ARI Research Note 90-11

Course of Action Assessment Tool (COAAT) Software Description

C. Glen Ross

Science Applications International Corporation

for

Contracting Officer's Representative
Jon Fallesen

Field Unit at Fort Leavenworth, Kansas
Stanley M. Halpin, Chief

Systems Research Laboratory
Robin L. Keesee, Director

February 1990

United States Army
Research Institute for the Behavioral and Social Sciences

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 is unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE --	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) SAIC-89/1636	5. MONITORING ORGANIZATION REPORT NUMBER(S) ARI Research Note 90-11
6a. NAME OF PERFORMING ORGANIZATION Science Applications International Corporation 6b. OFFICE SYMBOL (If applicable) --	7a. NAME OF MONITORING ORGANIZATION U.S. Army Research Institute Field Unit at Fort Leavenworth
6c. ADDRESS (City, State, and ZIP Code) 424 Delaware, Suite Ce Leavenworth, KS 66048	7b. ADDRESS (City, State, and ZIP Code) P.O. Box 3407 Fort Leavenworth, KS 66027-0347
8a. NAME OF FUNDING/SPONSORING ORGANIZATION U.S. Army Research Institute for the Behavioral and Social Sciences 8b. OFFICE SYMBOL (If applicable) PERI-S	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER 9-X5E-7825E-1
8c. ADDRESS (City, State, and ZIP Code) 5001 Eisenhower Avenue Alexandria, VA 22333-5600	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO.: 62785A PROJECT NO.: 790 TASK NO.: 1304 WORK UNIT ACCESSION NO.: C1

11. TITLE (Include Security Classification)
Course of Action Assessment Tool (COAAT) Software Description

12. PERSONAL AUTHOR(S)
Ross, C. Glen (SAIC)

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM 87/11 TO 89/11	1990, February	185

16. SUPPLEMENTARY NOTATION
Also see Ross, C. G. (1989) Course of Action Assessment Tool (COAAT) Functional Description, ARI Research Product 90-08. Jon Fallesen, Contracting Officer's Representative

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Decision support; Tactical operations planning; Computer programs; Course of action analysis; Military tactics; Course of action comparison; (AJG)

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This report describes the software of the Course of Action Assessment Tool (COAAT). COAAT is a computerized aid for assisting tactical operations officers in the assessment of various courses of action (COA). COAAT assists the analysts in organizing critical events (CE) according to his chosen method for analyzing the battlefield, analyzing the detailed actions of each COA, and summarizing and comparing the results so the preferred COA may be identified. COAAT is written in Common Lisp for Symbolics computers operating under Genera 7.2. COAAT was conceived and developed as a prototype for a field operating system. It is configured for operation in the laboratory environment of EDDIC (Experimental Development, Demonstration, and Integration Center) at the Army Research Institute Field Unit, Fort Leavenworth, Kansas. Although the current version of COAAT is not an artificial intelligence (AI) application, exercises using COAAT may reveal opportunities for the application of AI techniques.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION	
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	Unclassified	
22a. NAME OF RESPONSIBLE INDIVIDUAL Jon Fallesen	22b. TELEPHONE (Include Area Code) (913) 684-4933	22c. OFFICE SYMBOL PERI-SL

DD Form 1473, JUN 86 *Previous editions are obsolete.*

U.S. ARMY RESEARCH INSTITUTE FOR THE BEHAVIORAL AND SOCIAL SCIENCES

A Field Operating Agency Under the Jurisdiction of the Deputy Chief of Staff for Personnel

EDGAR M. JOHNSON
Technical Director

JON W. BLADES
COL, IN
Commanding

Research accomplished under contract
for the Department of the Army

Science Applications International Corporation

Technical review by

James W. Lussier

NOTICES

DISTRIBUTION: This report has been cleared for release to the Defense Technical Information Center (DTIC) to comply with regulatory requirements. It has been given no primary distribution other than to DTIC and will be available only through DTIC or the National Technical Information Service (NTIS).

FINAL DISPOSITION: This report may be destroyed when it is no longer needed. Please do not return it to the U.S. Army Research Institute for the Behavioral and Social Sciences.

NOTE: The views, opinions, and findings in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other authorized documents.

CONTENTS

LIST OF TABLES

COURSE OF ACTION ASSESSMENT TOOL (COAAT) SOFTWARE DESCRIPTION

INTRODUCTION

The Course of Action Assessment Tool (COAAT) is a prototype decision aid designed for use by the operations planning element of a tactical headquarters from Corps to Brigade. The system is currently configured for operations in the laboratory environment of the Experimental Development, Demonstration, and Integration Center (EDDIC) facility at the Army Research Institute Field Unit, Ft. Leavenworth, KS. COAAT is written in Symbolics Common Lisp, Genera 7.2, and is installed on Symbolics 3675 and 3640 in the EDDIC facility.

In the EDDIC configuration COAAT has three operational modes:

- Training mode. This mode represents the prototype for a field operational system and input comes from the human analyst. In the laboratory environment this mode is used for training exercise participants in the use of COAAT.
- Exercise mode. This mode is used for EDDIC computer-aided exercises. In this mode the analyst inputs data to the Critical Event (CE) Assignment Module, but pre-recorded data are presented to him for CE war-gaming in Module 2. (This serves to limit the divergence between exercises thereby facilitating the comparison of exercise results.)
- Demonstration mode. This mode is used for system demonstration and provides selected pre-recorded data for each of the modules. This mode requires limited input by the demonstrator, yet still demonstrates the full system capability.

SYSTEM ARCHITECTURE

COAAT has not been established as a separate system in the Symbolics environment. COAAT is run by loading the file coaat.lisp and evaluating the function COAAT. This loads all other required files and initializes and starts the COAAT process. The calling sequence is shown in Table 1.

Table 1
Calling Sequence for Function COAAT.

```
COAAT
     LOAD  all files
     START-COAAT
          COAAT-INIT-GLOBALS
          EXPOSE-COAAT
               MAIN-TITLE
     EXIT-COAAT
          if answer = exercise
               ASK-EXERCISE-SET
```

Once the COAAT process has been started then the function INIT-COAAT is evaluated within that process to run COAAT. The calling structure of the topmost functions, initiated by INIT-COAAT, is shown in Table 2.

COAAT is an input driven system. The basic screen displays are determined by input screen description files which consist of a list of screen description commands with their appropriate arguments. Each screen description command with its arguments is a sub-list. These commands control such things as the placement of items on the screen, the acceptance of input from the user, prompts displayed to the user, and the variable name for input values. The special functions (screen description commands) MROW and MSETS provide flexibility to the system and allow for repetition of rows containing a common set of fields (MROW) and for repetition of sets of fields, rows, and/or sets (MSETS). This allows the actual screen display to be determined at a second level by user input data. Therefore,

Table 2

Top Level Calling Sequence Initiated by Function INIT-COAAT

```
INIT-COAAT
COAAT-INIT-GLOBALS

SELECT-MISSION
     EXPLAIN-START-COAAT
     MAIN-TITLE

SELECT-MOA
     EXPLAIN-MOA
     MAIN-TITLE

START-IT
     RESET-COAAT
          PROCESS-MENU-LIST
          SET-PROCESS-CHOICE
          MAIN-TITLE
     MAIN-LOOP
          if menu choice = mod-1
               START-MOD-1
               RESET-COAAT

          if menu choice = mod-2
               START-MOD-2
               RESET-COAAT

          if menu choice = mod-3
               START-MOD-3
               RESET-COAAT

          if menu choice = exit
               if *mode* = exercise
                    if SAVE-EXERCISE-DATA?
                         ARCHIVE-EXERCISE-DATA
               EXIT-COAAT
                    if answer = exercise
                         ASK-EXERCISE-SET
               if choice = continue
                    RESET-COAAT
               else
                    return to START-IT

if *mode* not equal exit, loop.
```

the screen displays are not totally predetermined by the screen description files. These files provide the basic framework for a display which can then be completely defined by the user's input.

Each of the screen description commands is also an executable function. These functions are not called directly, they are "applied" by the function COMMAND-EXECUTE. Each screen description command has a set of arguments such as; width of field, variable name, default value, etc., as appropriate for the type of field. The argument list of the executable function includes those of the screen description command augmented by a common set of arguments when "applied" by COMMAND-EXECUTE. Appendix A contains a listing of the screen description commands with a brief explanation of each.

