *                +str (FACNUM, 4,0)
196 *
197 * facprt        facprt_ndx1   facpndx1    FACNUM
198 *
199 * eqpsup        eqpsup_ndx1   eqsundx1    upper (EQPDESC)
200 *
201 * eqpreq        eqpreq_ndx1   eqrendx1    REPID
202 *                eqpreq_ndx2   eqrendx2    upper (EQPDESC)+str (START,10,0)
203 *                eqpreq_ndx3   eqrendx3    upper (EQPDESC)+str (START+DURATION,10,0)
204 *                eqpreq_ndx4   eqrendx4    START
205 *                eqpreq_ndx5   eqrendx5    upper (EQPDESC)+str (EQPID,10,0)
206 *
207 * matsup        matsup_ndx1   masundx1    upper (MATDESC)
208 *
209 * matreq        matreq_ndx1   marendx1    REPID
210 *                matreq_ndx2   marendx2    MATID
211 *
212 * repinfo       repinfo_ndx1  repindx1    REPID
213 *
214 */
215
216 int repair_dbf, repair_ndx1, repair_ndx2;
217 int eqpsup_dbf, eqpsup_ndx1;

```

```
218 int  eqpreq_dbf, eqpreq_ndx1, eqpreq_ndx2, eqpreq_ndx3,
219      eqpreq_ndx4, eqpreq_ndx5;
220 int  matsup_dbf, matsup_ndx1;
221 int  matreq_dbf, matreq_ndx1, matreq_ndx2;
222 int  repinfo_dbf, repinfo_ndx1;
223 int  facprt_dbf, facprt_ndx1;
224
225 int open_repair(int rendx)
226 {
227     int status = 0;
228     repair_dbf = d4use_excl("REPAIR.DBF");
229     if (repair_dbf < 0) return (repair_dbf);
230     x4filter(d4deleted);
231     repair_ndx1 = i4open("REPRNDX1.NDX");
232     if (repair_ndx1 < 0) return (repair_ndx1);
233     repair_ndx2 = i4open("REPRNDX2.NDX");
234     if (repair_ndx2 < 0) return (repair_ndx2);
235
236     if (rendx) status = i4reindex(-1);
237     return status;
238 }
239
240 int open_eqpsup(int rendx)
241 {
242     int status = 0;
243     eqpsup_dbf = d4use_excl("eqpsup.DBF");
244     if (eqpsup_dbf < 0) return (eqpsup_dbf);
245     x4filter(d4deleted);
246     eqpsup_ndx1 = i4open("EQSUNDX1.NDX");
247     if (eqpsup_ndx1 < 0) return (eqpsup_ndx1);
248     if (rendx) status = i4reindex(-1);
249     return status;
250 }
251
252 int open_eqpreq(int rendx)
253 {
254     int status = 0;
255     eqpreq_dbf = d4use_excl("eqpreq.DBF");
256     if (eqpreq_dbf < 0) return (eqpreq_dbf);
257     x4filter(d4deleted);
258     eqpreq_ndx1 = i4open("EQRENDX1.NDX");
259     if (eqpreq_ndx1 < 0) return (eqpreq_ndx1);
260     eqpreq_ndx2 = i4open("EQRENDX2.NDX");
261     if (eqpreq_ndx2 < 0) return (eqpreq_ndx2);
262     eqpreq_ndx3 = i4open("EQRENDX3.NDX");
263     if (eqpreq_ndx3 < 0) return (eqpreq_ndx3);
264     eqpreq_ndx4 = i4open("EQRENDX4.NDX");
265     if (eqpreq_ndx4 < 0) return (eqpreq_ndx4);
266     eqpreq_ndx5 = i4open("EQRENDX5.NDX");
267     if (eqpreq_ndx5 < 0) return (eqpreq_ndx5);
268     if (rendx) status = i4reindex(-1);
269     return status;
270 }
271
272 int open_matsup(int rendx)
273 {
274     int status = 0;
275     matsup_dbf = d4use_excl("matsup.DBF");
276     if (matsup_dbf < 0) return (matsup_dbf);
277     x4filter(d4deleted);
278     matsup_ndx1 = i4open("MASUNDX1.NDX");
279     if (matsup_ndx1 < 0) return (matsup_ndx1);
280     if (rendx) status = i4reindex(-1);
281     return status;
282 }
283
284 int open_matreq(int rendx)
285 {
286     int status = 0;
287     matreq_dbf = d4use_excl("matreq.DBF");
288     if (matreq_dbf < 0) return (matreq_dbf);
289     x4filter(d4deleted);
290     matreq_ndx1 = i4open("MARENDX1.NDX");
291     if (matreq_ndx1 < 0) return (matreq_ndx1);
```

```
292     matreq_ndx2 = i4open("MARENDX2.NDX");
293     if (matreq_ndx2 < 0) return (matreq_ndx2);
294     if (rendx) status = i4reindex(-1);
295     return status;
296 }
297
298 int open_repinfo(int rendx)
299 {
300     int status = 0;
301     repinfo_dbf
302     = d4use_excl("repinfo.DBF");
303     if (repinfo_dbf < 0) return (repinfo_dbf);
304     x4filter(d4deleted);
305     repinfo_ndx1 = i4open("REPINDEX1.NDX");
306     if (repinfo_ndx1 < 0) return (repinfo_ndx1);
307     if (rendx) status = i4reindex(-1);
308     return status;
309 }
310
311 int open_facprt(int rendx)
312 {
313     int status = 0;
314     facprt_dbf = d4use_excl("facprty.DBF");
315     if (facprt_dbf < 0) return (facprt_dbf);
316     x4filter(d4deleted);
317     eqpsup_ndx1 = i4open("FACPNDX1.NDX");
318     if (facprt_ndx1 < 0) return (facprt_ndx1);
319     if (rendx) status = i4reindex(-1);
320     return status;
321 }
```

```

1 #include <stdio.h>
2 #include <conio.h>
3 #include <stdlib.h>
4 #include <dir.h>
5 #include <time.h>
6 #include <string.h>
7 #include <alloc.h>
8 #include "rmrsutil.h"
9 #include "rmrs.h"
10 #include "d4base.h"
11 #include "dw.h"
12 #include "dwmenu.h"
13 #include "dwsystem.h"
14
15 long FindEarliest(char * eqpdsc, long duration, long repstart, long recno);
16 int SchedAuto(int resched);
17
18 static FILE * deblog;
19 static long defrepstart;
20
21 /* SchedAuto
22 *
23 *
24 *
25 */
26 int SchedAuto(int resched)
27 {
28     int stat, exitcode = 0, totalreprs = 0, compreprs = 0;
29     char * statptr;
30     long repid;
31     extern HWND win;
32     LLLL * replst = NULL, * rlptr = NULL;
33     CONTEXT repair_ctx;
34
35     /* SET MODULE GLOBALS */
36     deblog = fopen("debug.log", "w");
37     defrepstart = time(NULL) + 1800L; /* CURRENT TIME + 30 MINUTES */
38
39     /* WINDOW STUFF */
40     win = vcreat(7, 40, EMPHNORML, YES);
41     vautoshd(win, CURRENT, SHADOW75, BOTTOMRIGHT);
42     vlocate(win, 9, 20);
43     vframe(win, EMPHNORML, FRDOUBLE);
44     if (resched) vmtitle(win, _TOP, CENTERJUST, EMPHNORML, " Full Schedule ");
45     else vmtitle(win, _TOP, CENTERJUST, EMPHNORML, " Incremental Schedule ");
46     visible(win, YES, YES);
47
48     /* OPEN DBF AND INDEX FILES */
49     vatputs(win, 1, 2, "Opening Data Files ...");
50     if (open_eqpreq(1))
51     {
52         u4error(-1, "BAD RETURN CODE", "OPEN_EQPREQ", (char *) 0);
53         vdelete(win, NONE);
54         return (-1);
55     }
56     if (open_eqpsup(1))
57     {
58         u4error(-2, "BAD RETURN CODE", "OPEN_EQPSUP", (char *) 0);
59         vdelete(win, NONE);
60         return (-2);
61     }
62     if (open_repair(1))
63     {
64         u4error(-3, "BAD RETURN CODE", "OPEN_REPAIR", (char *) 0);
65         vdelete(win, NONE);
66         return (-3);
67     }
68
69     /* RESET STATUS FOR REPAIR=Q & EQPREQ=A TO P & N RESPECTIVELY IF resched */
70     if (resched)
71     {
72         vatputs(win, 1, 2, "Clearing Schedule ...");
73         d4select(repair_dbf);
74         for (x4top(); !d4eof(); x4skip(1L))

```