COAAT consists of three modules: Critical Event Assignment Worksheet, Critical Event War-gaming, and Course Of Action Comparison. Two screens are used in the Critical Event War-gaming: the War-gaming Summary Sheet and the Critical Event War-gaming Worksheet. Module 3, Course of Action Comparison, uses a total of three different screens: Weighting of COA Assessment Measures, Scaling of Subjective Measures, and the COA Comparison Sheet. Appendix B contains listings of the screen description files used by COAAT. Each screen description file is read by the appropriate module control function as required and becomes the screen description table.

As shown in Table 2 the function MAIN-LOOP controls the execution of COAAT. Function MAIN-LOOP accepts the user choice from the main process menu and calls the appropriate module control routine: START-MOD-1, START-MOD-2, or START-MOD-3. The module control routines use the functions INPUT-TABLE or UPDATE-TABLE for accepting data from the user. Both of these functions use EDIT-TABLE for primary control of processing the screen description table. This function runs a loop and calls COMMAND-EXECUTE to process each command of the table until all elements have been processed. Function INIT-SCREEN is used to display the screen when all formatting information is known. INIT-SCREEN initializes the screen with data already recorded or with the default value of each field. REDISPLAY-RETURN-TABLE is used to display a screen which has been constructed based on previous input and formatting information for the display must come from the return table built during acceptance of the original input.

OPERATIONS AND MAINTENANCE

COAAT Source Files

The source code for COAAT is contained in seven separate files, these are: Coaat.lisp, Coaat-driver.lisp, Coaat-interpreter.lisp, Coaat-demons.lisp, Coaat-text.lisp, Coaat-flavor.lisp, and Coaat-sens-anal.lisp. All files are contained in the directory Mohawk:>COAAT> . The file Coaat.lisp contains the top level functions which load other files, initialize and start the system, and the definitions of all global variables (see Appendix C). File Coaat-driver.lisp, Appendix D, contains the second level functions: the main driver loop and the control functions for each of the separate modules. Coaat-interpreter.lisp, Appendix E, contains the functions which interpret the screen description files, drive the screen display, and accept input from the user. The file Coaat-demons.lisp, Appendix F, contains the functions which perform mathematical calculations using the variable data entered by the user. The file Coaat-text.lisp, Appendix G, contains functions which send text: title lines, prompts, and user instructions, to various windows or panes for display. The file Coaat-flavor.lisp, Appendix H, contains functions which define the various window flavors required by COAAT and set global variables which define the windows. The file Coaat-sens-anal.lisp contains the functions required for sensitivity analysis of the course of action (COA) assessment measures (see Appendix I). Appendix J contains an index for the function, flavor, and global variable definitions in these files.

Each file is compiled individually to create the binary version (filename.bin). The compiled (binary) versions of each file are loaded by function COAAT. If the compiled versions are not available the source file will be loaded.

Installation

As noted above, COAAT has not been configured as an internal Symbolics system. COAAT is installed by first loading the file >coaat>coaat and then evaluating the function COAAT. The function COAAT first loads each of the remaining files, as shown in Table 1. Function START-COAAT is called and the process is assigned to the SELECT Z key,

variables are initialized by function COAAT-INIT-GLOBALS, and the COAAT window is activated and exposed by function EXPOSE-COAAT. Following the window activation, the operational mode for the run is solicited via a pop-up menu using the function EXIT-COAAT. If the choice of mode is "exercise" the data set version is requested via a pop-up menu using the function ASK-EXERCISE-SET.

This completes the installation and initialization. COAAT must now be started within this process. This is accomplished by evaluating the function INIT-COAAT, refer to Table 2, from the command pane of the COAAT window.

COAAT can be run on any Symbolics machine in the EDDIC network by simply logging in as COAAT and following the run procedures above. All files will then be accessed over the network from the file server, Mohawk. Files can also be put into directory COAAT on any machine and then run directly from that machine. To operate in this manner filename paths in the files Coaat.lisp and Coaat-driver.lisp should be changed to reflect the local host name to insure the local directory is accessed.

Operational Modes

COAAT has three operational modes: Training, Exercise, and Demonstration. The exercise and demonstration modes require pre-recorded data which is then read in as appropriate. The training mode represents the prototype for an operational system; all data is entered by the user and no data is written or saved by the system. A field operating system would require selected data be saved to allow for interruptions and later continuation of the analysis, reference of previous assessments, etc. The exercise mode is designed to support EDDIC exercises. The exercise mode loads pre-recorded data at the start of Module 2. The demonstration mode is designed to require minimal input from the demonstrator, yet fully demonstrate COAAT capabilities. The demonstration mode loads selected data for all modules.

The variable *mode* is used to control variation in program execution for these modes. The choice of mode is solicited from the user by the function EXIT-COAAT using a pop-up menu. This menu offers the choices: TRAINING, EXERCISE, DEMONSTRATION,

RETURN, and EXIT COAAT. EXIT-COAAT is called by function COAAT at initial start-up of the COAAT process and by MAIN-LOOP whenever Exit is chosen from the main process menu. Function EXIT-COAAT returns "nil" when the choice is TRAINING, "exercise" when the choice is EXERCISE, "demo" when the choice is DEMONSTRATION, and "continue" when the choice is RETURN. The calling function sets the variable *mode* to the returned value except for "continue". When "continue" is returned from EXIT-COAAT to the function COAAT, *mode* is set to nil; i.e., training mode. When "continue" is returned to MAIN-LOOP, *mode* retains its current value (is not changed) and the main process menu will be displayed without initializing any data. Any mode choice will cause all data to be initialized and the mission specification menu will be displayed.

The exercise mode is designed to support the current computer-aided exercises in the EDDIC facility using the COAAT system. In this mode the exercise participant inputs critical events, which he has defined for each COA, in Module 1. Pre-recorded critical event data is read in for use in Module 2. The canned data is used to reduce the divergence in exercise solutions and thus facilitate the comparison of the solutions. The following data lists are established by the pre-recorded data: *ce-menu-list*, *ce-data-alist*, *variable-data-alist*, and mod2-summary-scrn. Two variations of the data have been created for use. Data set AB reflects COA 1 in the north along Axis A. Data set BA reflects COA 1 in the south along Axis B. When the exercise mode is chosen the data set identifier is solicited via a pop-up menu. The *ce-data-list* is not COA dependent and therefore requires only one file. All other data is contained in separate files for the two variations. Appendix K contains listings of the data files (data set AB only) used for the exercise mode in the current EDDIC exercises.

The exercise mode requires special data handling arrangements to allow the user to revisit Module 1 and access his original input data after accessing Module 2. This is accomplished by saving data from Module 1, *ce-data-alist* and *variable-data-alist*, as variables *mod1-ce-data-alist* and *mod1-data-alist* each time Module 1 is exited. When Module 2 is entered the pre-recorded critical event data is read in and presented to the participant for use in war-gaming. Data from Module 2 and 3 in *variable-data-alist* is saved to variable *exer-data-alist* each time either module is exited. This allows the user's data to be recovered upon reentry to Module 1 and other exercise data to be recovered after Module 1 has been revisited.

Data for exercise analysis is written to files each time Module 1 is exited and when the Exit choice is made from the main process menu. When Module 1 is exited the variables *ce-number-list* and *mod1-data-alist* are written to output file mod1-out.data. When the Exit choice is made from the process menu the variables *ce-menu-list* and *exer-data-alist* are written to the file exit-out.data. The user is then asked, via a pop-up menu, if the data should be saved. A response of Yes results in a call of ARCHIVE-EXERCISE-DATA to write the files to the directory m:>coaat-exercise-data. Files are renamed using the exercise ID number supplied by the user. Exercise analysis database files for dBase are also created in that same directory by the archive process.

The demonstration mode uses pre-recorded data for all modules. This data is read into the variables: *ce-menu-list*, *variable-data-alist*, and mod2-summary-scrn. The data is contained in the files: demo-ce-menu-list.file, demo-variable-data-list.file, and demo-mod2-summary.scrn.