```

75         if (f4char(f4ref("STATUS")) == 'Q') f4r_char(f4ref("STATUS"), 'P');
76
77         d4select(eqpreq_dbf);
78         for (x4top(); !d4eof(); x4skip(1L))
79             if (f4char(f4ref("STATUS")) == 'A') f4r_char(f4ref("STATUS"), 'N');
80     }
81
82     /* BUILD RECNO LIST FOR ELIGABLE REPAIRS */
83     d4select(repair_dbf);
84     i4select(repair_ndx2);
85     stat = x4seek_str("P"); /* find first POSSIBLE repair */
86     stat = ((stat == 1 || stat == 0) ? 1 : 0); /*
87     if (!stat)
88     {
89         u4error(-4, "No possible repairs found!", (char *) 0);
90         d4close_all();
91         vdelete(win, NONE);
92         return (-4); /* return no possible repairs found */
93     }
94     replst = CreateLLLLNode();
95     while (stat)
96     {
97         InLLLL(replst, d4recno());
98         x4skip(1L);
99         statptr = f4str(f4ref("STATUS"));
100        if (statptr[0] == 'P' && !d4eof()) stat = 1;
101        else stat = 0;
102    }
103    fprintf(deblog, "RECNO List of repairs to be scheduled\n");
104    DumpLLLLNodes(replst, deblog, " ");
105    fprintf(deblog, "\n");
106    totalreprs = CountLLLLNodes(replst);
107
108    /* SCHEDULE EACH ELIGABLE REPAIR IN replst */
109    complreprs = 0;
110    rlptr = replst;
111    while (rlptr)
112    {
113        x4go(rlptr->val);
114        repid = f4long(f4ref("REPID"));
115        vatputf(win, 1, 2, "Processing REPID : %4ld", repid);
116        vatputf(win, 2, 2, "Repairs Processed: %4d", complreprs);
117        vatputf(win, 3, 2, "Percent Complete : %3.0f%", ((float) complreprs / (float) totalreprs) * 100);
118        fprintf(deblog, "*****\n");
119        fprintf(deblog, "\nBEGIN PROCESSING REPID: %4ld\n", repid);
120        if (CalcSchedule(repid))
121        {
122            exitcode = 1;
123            break;
124        }
125        f4r_char(f4ref("STATUS"), 'Q');
126        fprintf(deblog, "PROCESSING COMPLETE REPID: %4ld\n", repid);
127        rlptr = rlptr->next;
128        complreprs++;
129    }
130    fprintf(deblog, "*****\n");
131    fprintf(deblog, "SCHEDULE PROCESSING COMPLETE\n");
132
133    /* DISPLAY FINAL STATS */
134    vatputf(win, 2, 2, "Repairs Processed: %3d", complreprs);
135    vatputs(win, 3, 2, "Percent Complete : 100");
136    vatputs(win, 5, 2, "press any key ...");
137    getch();
138    /* FREE replst STORAGE */
139    if (replst) FreeLLLL(replst);
140    replst = NULL;
141    rlptr = NULL;
142    /* REMOVE PROGRESS WINDOW */
143    vdelete(win, NONE);
144    /* CLOSE ALL FILES */
145    d4close_all();
146    fclose(deblog);
147    return (exitcode);

```

```

148 }
149
150
151 /* CalcSchedule
152 *
153 *
154 */
155 int CalcSchedule(long repid)
156 {
157     int done, different, stat;
158     long recno, eqpid, start, duration, repstart, maxstart;
159     time_t tempt;
160     char * ptr, eqpdesc[26], tmp[26];
161     LLLL * replst = NULL, * rlpstr = NULL;
162     CONTEXT def;
163     extern HWND win;
164
165     context_save(& def);
166
167     if (BuildList(repid, & replst)) u4error(0, "BUILDLIST ERROR", (char *) 0);
168     fprintf(deblog, " EQP RECNO List for REPID %ld\n", repid);
169     DumpLLLLNodes(replst, deblog, " ");
170
171     d4select(eqpreq_dbf);
172     i4select(eqpreq_ndxl);
173
174     repstart = defrepstart;          /* CURRENT TIME + 30 MINUTES */
175
176     done = 0;
177     while (!done)
178     {
179         rlpstr = replst;
180         while (rlpstr)
181         {
182             x4go(rlpstr->val);
183             recno = rlpstr->val;
184             fprintf(deblog, " EQP RECNO %ld processing\n", recno);
185             strcpy(eqpdesc, f4str(f4ref("EQPDESC")));
186             c4trim_n(eqpdesc, 25);
187             strupr(eqpdesc);
188             duration = f4long(f4ref("DURATION"));
189             fprintf(deblog, " Looking for time slot for %s >= %s", eqpdesc, ctime(& repstart));
190
191             start = FindEarliest(eqpdesc, duration, repstart, recno);
192             fprintf(deblog, " Found starttime for %s of %s\n", eqpdesc, ctime(& start));
193             f4r_long(f4ref("START"), start);
194             rlpstr = rlpstr->next;
195         }
196
197         fprintf(deblog, " CHECKING IF ALL SAME\n");
198         rlpstr = replst;
199         x4go(rlpstr->val);
200         maxstart = f4long(f4ref("START"));
201         different = 0;
202         rlpstr = rlpstr->next;
203         while (rlpstr)
204         {
205             x4go(rlpstr->val);
206             start = f4long(f4ref("START"));
207             if (start != maxstart)
208             {
209                 maxstart = (start > maxstart ? start : maxstart);
210                 different = 1;
211             }
212             rlpstr = rlpstr->next;
213         }
214         repstart = maxstart;
215         if (!different)
216         {
217             done = 1;
218             fprintf(deblog, " POSTING - all same at %s\n", ctime(& repstart));
219             Post(repid, 1);          /* CHANGE ALL STATUS 'T' TO 'A' FOR REPID */
220         }
221     }
222 }

```

```

221     {
222         fprintf(deblog, "    ROLLING BACK - not same, max= %s\n", ctime(& repstart));
223         Post(repid, 0);          /* CHANGE ALL STATUS 'T' BACK TO 'N' FOR REPID */
224     }
225 }
226 if (replst) FreeLLLL(replst);
227 replst = NULL;
228 rlpstr = NULL;
229 context_restore(& def);
230 return 0;
231 }
232
233
234 /* Post
235 *
236 *
237 */
238 int Post(long repid, int mode)
239 {
240     long tmprepid, statusfld;
241     char tmpstatus;
242     int stat;
243     CONTEXT def;
244     extern HWND win;
245
246     context_save(& def);
247
248     d4select(eqpreq_dbf);
249     i4select(eqpreq_ndx1);
250
251     stat = d4seek_double(repid);
252     stat = ((stat == 0) ? 1 : 0);
253     if (!stat)
254     {
255         context_restore(& def);
256         return (-2);          /* error */
257     }
258     statusfld = f4ref("STATUS");
259
260     if (mode == 1)
261     {
262         while (stat)
263         {
264             tmpstatus = f4char(statusfld);
265             if (tmpstatus == 'T') f4r_char(statusfld, 'A');
266             d4skip(1L);
267             tmprepid = f4long(f4ref("REPID"));
268             if (tmprepid == repid && ! d4eof()) stat = 1;
269             else stat = 0;
270         }
271     }
272
273     if (mode == 0)
274     {
275         while (stat)
276         {
277             tmpstatus = f4char(statusfld);
278             if (tmpstatus == 'T') f4r_char(statusfld, 'N');
279             d4skip(1L);
280             tmprepid = f4long(f4ref("REPID"));
281             if (tmprepid == repid && ! d4eof()) stat = 1;
282             else stat = 0;
283         }
284     }
285     context_restore(& def);
286     return (0);
287 }
288
289
290 /* BuildList
291 *
292 *
293 */
294 int BuildList(long repid, LLLL **ptr)

```

```

295 {
296     long eqprecno, tmprepid;
297     int stat;
298     CONTEXT def;
299
300     context_save(& def);
301
302     * ptr = CreateLLLLNode();
303     d4select(eqpreq_dbf);
304     i4select(eqpreq_ndx1);
305     stat = x4seek_double(repid);
306     stat = ((stat == 1 || stat == 0) ? 1 : 0);
307     if (! stat)
308     {
309         context_restore(& def);
310         return (-2);          /* error: no equip found for repid in eqpreq */
311     }
312     while (stat)
313     {
314         eqprecno = d4recno();
315         /* ADD TO LIST */
316         InLLLL(* ptr, eqprecno);
317         x4skip(1L);
318         tmprepid = f4long(f4ref("REPID"));
319         if (tmprepid == repid) stat = 1;
320         else stat = 0;
321     }
322     context_restore(& def);
323     return (0);
324 }
325
326 /* FindEarliest
327 *
328 *
329 */
330 long FindEarliest(char * eqpdesc, long duration, long repstart, long recno)
331 {
332     int result, done, pieceseqp, found;
333     char status, * tmpstr;
334     CONTEXT def;
335     long tmp, tmpstart, tmpduration, tmpend, tmpeqp, eqpid;
336     long statusfld, startfld, durationfld, eqpdescfld, eqpidfld;
337     LLLL * idlst = NULL, * straddle = NULL, * lookback = NULL, * ptr = NULL;
338     extern HWND win;
339
340     context_save(& def);
341
342     strupr(eqpdesc);
343     c4trim_n(eqpdesc, 25);
344
345     /* GET ID's FOR EACH EQPDESC IN EQPSUP FILE */
346     d4select(eqpsup_dbf);
347     i4select(eqpsup_ndx1);          /* eqpdesc */
348     eqpdescfld = f4ref("EQPDESC");
349     eqpidfld = f4ref("EQPID");
350     result = d4seek_str(strupr(eqpdesc));
351     if (result != 0 && result != 1) return (1);
352     idlst = CreateLLLLNode();
353     InLLLL(idlst, f4long(eqpidfld));
354     x4skip(1L);
355     tmpstr = strupr(f4str(eqpdescfld));
356     c4trim_n(tmpstr, 255);
357     while (strcmp(tmpstr, eqpdesc) == 0 && ! d4eof())
358     {
359         InLLLL(idlst, f4long(eqpidfld));
360         x4skip(1L);
361         tmpstr = strupr(f4str(eqpdescfld));
362         c4trim_n(tmpstr, 255);
363     }
364     pieceseqp = CountLLLLNodes(idlst);
365     fprintf(deblog, "      Found %d instances of %s in eqpsup file\n", pieceseqp, eqpdesc);
366     fprintf(deblog, "      EQP ID List\n");
367     DumpLLLLNodes(idlst, deblog, "      ");
368     /* ***** */

```