APPENDIX A

COAAT SCREEN DESCRIPTION COMMANDS AND CALCULATION FUNCTIONS

APPENDIX A - COAAT SCREEN DESCRIPTION COMMANDS

CONTENTS

GENERAL

This appendix contains a listing of all screen description commands available in COAAT to describe the screen displays. The calculation functions used by the demon fields are also included. Each command and calculation function is listed in the syntax required for the COAAT screen description file and is followed by a brief description of its use. Current screen description files are provided in Appendix B.

SCREEN DESCRIPTION COMMANDS

CEFIELD length

This function is a specialized input function to accept the critical event (CE) number from the user. The CEFIELD screen description command has a single argument, the length (width) of the display field. CEFIELD checks for a minimum of 2 and maximum of 3 character input. The first character input must be an alphabetic character. The additional character(s) must be numeric. The resulting CE number cannot duplicate a previous input CE number for this COA, except that previously entered for this field. (No CE can occur more than once for a single COA.) If any of these input constraints are not met, a prompt is displayed explaining the required input and the entry must be repeated.

CENTER string

This function is used to center a character string on the screen display. This command has a single argument, the string to be displayed. The cursor will be advanced to the next line if it is not at the beginning of a row. When the display is complete the cursor will be advanced to a new row.

DEMON length variable function &rest args

This function is used to display a value calculated from other input data. The arguments to this command are: the length of the field to be displayed, the variable name of the calculated value, the function to be applied in the calculation, and the variable names (&rest args) of the arguments for the calculation.

The DEMON function assumes that the variable and its arguments, unless global, are CE specific. The variable names are converted to CE specific by appending the current CE number. Global variables, delimited by *, in the argument list are not changed. This function requires context information to access the variable data.

DFIELD length default variable

This function is used to display data entered into COAAT on previous screens. Data are recovered from *variable-data-alist*. If no data are found, the default value is displayed. The arguments to this command are: length of the display field, the default value for display, and the variable name of the data to be displayed.

FFIELD length string

This function is used to display a fixed character string at the current cursor position. The FFIELD screen description command has two arguments: length of the display field and the string to be displayed.

IFIELD length type &optional default variable

This function is used to accept input data from the user. The IFIELD screen description command has two required arguments: length (width) of the display field and the type of data to be accepted as input. Two optional arguments are also available; the default value for display and the variable name to represent the data input.

The type of data allowed for input is identified by the type argument. Type arguments which may be used are: any, alpha, decimal, numeric, and natural.

All non-global variables are assumed to be CE specific and their names are modified by adding "-" and the current global CE number (*ce-num*) to the variable argument. If variable is enclosed by "*" to denote a global variable, it is used directly. If no variable data exist, the default value is displayed. If the input variable name is either ce-type, objective, or comment, then a second variable is created by appending the variable argument with the base CE number (axis and sequence number). This name and value are then recorded in *ce-data-alist*.

LEFT string &optional type variable

This function is used to display a character string at the left margin of the screen. Optionally, the string may be followed by either a numeric or alphabetic character determined in sequence within a context. The LEFT screen description command has three arguments: a string to be displayed left justified on the screen, the type of sequence character to be generated (numeric or alpha), and the name of the variable which will have the value of the string appended with the sequence character. The type and variable name are optional arguments. If the cursor is not at the left margin it will be advanced to a new line for the display. The cursor will also be advanced to a new line following the display.

MAA-PFIELD length

This function is a specialized version of PFIELD designed to prompt for and accept input of the designated main attack axis for a COA. The MAA-PFIELD screen description command has a single argument, length of the display field. The prompt called by this function is ATK-AXIS-PROMPT. The variable saved by this function is "main-atk-axis-#", where # is the COA number.

MFIELD length menu-list variable

This function is used to accept input from the user through selection from a pop-up menu. The MFIELD screen description command has three arguments: length of the display field, a list of the items to be presented on the pop-up menu, and the variable name to be used for the input data. A specialized function named "CHOOSE-variable name" (e.g., CHOOSE-CE-TYPE) must exist for each variable using MFIELD. That function handles any special menu requirements, putting up the menu, and returning the selected value.

MOA-PFIELD length

This function is a specialized version of PFIELD designed to prompt for and accept input of the name for the avenue/belt/box (method of analysis). The MOA-PFIELD screen description command has a single argument, length of the display field. The prompt function used is MOA-NAME-PROMPT. The name of the variable is constructed as "#1-type-#2", where #1 is the COA number, type is the MOA type (avenue/belt/box), and #2 is the MOA number within this COA.

MROW name description &rest commands

This function provides for a collection of fields to make up a repeatable row of the display. The MROW screen description command has three arguments: name, the name of this collection of fields; description, a string which will be displayed in a pop-up menu at the end of the row; and commands, a variable number of subordinate screen description commands which are included in this row. Any of the screen commands, except MSETS, may be included in the subordinate commands. When all commands of the row have been processed a pop-up menu is provided listing the description arguments of all rows and/or sets (see MSETS below) included in the current top-level row or set. The choice from this menu determines the next row or set to be processed or completion of the current top-level row or set.

MSETS name description &rest commands

This function provides for aggregations of screen description commands into repeatable sets. The MSETS screen description command has three arguments: name, the name of this collection of fields, rows, and/or sets; description, a string which will be displayed in a pop-up menu for the user's choice at the end of the set; and commands, a variable number of subordinate screen description commands included in this set. Any of the screen commands, including MSETS may be included in the set of subordinate commands. The MSETS command provides flexibility for the screen definition and allows the actual display to be determined by input rather than be fixed in advance. When all commands of the set have been processed a pop-up menu is provided listing the description arguments of all rows and/or sets (see MROW above) included in the current top-level row or set. The choice from this menu determines the next row or set to be processed or completion of the current top-level row or set.

NEWLINE &optional (number 1)

This function is used to advance the cursor to a new line. The NEWLINE screen description command has a single optional argument, the number of lines to advance. The default value is 1.

PFIELD length type default variable

This function is used to provide a prompt for input with a pop-up prompt window and accept input from the user. The PFIELD screen description command has four arguments: length of the display field, the type of data to be accepted as input, the default value for display, and the variable name for the data. This function is a specialized version of IFIELD (see IFIELD) which causes a pop-up prompt to be displayed to the user. The prompt to be displayed is determined from the variable name for the field. The prompt must be contained in a function "variable name-PROMPT". The variable name is considered to be global if it is delimited by asterisks (*), otherwise the variable is assumed to be specific to the current critical event (CE) and the CE number is appended to the variable name for storage in *variable-data-alist*.

PTFIELD length default variable

This is a specialized function used to prompt for and accept the scale value for subjective COA comparison factors in Module 3. The PTFIELD screen description command has three arguments: length (width) of the display field, the default value for display, and the variable name for the data. This function is a specialized version of PFIELD (see PFIELD).

The input variable names for PTFIELD are assumed to end in "-#-scale", where # is the COA number. These last eight characters are deleted from the name and the prefix "scale-" is added to create the name of the prompt to be called.

SIFIELD length &optional default variable

This function is a specialized version of IFIELD used to accept signed (+/-) numerical input data. See IFIELD above. The primary differences from IFIELD are: the only input type allowed by SIFIELD is natural; if the first character entered is not "-", a leading blank is inserted to insure proper alignment of column entries; and SIFIELD has no special handling requirement for variables CE type, objective, and comment since alphanumeric entries are not accepted.

The SIFIELD screen description command has a three arguments: length of the display field, the default value for the display, and the variable name to be used for the data input. The default value and variable name are optional arguments.

TFIELD length string &optional type variable

This function used to display a text entry at the current cursor position. Optionally, the field may be followed by either a numeric or alphabetic character drawn in sequence within the current context. Special characters may be used within the text to create multiple line output (%) or to denote the name of a variable to be included in the text (&). Each character of the text input is evaluated. If the character is "%", the cursor is advanced one line and returned to the original x-coordinate. If the character is "&", all characters before the next "&" are interpreted as a variable name and when the second "&" is encountered the variable is recovered and displayed. Other characters are displayed as they are evaluated. If type and variable are included in the argument list a character of the defined type (numeric or alphabetic) will then be displayed in sequence within the current context. If the variable is *coa*, the moa-number is initialized to zero.