```

369
370  /**** LOOK FOR ANY EQPDESC ID NOT USED AFTER REPSTART ****/
371  d4select (eqpreq_dbf);
372  i4select (eqpreq_ndx2);          /* eqpdesc+start          */
373  statusfld = f4ref("STATUS");
374  startfld = f4ref("START");
375  durationfld = f4ref("DURATION");
376  eqpdescfld = f4ref("EQPDESC");
377  eqpidfld = f4ref("EQPID");
378  result = d4seek_str(strupr(eqpdesc));
379  if (result != 0 && result != 1) return (-1L);
380  tmpstr = strupr(f4str(eqpdescfld));
381  c4trim_n(tmpstr, 255);
382  if (strcmp(tmpstr, eqpdesc) != 0) return (-2L);
383  ptr = idlst;
384  do
385  {
386      found = 0;
387      result = d4seek_str(strupr(eqpdesc));
388      tmpstr = strupr(f4str(eqpdescfld));
389      c4trim_n(tmpstr, 255);
390      while (strcmp(tmpstr, eqpdesc) == 0 && ! d4eof())
391      {
392          tmpeqpid = f4long(eqpidfld);
393          status = f4char(statusfld);
394          tmpstart = f4long(startfld);
395          tmpduration = f4long(durationfld);
396          if ((ptr->val == tmpeqpid) &&
397              (status == 'A' || status == 'L' || status == 'T') &&
398              (tmpstart + tmpduration > repstart))
399          {
400              {
401                  found = 1;
402                  break;
403              }
404              d4skip(1L);
405              tmpstr = strupr(f4str(eqpdescfld));
406              c4trim_n(tmpstr, 255);
407          }
408          if (! found)
409          {
410              fprintf(deblog, "      EQPID %ld COMPLETELY FREE AFTER REPSTART\n", ptr->val);
411              d4go(recno);
412              f4r_char(f4ref("STATUS"), 'T');
413              f4r_long(f4ref("START"), repstart);
414              f4r_long(f4ref("EQPID"), ptr->val);
415              FreeLLLL(idlst);
416              context_restore(& def);
417              return (repstart);
418          }
419          ptr = ptr->next;
420      }
421      while (ptr);
422  /*****
423  /**** ADD ANY ALLOCATED ID's THAT STRADDLE REPSTART TO lookback ****/
424  ptr = idlst;
425  lookback = CreateLLLLNode();
426  d4select (eqpreq_dbf);
427  i4select (eqpreq_ndx2);          /* eqpdesc+...          */
428  result = d4seek_str(strupr(eqpdesc));
429  if (result != 0 && result != 1) return (-1L);
430  tmpstr = strupr(f4str(eqpdescfld));
431  c4trim_n(tmpstr, 255);
432  if (strcmp(tmpstr, eqpdesc) != 0) return (-2L);
433  do
434  {
435      result = d4seek_str(strupr(eqpdesc));
436      tmpstr = strupr(f4str(eqpdescfld));
437      c4trim_n(tmpstr, 255);
438      while (strcmp(tmpstr, eqpdesc) == 0 && ! d4eof())
439      {
440          tmpeqpid = f4long(eqpidfld);
441          status = f4char(statusfld);

```