The TFIELD screen description command has four arguments: the length (width) of the field, a string to be displayed (may be multiple lines), the type of sequence character to be generated (numeric or alpha); and the name of the variable to get the value of the string appended with the sequence character. The type and variable name are optional arguments.

WSDEMON length variable function &rest args

This function is used to display a value calculated from other input data. This is a special case of the DEMON function which does not require context information. The

WSDEMON screen description command has three arguments: the width of the field to be displayed, the variable name of the calculated variable, and the function to be applied for the calculation. Additionally, the variable names of the arguments for the calculation must be furnished.

The WSDEMON function assumes that the variable and its arguments, unless global, are CE specific. The variable names are converted to CE specific by appending the current CE number. Global variables, delimited by *, in the argument list are not changed. This function does not require context information; all variables are assumed to be contained in *variable-data-alist*.

XDEMON length variable function &rest args

This function is used to display a value calculated from other input data. This function is a special case of the DEMON function which uses variables names exactly as input. The XDEMON screen description command has three arguments: the width of the field to be displayed, the variable name of the calculated variable, and the function to be applied for the calculation. Additionally, the variable names of the arguments for the calculation must be furnished.

CALCULATION FUNCTIONS FOR USE IN DEMON FIELDS

CALC-EC-EQUIP-SCALE-VAL variable

This function is used in a demon field to calculate the scale value for enemy equipment casualties. The scale value is based on the percent of the enemy authorized strength which is lost. Authorized strength is currently established within the function and should be made an input variable at some time. Variable is the name for the enemy equipment loss data.

CALC-EC-PERS-SCALE-VAL variable

This function is used in a demon field to calculate the scale value for enemy personnel casualties. The scale value is based on the percent of the enemy authorized strength which is lost. Authorized strength is currently established within the function and should be made an input variable at some time. Variable is the name for the enemy personnel casualty data.

CALC-FC-EQUIP-SCALE-VAL variable

This function is used in a demon field to calculate the scale value for friendly equipment casualties. The scale value is based on the percent of the division's authorized strength which is lost. Division authorized strength is currently established within the function and should be made an input variable at some time. Variable is the name for the friendly equipment loss data.

CALC-FC-PERS-SCALE-VAL variable

This function is used in a demon field to calculate the scale value for friendly personnel casualties. The scale value is based on the percent of the division's authorized strength which is lost. Division authorized strength is currently established within the function and should be made an input variable at some time. Variable is the name for the friendly personnel casualty data.

CALC-FEBA-MVMT-SCALE-VAL variable

This function is used in a demon field to calculate the scale value for FEBA movement. The scale value is based on the distance moved along the main attack axis. Need to establish a procedure for input of these values which are now hardwired in the function. Variable is the name for the FEBA movement data.

CALC-RE-AMMO-SCALE-VAL variable

This function is used in a demon field to calculate the scale value for ammunition consumed. The scale value is based on the percent of the division's authorized basic load which is used. Division authorized basic load is currently established within the function and should be made an input variable at some time. Variable is the name for the data on ammunition expended.

CALC-RE-POL-SCALE-VAL variable

This function is used in a demon field to calculate the scale value for POL consumption. The scale value is based on the percent of the division's authorized quantity which is consumed. Division authorized quantity is currently established within the function and should be made an input variable at some time. Variable is the name for the data on POL expended.

CALC-TIME-RQD-SCALE-VAL variable

This function is used in a demon field to calculate the scale value for time required for the operation. The scale value is based on the total time required on the main axis of attack. Need to establish a procedure for input of these values which are now hardwired in the function. Variable is the name for the time required (duration) data.

INT-TOTAL &rest variables

This function is used in a demon field to calculate the integer total of a set of variables given the names of the variables to be included.

TOTAL &rest variables

This function is used in a demon field to calculate the total of a set of variables given the names of the variables to be included.

WEIGHTED-VALUE &rest value-weight

This function is used in a demon field to calculate the sum of a set of weighted values given a list of variable names for value-weight pairs.

APPENDIX B

COAAT SCREEN DESCRIPTION FILES

>COAAT>COAAT-MOD-1-OFFENSE.SCRN - 10/10/89

```
;;;... -*- Mode: LISP; Syntax: Common-Lisp; Package: USER; Base: 10 -*-...
;*************************************************
;*        COAAT - MODULE 1 SCREEN DATA           *
;*        Critical Event Assignment Worksheet    *
;*************************************************

( (NEWLINE 1)
  (FFIELD 20 " ")
  (TFIELD 17 "CE Assignment")
  (FFIELD 3 " ")
  (FFIELD 20 "CE Type")
  (FFIELD 3 " ")
  (FFIELD 20 "Objective")
  (FFIELD 3 " ")
  (FFIELD 20 "Comments")
  (NEWLINE 2)
  (MSETS outer " COURSE OF ACTION "
      (FFIELD 20 " ")
      (TFIELD 6 "COA-" numeric *coa*)
      (FFIELD 1 " ")
      (FFIELD 19 "Main Attack = Axis-")
      (MAA-PFIELD 2)
      (NEWLINE 1)
      (MSETS inner " &*moa-type*& "
         (FFIELD 24 " ")
         (TFIELD 8 "&*moa-type*&-")
         (MOA-PFIELD 8)
         (NEWLINE 1)
         (MROW only " CRITICAL EVENT "
         (FFIELD 28 " ")
         (CEFIELD 7)
         (FFIELD 3 " ")
         (MFIELD 20  ("Passage of Lines    "                    ; CE types for offense
                      "Penetrate En 1st Ech"
                      "Penetrate En 2nd Ech"
                      "Breach Obstacle Belt"
                      "Cross River         "
                      "Seize Objective     "
                      "Seize Key Terrain   "
                      "Seize Town          "
                      "Defeat Enemy CATK   "
                      "Destroy Enemy Force "
                      "Capture Enemy Force "
                      "Bypass Enemy Force  "
                      "Seize Beachhead     "
                      "Fix En in Position  ") ce-type)
         (FFIELD 3 " ")
         (IFIELD 20 any "" objective)
         (FFIELD 3 " ")
         (IFIELD 20 any "" comment)
         (NEWLINE 1))
      (NEWLINE 1))))
```

>COAAT>COAAT-MOD-2.SCRN - 10/6/88

```
;;;... -*- Mode: LISP; Syntax: Common-Lisp; Package: User; Base: 10  -*- ...
;*****************************************************************
;*        COAAT - MODULE 2 SCREEN DATA                           *
;*        Wargaming Summary Data Sheet Header                    *
;*        The table is completed by the function CREATE-MOD2-TABLE  *
;*        based on data entered in module 1.                     *
;*****************************************************************

((CENTER "WARGAMING SUMMARY SHEET")
 (NEWLINE 1)
 (FFIELD 14 " ")
 (FFIELD 21 "   CE Type")
 (FFIELD 20 "   Objective")
 (TFIELD 14 "   Friendly%  Casualties% Pers  Equip")
 (TFIELD 14 "    Enemy  %  Casualties% Pers  Equip")
 (TFIELD 12 "  Percent%  Expended%  POL  Ammo")
 (TFIELD 8  "  FEBA%  Mvmt%  (km)")
 (TFIELD 8  "  Time%  Rqd%  (hrs)")
 (NEWLINE 1) )
```