```

442         tmpstart = f4long(startfld);
443         tmpduration = f4long(durationfld);
444         if ((ptr->val == tmpeqpid) &&
445             (status == 'A' || status == 'L' || status == 'T') &&
446             (tmpstart + tmpduration > repstart) &&
447             (tmpstart <= repstart)
448         )
449         {
450             InLLLL(lookback, ptr->val);
451             break;
452         }
453         d4skip(1L);
454         tmpstr = strupr(f4str(eqpdscfld));
455         c4trim_n(tmpstr, 255);
456     }
457     ptr = ptr->next;
458 }
459 // while (ptr);
460 //     fprintf(deblog, "Lookback List\n");
461 //     DumpLLLLNodes(lookback);
462 // *****/
463 /**** SEARCH BACKWARD FROM FIRST ID START>REPSTART TO REPSTART **/
464 /**** DO NOT SEARCH BACKWARDS IF ID IS IN lookback *****/
465 d4select(eqpreq_dbf);
466 i4select(eqpreq_ndx2); /* eqpdsc+start - important */
467 result = d4seek_str(strupr(eqpdsc));
468 if (result != 0 && result != 1) return (-1L);
469 tmpstr = strupr(f4str(eqpdscfld));
470 c4trim_n(tmpstr, 255);
471 if (strcmp(tmpstr, eqpdsc) != 0) return (-2L);
472 while (strcmp(tmpstr, eqpdsc) == 0 && ! d4eof()) /* FOR ALL EQPID's */
473 {
474     tmpeqpid = f4long(eqpidfld);
475     status = f4char(statusfld);
476     tmpstart = f4long(startfld);
477     tmpduration = f4long(durationfld);
478     if ((status == 'A' || status == 'L' || status == 'T') &&
479         (tmpstart >= repstart)
480     )
481     {
482         if (! InLLLL(lookback, tmpeqpid) && (tmpstart - repstart) >= duration)
483         {
484             /** BINGO **/
485             fprintf(deblog, "      EQPID %ld FREE AT REPSTART\n", tmpeqpid);
486             d4go(recno);
487             f4r_char(f4ref("STATUS"), 'T');
488             f4r_long(f4ref("START"), repstart);
489             f4r_long(f4ref("EQPID"), tmpeqpid);
490             FreeLLLL(idlst);
491             FreeLLLL(lookback);
492             context_restore(& def);
493             return (repstart);
494         }
495     }
496     d4skip(1L);
497     tmpstr = strupr(f4str(eqpdscfld));
498     c4trim_n(tmpstr, 255);
499 }
500 // *****/
501 /**** SEARCH FOWARD FROM EACH ID FOR END > REPSTART **/
502 ptr = idlst;
503 d4select(eqpreq_dbf);
504 i4select(eqpreq_ndx3); /* eqpdsc+(start+duration) */
505 result = d4seek_str(strupr(eqpdsc));
506 if (result != 0 && result != 1) return (-1L);
507 tmpstr = strupr(f4str(eqpdscfld));
508 c4trim_n(tmpstr, 255);
509 if (strcmp(tmpstr, eqpdsc) != 0) return (-2L);
510 while (strcmp(tmpstr, eqpdsc) == 0 && ! d4eof())
511 {
512     tmpeqpid = f4long(eqpidfld);
513     status = f4char(statusfld);

```

```

515     tmpstart = f4long(startfld);
516     tmpduration = f4long(durationfld);
517     if ((status == 'A' || status == 'L' || status == 'T') &&
518         (tmpstart + tmpduration) >= repstart)
519     )
520     {
521         if (SlotFoward(d4recno(), duration))
522         {
523             /** BINGO ***/
524             tmp = tmpstart + tmpduration;
525             fprintf(deblog, "      EQPID %ld AVAIL AT %s\n", tmpeqpid, ctime(& tmp));
526             d4go(recno);
527             f4r_char(f4ref("STATUS"), 'T');
528             f4r_long(f4ref("START"), tmpstart + tmpduration);
529             f4r_long(f4ref("EQPID"), tmpeqpid);
530             FreeLLLL(idlst);
531             FreeLLLL(lookback);
532             context_restore(& def);
533             return (tmpstart + tmpduration);
534         }
535     }
536     d4skip(1L);
537     tmpstr = strupr(f4str(eqpdscfld));
538     c4trim_n(tmpstr, 255);
539 }
540 /*****
541 context_restore(& def);
542 fprintf(deblog, "ERROR REACHED END OF FindEarliest ROUTINE\n");
543 return (-1L);
544 /* SHOULD NEVER REACH THIS EXIT POINT
545 */
546
547 int SlotFoward(long rec, long duration)
548 {
549     long statusfld, startfld, durationfld, eqpidfld, eqpdscfld;
550     long curid, curend, tststart, tmpeqpid;
551     char * tmpstr, eqpdsc[255], status;
552     int iseof;
553
554     CONTEXT def;
555
556     context_save(& def);
557
558     d4select(eqpreq_dbf);
559     i4select(eqpreq_ndx2);
560     statusfld = f4ref("STATUS");
561     startfld = f4ref("START");
562     durationfld = f4ref("DURATION");
563     eqpdscfld = f4ref("EQPDESC");
564     eqpidfld = f4ref("EQPID");
565
566     d4go(rec);
567     curend = f4long(startfld) + f4long(durationfld);
568     curid = f4long(eqpidfld);
569     status = f4char(statusfld);
570     strcpy(eqpdsc, f4str(eqpdscfld));
571     strupr(eqpdsc);
572     c4trim_n(eqpdsc, 255);
573     do
574     {
575         d4skip(1L);
576         iseof = d4eof();
577         tmpeqpid = f4long(eqpidfld);
578         tmpstr = f4str(eqpdscfld);
579         strupr(tmpstr);
580         c4trim_n(tmpstr, 255);
581         status = f4char(statusfld);
582     }
583     while (((curid != tmpeqpid) ||
584            (status != 'A' && status != 'L' && status != 'T')) &&
585            ! iseof &&
586            strcmp(eqpdsc, tmpstr) == 0
587    );
588

```

```
589     if ((strcmp(eqpdesc, tmpstr) != 0) || d4eof())
590     {
591         context_restore(& def);
592         return (1);
593     }
594
595
596     if ((f4long(startfld) - curend) >= duration)
597     {
598         context_restore(& def);
599         return (1);
600     }
601     context_restore(& def);
602     return (0);
603 }
```

```

1 #include <stdio.h>
2 #include <math.h>
3 #include <conio.h>
4 #include <stdlib.h>
5 #include <dir.h>
6 #include <time.h>
7 #include <string.h>
8 #include <alloc.h>
9 #include "d4base.h"
10 #include "dw.h"
11 #include "dwmenu.h"
12 #include "dwsystem.h"
13
14 FILE * sched;
15
16 typedef struct
17 {
18     int dbf;
19     int ndx;
20     long rec;
21 }
22     CONTEXT;
23
24 int eqpreq_dbf, eqpreq_ndx;
25
26 int main(int argc, char * argv[])
27 {
28     d4init();
29     sched = fopen("sched", "w");
30     PrtSch();
31     d4init_undo();
32     exit(0);
33 }
34
35 /* PrtSch
36 *
37 *
38 */
39 int PrtSch(void)
40 {
41     int done, i, thisone, r = 0, c = 1;
42     char status, linebuff[300], curdesc[26], * tmpdesc, * timeptr;
43     long start, duration, currepid, minstart, delta = 900L, curtime;
44     long cureqid, eqpidfld, eqpdescfld, eqpid;
45     long statusfld, startfld, durationfld, repidfld, repid;
46     HWND win;
47     unsigned int key = 0;
48
49     /* WINDOW STUFF */
50     win = vcreat(50, 80, ERR, YES);
51     vwind(win, 23, 78, 1, 1);
52     vlocate(win, 1, 1);
53     vframe(win, ERR, FRDOUBLE);
54     vshadow(win, CURRENT, SHADOW75, BOTTOMRIGHT);
55     vmtitle(win, _TOP, CENTERJUST, ERR, " Auto-mode Repair Schedule ");
56
57     eqpreq_dbf = d4use_excl("EQPREQ.DBF");
58     eqpreq_ndx = i4index("tmp", "START", 0, 0);
59
60     statusfld = f4ref("STATUS");
61     startfld = f4ref("START");
62     durationfld = f4ref("DURATION");
63     repidfld = f4ref("REPID");
64     eqpidfld = f4ref("EQPID");
65     eqpdescfld = f4ref("EQPDESC");
66
67     /***** GET EARLIST START *****/
68     minstart = 2147000000;
69     d4top();
70     while (! d4eof())
71     {
72         status = f4char(statusfld);
73         start = f4long(startfld);

```