>COAAT>COAAT-MOD-2A.SCRN - 4/28/89

```
;;;... -*- Mode: LISP; Syntax: Common-Lisp; Package: User; Base: 10  -*- ...
;*************************************************
;*        COAAT - MODULE 2A SCREEN DATA          *
;*        Critical Event Wargaming Worksheet     *
;*************************************************

((NEWLINE 1)
 (FFIELD 20 " ")
 (FFIELD 4 "COA ")
 (TFIELD 2 "&*coa*&")
 (FFIELD 2 ", ")
 (TFIELD 7 "&*ce-num*&")
 (FFIELD 3 " - ")
 (TFIELD 20 "&*type*&")
 (FFIELD 3 " - ")
 (TFIELD 20 "&*objective*&")
 (NEWLINE 2)
 (FFIELD 4 " ")
 (FFIELD 5 "Phase")
 (FFIELD 12 " ")
 (TFIELD 14 "  Friendly% Casualties%Pers  Equip")
 (TFIELD 14 "   Enemy  % Casualties%Pers  Equip")
 (TFIELD 12 "  Percent% Expended% POL  Ammo")
 (TFIELD  8 "  FEBA  %  Mvmt%  (km)")
 (TFIELD  8 "  Time  %  Rqd%  (hrs)")
 (FFIELD 20 "  Comments")
 (NEWLINE 2)
 (MSETS outer ""
       (FFIELD 2 " ")
       (FFIELD 2 "A")
       (FFIELD 15 "(Preparation)")
       (MROW A ""
          (FFIELD 2 " ")
          (IFIELD 5 natural "0" frnd-cas-per-a)
          (FFIELD 2 " ")
          (IFIELD 5 natural "0" frnd-cas-eq-a)
          (FFIELD 2 " ")
          (IFIELD 5 natural "0" en-cas-per-a)
          (FFIELD 2 " ")
          (IFIELD 5 natural "0" en-cas-eq-a)
          (FFIELD 3 " ")
          (IFIELD 4 natural "0" pol-ex-a)
          (FFIELD 2 " ")
          (IFIELD 4 natural "0" ammo-ex-a)
          (FFIELD 3 " ")
          (IFIELD 5 natural "0" distance-a)
          (FFIELD 3 " ")
          (IFIELD 5 decimal "0" duration-a)
          (FFIELD 4 " ")
          (IFIELD 20 any " " comment-a)
          (NEWLINE 1))
       (FFIELD 2 " ")
       (FFIELD 2 "B")
```

```
(FFIELD 15 "(The Event)")
(MROW B ""
   (FFIELD 2 " ")
   (IFIELD 5 natural "0" frnd-cas-per-b)
   (FFIELD 2 " ")
   (IFIELD 5 natural "0" frnd-cas-eq-b)
   (FFIELD 2 " ")
   (IFIELD 5 natural "0" en-cas-per-b)
   (FFIELD 2 " ")
   (IFIELD 5 natural "0" en-cas-eq-b)
   (FFIELD 3 " ")
   (IFIELD 4 natural "0" pol-ex-b)
   (FFIELD 2 " ")
   (IFIELD 4 natural "0" ammo-ex-b)
   (FFIELD 3 " ")
   (IFIELD 5 natural "0" distance-b)
   (FFIELD 3 " ")
   (IFIELD 5 decimal "0" duration-b)
   (FFIELD 4 " ")
   (IFIELD 20 any " " comment-b)
   (NEWLINE 1))
(FFIELD 2 " ")
(FFIELD 2 "C")
(FFIELD 15 "(Consolidation)")
(MROW C ""
   (FFIELD 2 " ")
   (IFIELD 5 natural "0" frnd-cas-per-c)
   (FFIELD 2 " ")
   (IFIELD 5 natural "0" frnd-cas-eq-c)
   (FFIELD 2 " ")
   (IFIELD 5 natural "0" en-cas-per-c)
   (FFIELD 2 " ")
   (IFIELD 5 natural "0" en-cas-eq-c)
   (FFIELD 3 " ")
   (IFIELD 4 natural "0" pol-ex-c)
   (FFIELD 2 " ")
   (IFIELD 4 natural "0" ammo-ex-c)
   (FFIELD 3 " ")
   (IFIELD 5 natural "0" distance-c)
   (FFIELD 3 " ")
   (IFIELD 5 decimal "0" duration-c)
   (FFIELD 4 " ")
   (IFIELD 20 any " " comment-c)
   (NEWLINE 2))
(FFIELD 11 " ")
(FFIELD 5 "Total")
```

```
(MROW total ""
   (FFIELD 2 " ")
   (WSDEMON 6 fc-pers INT-TOTAL frnd-cas-per-a frnd-cas-per-b frnd-cas-per-c)
   (FFIELD 1 " ")
   (WSDEMON 6  fc-equip INT-TOTAL frnd-cas-eq-a frnd-cas-eq-b frnd-cas-eq-c)
   (FFIELD 1 " ")
   (WSDEMON 6  ec-pers INT-TOTAL en-cas-per-a en-cas-per-b en-cas-per-c)
   (FFIELD 1 " ")
   (WSDEMON 6  ec-equip INT-TOTAL en-cas-eq-a en-cas-eq-b en-cas-eq-c)
   (FFIELD 2 " ")
   (WSDEMON 6  re-pol INT-TOTAL pol-ex-a pol-ex-b pol-ex-c)
   ; (FFIELD 0 " ")
   (WSDEMON 6  re-ammo INT-TOTAL ammo-ex-a ammo-ex-b ammo-ex-c)
   (FFIELD 1 " ")
   (WSDEMON 6  feba-mvmt INT-TOTAL distance-a distance-b distance-c)
   (FFIELD 3 " ")
   (WSDEMON 6  time-rqd TOTAL duration-a duration-b duration-c)
   (NEWLINE 1))))
```

>COAAT>COAAT-MOD-3A.SCRN - 4/06/89

```
;;;... -*- Mode: LISP; Syntax: Common-Lisp; Package: User; Base: 10  -*- ...
;*************************************************************
;*        COAAT - MODULE 3A SCREEN DATA                      *
;*        Input Weights for all Comparison Measures          *
;*************************************************************

((NEWLINE 1)
 (FFIELD 6 " ")
 (FFIELD 8 "MEASURES")
 (FFIELD 16 " ")
 (FFIELD 7 "WEIGHTS")
 (NEWLINE 2)
 (FFIELD 2 " ")
 (FFIELD 22 "OBJECTIVE (WAR-GAMING)")
 (NEWLINE 1)
 (FFIELD 4 " ")
 (FFIELD 25 "Friendly Cas, Pers    ")
 (FFIELD 2 " ")
 (IFIELD 5 decimal "0" *fc-pers-weight*)
 (NEWLINE 1)
 (FFIELD 4 " ")
 (FFIELD 25 "Friendly Loss, Equip ")
 (FFIELD 2 " ")
 (IFIELD 5 decimal "0" *fc-equip-weight*)
 (NEWLINE 1)
 (FFIELD 4 " ")
 (FFIELD 25 "Enemy Cas, Pers      " )
 (FFIELD 2 " ")
 (IFIELD 5 decimal "0" *ec-pers-weight*)
 (NEWLINE 1)
 (FFIELD 4 " ")
 (FFIELD 25 "Enemy Loss, Equip    " )
 (FFIELD 2 " ")
 (IFIELD 5 decimal "0" *ec-equip-weight*)
 (NEWLINE 1)
 (FFIELD 4 " ")
 (FFIELD 25 "POL Expended         " )
 (FFIELD 2 " ")
 (IFIELD 5 decimal "0" *re-pol-weight*)
 (NEWLINE 1)
 (FFIELD 4 " ")
 (FFIELD 25 "Ammo Expended        " )
 (FFIELD 2 " ")
 (IFIELD 5 decimal "0" *re-ammo-weight*)
 (NEWLINE 1)
 (FFIELD 4 " ")
 (FFIELD 25 "FEBA Mvmt (km)       " )
 (FFIELD 2 " ")
 (IFIELD 5 decimal "0" *feba-mvmt-weight*)
 (NEWLINE 1)
 (FFIELD 4 " ")
 (FFIELD 25 "Time Required (hrs)  " )
 (FFIELD 2 " ")
```

```
(IFIELD 5 decimal "0" *time-rqd-weight*)
(NEWLINE 3)
(FFIELD 2 " ")
(FFIELD 10 "SUBJECTIVE")
(NEWLINE 1)
(FFIELD 4 " ")
(FFIELD 25 "Accomplish Mission       " )
(FFIELD 2 " ")
(IFIELD 5 decimal "0" *subj-a-weight*)
(NEWLINE 1)
(FFIELD 4 " ")
(FFIELD 25 "Effective Use of Assets  " )
(FFIELD 2 " ")
(IFIELD 5 decimal "0" *subj-b-weight*)
(NEWLINE 1)
(FFIELD 4 " ")
(FFIELD 25 "Flexibility              " )
(FFIELD 2 " ")
(IFIELD 5 decimal "0" *subj-c-weight*)
(NEWLINE 1)
(FFIELD 4 " ")
(FFIELD 25 "Facilitate Future Ops    " )
(FFIELD 2 " ")
(IFIELD 5 decimal "0" *subj-d-weight*)
(NEWLINE 1)
(FFIELD 4 " ")
(FFIELD 25 "Risk                     " )
(FFIELD 2 " ")
(IFIELD 5 decimal "0" *subj-e-weight*)
(NEWLINE 1)
(FFIELD 4 " ")
(IFIELD 25 any "User Choice 1 " *subj-f-name*)
(FFIELD 2 " ")
(IFIELD 5 decimal "0" *subj-f-weight*)
(NEWLINE 1)
(FFIELD 4 " ")
(IFIELD 25 any "User Choice 2 " *subj-g-name*)
(FFIELD 2 " ")
(IFIELD 5 decimal "0" *subj-g-weight*)
(NEWLINE 1)
(FFIELD 4 " ")
(IFIELD 25 any "User Choice 3 " *subj-h-name*)
(FFIELD 2 " ")
(IFIELD 5 decimal "0" *subj-h-weight*)
(NEWLINE 1)   )
```

>COAAT>COAAT-MOD-3B.SCRN - 10/10/89

```
;;;... -*- Mode: LISP; Syntax: Common-Lisp; Package: User; Base: 10  -*- ...
;*********************************************************
;*        COAAT - MODULE 3B SCREEN DATA                  *
;*        Input Scale Values for Subjective Measures     *
;*********************************************************

((NEWLINE 1)
 (FFIELD 6 " ")
 (FFIELD 8 "MEASURES")
 (FFIELD 16 " ")
 (FFIELD 7 "WEIGHTS")
 (FFIELD 12 " ")
 (FFIELD 5 "COA-1")
 (FFIELD 15 " ")
 (FFIELD 5 "COA-2")
 (NEWLINE 1)
 (FFIELD 44 " ")
 (FFIELD 16 "SCALED  WEIGHTED")
 (FFIELD 4 " ")
 (FFIELD 16 "SCALED  WEIGHTED")
 (NEWLINE 1)
 (FFIELD 2 " ")
 (FFIELD 22 "OBJECTIVE (WAR-GAMING)")
 (NEWLINE 1)
 (FFIELD 4 " ")
 (FFIELD 25 "Friendly Cas, Pers   ")
 (FFIELD 2 " ")
 (DFIELD 5 "" *fc-pers-weight*)
 (NEWLINE 1)
 (FFIELD 4 " ")
 (FFIELD 25 "Friendly Loss, Equip ")
 (FFIELD 2 " ")
 (DFIELD 5 "" *fc-equip-weight*)
 (NEWLINE 1)
 (FFIELD 4 " ")
 (FFIELD 25 "Enemy Cas, Pers      " )
 (FFIELD 2 " ")
 (DFIELD 5 "" *ec-pers-weight*)
 (NEWLINE 1)
 (FFIELD 4 " ")
 (FFIELD 25 "Enemy Loss, Equip    " )
 (FFIELD 2 " ")
 (DFIELD 5 "" *ec-equip-weight*)
 (NEWLINE 1)
 (FFIELD 4 " ")
 (FFIELD 25 "POL Expended         " )
 (FFIELD 2 " ")
 (DFIELD 5 "" *re-pol-weight*)
 (NEWLINE 1)
 (FFIELD 4 " ")
 (FFIELD 25 "Ammo Expended        " )
 (FFIELD 2 " ")
 (DFIELD 5 "" *re-ammo-weight*)
```

```
(NEWLINE 1)
(FFIELD 4 " ")
(FFIELD 25 "FEBA Mvmt (km)       " )
(FFIELD 2 " ")
(DFIELD 5 "" *feba-mvmt-weight*)
(NEWLINE 1)
(FFIELD 4 " ")
(FFIELD 25 "Time Required (hrs)  " )
(FFIELD 2 " ")
(DFIELD 5 "" *time-rqd-weight*)
(NEWLINE 1)
(NEWLINE 2)
(FFIELD 2 " ")
(FFIELD 10 "SUBJECTIVE")
(NEWLINE 1)
(FFIELD 4 " ")
(FFIELD 25 "Accomplish Mission       " )
(FFIELD 2 " ")
(DFIELD 5 "" *subj-a-weight*)
(FFIELD 10 " ")
(PTFIELD 2 "" subj-a-1-scale)
(FFIELD 5 " ")
(XDEMON 6 subj-a-1-wtd-score WEIGHTED-VALUE subj-a-1-scale *subj-a-weight*)
(FFIELD 7 " ")
(PTFIELD 2 "" subj-a-2-scale)
(FFIELD 5 " ")
(XDEMON 6 subj-a-2-wtd-score WEIGHTED-VALUE subj-a-2-scale *subj-a-weight*)
(NEWLINE 1)
(FFIELD 4 " ")
(FFIELD 25 "Effective Use of Assets  " )
(FFIELD 2 " ")
(DFIELD 5 "" *subj-b-weight*)
(FFIELD 10 " ")
(PTFIELD 2 "" subj-b-1-scale)
(FFIELD 5 " ")
(XDEMON 6 subj-b-1-wtd-score WEIGHTED-VALUE subj-b-1-scale *subj-b-weight*)
(FFIELD 7 " ")
(PTFIELD 2 "" subj-b-2-scale)
(FFIELD 5 " ")
(XDEMON 6 subj-b-2-wtd-score WEIGHTED-VALUE subj-b-2-scale *subj-b-weight*)
(NEWLINE 1)
(FFIELD 4 " ")
(FFIELD 25 "Flexibility              " )
(FFIELD 2 " ")
(DFIELD 5 "" *subj-c-weight*)
(FFIELD 10 " ")
(PTFIELD 2 "" subj-c-1-scale)
(FFIELD 5 " ")
(XDEMON 6 subj-c-1-wtd-score WEIGHTED-VALUE subj-c-1-scale *subj-c-weight*)
(FFIELD 7 " ")
(PTFIELD 2 "" subj-c-2-scale)
(FFIELD 5 " ")
(XDEMON 6 subj-c-2-wtd-score WEIGHTED-VALUE subj-c-2-scale *subj-c-weight*)
```

```
(NEWLINE 1)
(FFIELD 4 " ")
(FFIELD 25 "Facilitate Future Ops     " )
(FFIELD 2 " ")
(DFIELD 5 "" *subj-d-weight*)
(FFIELD 10 " ")
(PTFIELD 2 "" subj-d-1-scale)
(FFIELD 5 " ")
(XDEMON 6 subj-d-1-wtd-score WEIGHTED-VALUE subj-d-1-scale *subj-d-weight*)
(FFIELD 7 " ")
(PTFIELD 2 "" subj-d-2-scale)
(FFIELD 5 " ")
(XDEMON 6 subj-d-2-wtd-score WEIGHTED-VALUE subj-d-2-scale *subj-d-weight*)
(NEWLINE 1)
(FFIELD 4 " ")
(FFIELD 25 "Risk                    " )
(FFIELD 2 " ")
(DFIELD 5 "" *subj-e-weight*)
(FFIELD 10 " ")
(PTFIELD 2 "" subj-e-1-scale)
(FFIELD 5 " ")
(XDEMON 6 subj-e-1-wtd-score WEIGHTED-VALUE subj-e-1-scale *subj-e-weight*)
(FFIELD 7 " ")
(PTFIELD 2 "" subj-e-2-scale)
(FFIELD 5 " ")
(XDEMON 6 subj-e-2-wtd-score WEIGHTED-VALUE subj-e-2-scale *subj-e-weight*)
(NEWLINE 1)
(FFIELD 4 " ")
(DFIELD 25 " " *subj-f-name*)
(FFIELD 2 " ")
(DFIELD 5 "" *subj-f-weight*)
(FFIELD 10 " ")
(PTFIELD 2 "" subj-f-1-scale)
(FFIELD 5 " ")
(XDEMON 6 subj-f-1-wtd-score WEIGHTED-VALUE subj-f-1-scale *subj-f-weight*)
(FFIELD 7 " ")
(PTFIELD 2 "" subj-f-2-scale)
(FFIELD 5 " ")
(XDEMON 6 subj-f-2-wtd-score WEIGHTED-VALUE subj-f-2-scale *subj-f-weight*)
(NEWLINE 1)
(FFIELD 4 " ")
(DFIELD 25 " " *subj-g-name*)
(FFIELD 2 " ")
(DFIELD 5 "" *subj-g-weight*)
(FFIELD 10 " ")
(PTFIELD 2 "" subj-g-1-scale)
(FFIELD 5 " ")
(XDEMON 6 subj-g-1-wtd-score WEIGHTED-VALUE subj-g-1-scale *subj-g-weight*)
(FFIELD 7 " ")
(PTFIELD 2 "" subj-g-2-scale)
(FFIELD 5 " ")
(XDEMON 6 subj-g-2-wtd-score WEIGHTED-VALUE subj g-2-scale *subj-g-weight*)
(NEWLINE 1)
```

```
(FFIELD 4 " ")
(DFIELD 25 " " *subj-h-name*)
(FFIELD 2 " ")
(DFIELD 5 "" *subj-h-weight*)
(FFIELD 10 " ")
(PTFIELD 2 "" subj-h-1-scale)
(FFIELD 5 " ")
(XDEMON 6 subj-h-1-wtd-score WEIGHTED-VALUE subj-h-1-scale *subj-h-weight*)
(FFIELD 7 " ")
(PTFIELD 2 "" subj-h-2-scale)
(FFIELD 5 " ")
(XDEMON 6 subj-h-2-wtd-score WEIGHTED-VALUE subj-h-2-scale *subj-h-weight*)
(NEWLINE 1)
(FFIELD 15 " ")
(FFIELD 8 "SUBTOTAL")
(FFIELD 28 " ")
(XDEMON 8 subj-subtot-1 TOTAL subj-a-1-wtd-score subj-b-1-wtd-score
                              subj-c-1-wtd-score subj-d-1-wtd-score subj-e-1-wtd-score
                              subj-f-1-wtd-score subj-g-1-wtd-score subj-h-1-wtd-score)
(FFIELD 12 " ")
(XDEMON 8 subj-subtot-2 TOTAL subj-a-2-wtd-score subj-b-2-wtd-score
                              subj-c-2-wtd-score subj-d-2-wtd-score subj-e-2-wtd-score
                              subj-f-2-wtd-score subj-g-2-wtd-score subj-h-2-wtd-score)
(NEWLINE 1)  )
```