```

72     if ((status == 'A' || status == 'L' || status == 'C') && start < minstart)
73         minstart = start;
74     d4skip(1L);
75 }
76
77 /* BUILD SCHEDULE */
78 curtime = time(NULL);
79 timeptr = ctime(& curtime);
80 timeptr[strlen(timeptr) - 1] = '\0';
81 vatputf(win, r++, c, "Schedule as of :%s", timeptr);
82 vatputf(win, r++, c, "Each character represents %ld minutes", delta / 60);
83 vatputf(win, r++, c, "Repair Schedule");
84 vatputf(win, r++, c, "%20s|", " ");
85 d4top();
86 while (! d4eof())
87 {
88     for (i = 0; i < 299; i++) linebuff[i] = 32;
89     linebuff[299] = '\0';
90     status = f4char(statusfld);
91     while (! (status == 'A' || status == 'L'))
92     {
93         d4skip(1L);
94         status = f4char(statusfld);
95     }
96     currepid = f4long(repidfld);
97     start = f4long(startfld);
98     duration = f4long(durationfld);
99     Add2Sch(linebuff, minstart, start, duration, delta, status, currepid);
100    TrimSch(linebuff);
101    vatputf(win, r++, c, "REPID %-4ld%10s:%s", currepid, " ", linebuff);
102    while (currepid == f4long(repidfld))
103    {
104        d4skip(1L);
105    }
106 }
107
108
109 eqpreq_ndx = i4index("tmp", "upper(eqpdesc)+str(eqpid,10,0)", 0, 0);
110
111 vatputf(win, r++, c, "Equipment Allocation Schedule");
112 vatputf(win, r++, c, "%20s|", " ");
113 d4top();
114 while (! d4eof())
115 {
116     for (i = 0; i < 299; i++) linebuff[i] = 32;
117     linebuff[299] = '\0';
118     cureqid = f4long(eqpidfld);
119     strcpy(curdesc, f4str(eqpdescfld));
120     strupr(curdesc);
121     c4trim_n(curdesc, 25);
122     eqpid = f4long(eqpidfld);
123     thisone = 0;
124     while (cureqid == eqpid && ! d4eof())
125     {
126         status = f4char(statusfld);
127         if ((status == 'A' || status == 'L' || status == 'C'))
128         {
129             thisone = 1;
130             start = f4long(startfld);
131             duration = f4long(durationfld);
132             tmpdesc = f4str(eqpdescfld);
133             strupr(tmpdesc);
134             c4trim_n(tmpdesc, 255);
135             repid = f4long(f4ref("REPID"));
136             Add2Sch(linebuff, minstart, start, duration, delta, status, repid);
137         }
138         d4skip(1L);
139         eqpid = f4long(eqpidfld);
140     }
141     TrimSch(linebuff);
142     if (thisone)
143         vatputf(win, r++, c, "%-12s ID %-4ld:%s", curdesc, cureqid, linebuff);
144 }
145 visible(win, YES, YES);

```

```

146 while (key != ESC)
147 {
148     switch (key = getkey())
149     {
150         case CURUP :
151             vmovedir(win, -1, 0);
152             break;
153         case CURDN :
154             vmovedir(win, 1, 0);
155             break;
156         case CURLF :
157             vmovedir(win, 0, -1);
158             break;
159         case CURRT :
160             vmovedir(win, 0, 1);
161             break;
162         case PGUP :
163             vmovedir(win, -20, 0);
164             break;
165         case PGDN :
166             vmovedir(win, 20, 0);
167             break;
168         default :
169             break;
170     }
171 }
172 clrscr();
173 vexit(0);
174 }
175
176 int context_save(CONTEXT * def)
177 {
178     def->rec = d4recno();
179     def->ndx = i4select(-1);
180     def->dbf = d4select(-1);
181     return (1);
182 }
183
184 int context_restore(CONTEXT * def)
185 {
186     d4select(def->dbf);
187     i4select(def->ndx);
188     if (d4eof()) d4skip(-1L);
189     d4go(def->rec);
190     return (1);
191 }
192
193 int TrimSch(char * linebuff)
194 {
195     int i;
196     for (i = 298; i > -1; i--)
197     {
198         if (linebuff[i] != 32 || i == 0)
199         {
200             if (i) linebuff[i + 1] = '\0';
201             else linebuff[0] = '\0';
202             break;
203         }
204     }
205 }
206
207 int Add2Sch(char * linebuff, long minstart, long start, long duration,
208             long delta, char marker, long repid)
209 {
210     double dbeg, dend;
211     int beg, end, i, buffstart, buffend;
212
213     dbeg = ((double) (start - minstart)) / ((double) delta);
214     dend = dbeg + ((double) duration / ((double) delta));
215     dbeg = (((ceil(dbeg) - dbeg) < .5) ? ceil(dbeg) : floor(dbeg));
216     dend = (((ceil(dend) - dend) < .5) ? ceil(dend) : floor(dend));
217
218     beg = (int) dbeg;
219     end = (int) dend - 1;

```

```
220
221     buffstart = 0;
222     buffend = 298;
223
224     if (beg < 0 || beg > 298) return (-1);
225     if (end > 298) end = 298;
226     for (i = beg; i <= end; i++) linebuff[i] = (char) (repid + 64);
227     return (1);
228 }
229
230
231
232
```