>COAAT>COAAT-MOD-3C.SCRN - 10/10/89

```
;;;... -*- Mode: LISP; Syntax: Common-Lisp; Package: User; Base: 10  -*- ...
;*****************************************************************************
;*        COAAT - MODULE 3C SCREEN DATA                                      *
;*        Course of Action Comparison, All factors with weighted values      *
;*****************************************************************************

((NEWLINE 1)
 (FFIELD 6 " ")
 (FFIELD 8 "MEASURES")
 (FFIELD 16 " ")
 (FFIELD 7 "WEIGHTS")
 (FFIELD 12 " ")
 (FFIELD 5 "COA-1")
 (FFIELD 15 " ")
 (FFIELD 5 "COA-2")
 (NEWLINE 1)
 (FFIELD 44 " ")
 (FFIELD 16 "SCALED  WEIGHTED")
 (FFIELD 4 " ")
 (FFIELD 16 "SCALED  WEIGHTED")
 (NEWLINE 1)
 (FFIELD 2 " ")
 (FFIELD 22 "OBJECTIVE (WAR-GAMING)")
 (NEWLINE 1)
 (FFIELD 4 " ")
 (FFIELD 25 "Friendly Cas, Pers    ")
 (FFIELD 2 " ")
 (DFIELD 5 "" *fc-pers-weight*)
 (FFIELD 10 " ")
 (DFIELD 2 "" fc-pers-1-scale)
 (FFIELD 5 " ")
 (XDEMON 6 fc-pers-1-wtd-score WEIGHTED-VALUE fc-pers-1-scale *fc-pers-weight*)
 (FFIELD 7 " ")
 (DFIELD 2 "" fc-pers-2-scale)
 (FFIELD 5 " ")
 (XDEMON 6 fc-pers-2-wtd-score WEIGHTED-VALUE fc-pers-2-scale *fc-pers-weight*)
 (NEWLINE 1)
 (FFIELD 4 " ")
 (FFIELD 25 "Friendly Loss, Equip ")
 (FFIELD 2 " ")
 (DFIELD 5 "" *fc-equip-weight*)
 (FFIELD 10 " ")
 (DFIELD 2 "" fc-equip-1-scale)
 (FFIELD 5 " ")
 (XDEMON 6 fc-equip-1-wtd-score WEIGHTED-VALUE fc-equip-1-scale *fc-equip-weight*)
 (FFIELD 7 " ")
 (DFIELD 2 "" fc-equip-2-scale)
 (FFIELD 5 " ")
 (XDEMON 6 fc-equip-2-wtd-score WEIGHTED-VALUE fc-equip-2-scale *fc-equip-weight*)
 (NEWLINE 1)
 (FFIELD 4 " ")
 (FFIELD 25 "Enemy Cas, Pers      " )
 (FFIELD 2 " ")
```

```
(DFIELD 5 "" *ec-pers-weight*)
(FFIELD 10 " ")
(DFIELD 2 "" ec-pers-1-scale)
(FFIELD 5 " ")
(XDEMON 6 ec-pers-1-wtd-score WEIGHTED-VALUE ec-pers-1-scale *ec-pers-weight*)
(FFIELD 7 " ")
(DFIELD 2 "" ec-pers-2-scale)
(FFIELD 5 " ")
(XDEMON 6 ec-pers-2-wtd-score WEIGHTED-VALUE ec-pers-2-scale *ec-pers-weight*)
(NEWLINE 1)
(FFIELD 4 " ")
(FFIELD 25 "Enemy Loss, Equip    " )
(FFIELD 2 " ")
(DFIELD 5 "" *ec-equip-weight*)
(FFIELD 10 " ")
(DFIELD 2 "" ec-equip-1-scale)
(FFIELD 5 " ")
(XDEMON 6 ec-equip-1-wtd-score WEIGHTED-VALUE ec-equip-1-scale *ec-equip-weight*)
(FFIELD 7 " ")
(DFIELD 2 "" ec-equip-2-scale)
(FFIELD 5 " ")
(XDEMON 6 ec-equip-2-wtd-score WEIGHTED-VALUE ec-equip-2-scale *ec-equip-weight*)
(NEWLINE 1)
(FFIELD 4 " ")
(FFIELD 25 "POL Expended         " )
(FFIELD 2 " ")
(DFIELD 5 "" *re-pol-weight*)
(FFIELD 10 " ")
(DFIELD 2 "" re-pol-1-scale)
(FFIELD 5 " ")
(XDEMON 6 re-pol-1-wtd-score WEIGHTED-VALUE re-pol-1-scale *re-pol-weight*)
(FFIELD 7 " ")
(DFIELD 2 "" re-pol-2-scale)
(FFIELD 5 " ")
(XDEMON 6 re-pol-2-wtd-score WEIGHTED-VALUE re-pol-2-scale *re-pol-weight*)
(NEWLINE 1)
(FFIELD 4 " ")
(FFIELD 25 "Ammo Expended        " )
(FFIELD 2 " ")
(DFIELD 5 "" *re-ammo-weight*)
(FFIELD 10 " ")
(DFIELD 2 "" re-ammo-1-scale)
(FFIELD 5 " ")
(XDEMON 6 re-ammo-1-wtd-score WEIGHTED-VALUE re-ammo-1-scale *re-ammo-weight*)
(FFIELD 7 " ")
(DFIELD 2 "" re-ammo-2-scale)
(FFIELD 5 " ")
(XDEMON 6 re-ammo-2-wtd-score WEIGHTED-VALUE re-ammo-2-scale *re-ammo-weight*)
(NEWLINE 1)
(FFIELD 4 " ")
(FFIELD 25 "FEBA Mvmt (km)       " )
(FFIELD 2 " ")
(DFIELD 5 "" *feba-mvmt-weight*)
```

```
(FFIELD 10 " ")
(DFIELD 2 "" feba-mvmt-1-scale)
(FFIELD 5 " ")
(XDEMON 6 feba-mvmt-1-wtd-score WEIGHTED-VALUE feba-mvmt-1-scale *feba-mvmt-weight*)
(FFIELD 7 " ")
(DFIELD 2 "" feba-mvmt-2-scale)
(FFIELD 5 " ")
(XDEMON 6 feba-mvmt-2-wtd-score WEIGHTED-VALUE feba-mvmt-2-scale *feba-mvmt-weight*)
(NEWLINE 1)
(FFIELD 4 " ")
(FFIELD 25 "Time Required (hrs)  " )
(FFIELD 2 " ")
(DFIELD 5 "" *time-rqd-weight*)
(FFIELD 10 " ")
(DFIELD 2 "" time-rqd-1-scale)
(FFIELD 5 " ")
(XDEMON 6 time-rqd-1-wtd-score WEIGHTED-VALUE time-rqd-1-scale *time-rqd-weight*)
(FFIELD 7 " ")
(DFIELD 2 "" time-rqd-2-scale)
(FFIELD 5 " ")
(XDEMON 6 time-rqd-2-wtd-score WEIGHTED-VALUE time-rqd-2-scale *time-rqd-weight*)
(NEWLINE 1)
(FFIELD 15 " ")
(FFIELD 8 "SUBTOTAL")
(FFIELD 28 " ")
(XDEMON 8 wg-subtot-1 TOTAL fc-pers-1-wtd-score fc-equip-1-wtd-score ec-pers-1-wtd-score
                          ec-equip-1-wtd-score re-pol-1-wtd-score re-ammo-1-wtd-score
                          feba-mvmt-1-wtd-score time-rqd-1-wtd-score)
(FFIELD 12 " ")
(XDEMON 8 wg-subtot-2 TOTAL fc-pers-2-wtd-score fc-equip-2-wtd-score ec-pers-2-wtd-score
                          ec-equip-2-wtd-score re-pol-2-wtd-score re-ammo-2-wtd-score
                          feba-mvmt-2-wtd-score time-rqd-2-wtd-score)
(NEWLINE 2)
(FFIELD 2 " ")
(FFIELD 10 "SUBJECTIVE")
(NEWLINE 1)
(FFIELD 4 " ")
(FFIELD 25 "Accomplish Mission       " )
(FFIELD 2 " ")
(DFIELD 5 "" *subj-a-weight*)
(FFIELD 10 " ")
(DFIELD 2 "" subj-a-1-scale)
(FFIELD 5 " ")
(XDEMON 6 subj-a-1-wtd-score WEIGHTED-VALUE subj-a-1-scale *subj-a-weight*)
(FFIELD 7 " ")
(DFIELD 2 "" subj-a-2-scale)
(FFIELD 5 " ")
(XDEMON 6 subj-a-2-wtd-score WEIGHTED-VALUE subj-a-2-scale *subj-a-weight*)
(NEWLINE 1)
(FFIELD 4 " ")
(FFIELD 25 "Effective Use of Assets  " )
(FFIELD 2 " ")
(DFIELD 5 "" *subj-b-weight*)
```

```
(FFIELD 10 " ")
(DFIELD 2 "" subj-b-1-scale)
(FFIELD 5 " ")
(XDEMON 6 subj-b-1-wtd-score WEIGHTED-VALUE subj-b-1-scale *subj-b-weight*)
(FFIELD 7 " ")
(DFIELD 2 "" subj-b-2-scale)
(FFIELD 5 " ")
(XDEMON 6 subj-b-2-wtd-score WEIGHTED-VALUE subj-b-2-scale *subj-b-weight*)
(NEWLINE 1)
(FFIELD 4 " ")
(FFIELD 25 "Flexibility                " )
(FFIELD 2 " ")
(DFIELD 5 "" *subj-c-weight*)
(FFIELD 10 " ")
(DFIELD 2 "" subj-c-1-scale)
(FFIELD 5 " ")
(XDEMON 6 subj-c-1-wtd-score WEIGHTED-VALUE subj-c-1-scale *subj-c-weight*)
(FFIELD 7 " ")
(DFIELD 2 "" subj-c-2-scale)
(FFIELD 5 " ")
(XDEMON 6 subj-c-2-wtd-score WEIGHTED-VALUE subj-c-2-scale *subj-c-weight*)
(NEWLINE 1)
(FFIELD 4 " ")
(FFIELD 25 "Facilitate Future Ops     " )
(FFIELD 2 " ")
(DFIELD 5 "" *subj-d-weight*)
(FFIELD 10 " ")
(DFIELD 2 "" subj-d-1-scale)
(FFIELD 5 " ")
(XDEMON 6 subj-d-1-wtd-score WEIGHTED-VALUE subj-d-1-scale *subj-d-weight*)
(FFIELD 7 " ")
(DFIELD 2 "" subj-d-2-scale)
(FFIELD 5 " ")
(XDEMON 6 subj-d-2-wtd-score WEIGHTED-VALUE subj-d-2-scale *subj-d-weight*)
(NEWLINE 1)
(FFIELD 4 " ")
(FFIELD 25 "Risk                       " )
(FFIELD 2 " ")
(DFIELD 5 "" *subj-e-weight*)
(FFIELD 10 " ")
(DFIELD 2 "" subj-e-1-scale)
(FFIELD 5 " ")
(XDEMON 6 subj-e-1-wtd-score WEIGHTED-VALUE subj-e-1-scale *subj-e-weight*)
(FFIELD 7 " ")
(DFIELD 2 "" subj-e-2-scale)
(FFIELD 5 " ")
(XDEMON 6 subj-e-2-wtd-score WEIGHTED-VALUE subj-e-2-scale *subj-e-weight*)
(NEWLINE 1)
(FFIELD 4 " ")
(DFIELD 25 " " *subj-f-name*)
(FFIELD 2 " ")
(DFIELD 5 "" *subj-f-weight*)
(FFIELD 10 " ")
```

```
(DFIELD 2 "" subj-f-1-scale)
(FFIELD 5 " ")
(XDEMON 6 subj-f-1-wtd-score WEIGHTED-VALUE subj-f-1-scale *subj-f-weight*)
(FFIELD 7 " ")
(DFIELD 2 "" subj-f-2-scale)
(FFIELD 5 " ")
(XDEMON 6 subj-f-2-wtd-score WEIGHTED-VALUE subj-f-2-scale *subj-f-weight*)
(NEWLINE 1)
(FFIELD 4 " ")
(DFIELD 25 " " *subj-g-name*)
(FFIELD 2 " ")
(DFIELD 5 "" *subj-g-weight*)
(FFIELD 10 " ")
(DFIELD 2 "" subj-g-1-scale)
(FFIELD 5 " ")
(XDEMON 6 subj-g-1-wtd-score WEIGHTED-VALUE subj-g-1-scale *subj-g-weight*)
(FFIELD 7 " ")
(DFIELD 2 "" subj-g-2-scale)
(FFIELD 5 " ")
(XDEMON 6 subj-g-2-wtd-score WEIGHTED-VALUE subj-g-2-scale *subj-g-weight*)
(NEWLINE 1)
(FFIELD 4 " ")
(DFIELD 25 " " *subj-h-name*)
(FFIELD 2 " ")
(DFIELD 5 "" *subj-h-weight*)
(FFIELD 10 " ")
(DFIELD 2 "" subj-h-1-scale)
(FFIELD 5 " ")
(XDEMON 6 subj-h-1-wtd-score WEIGHTED-VALUE subj-h-1-scale *subj-h-weight*)
(FFIELD 7 " ")
(DFIELD 2 "" subj-h-2-scale)
(FFIELD 5 " ")
(XDEMON 6 subj-h-2-wtd-score WEIGHTED-VALUE subj-h-2-scale *subj-h-weight*)
(NEWLINE 1)
(FFIELD 15 " ")
(FFIELD 8 "SUBTOTAL")
(FFIELD 28 " ")
(XDEMON 8 subj-subtot-1 TOTAL subj-a-1-wtd-score subj-b-1-wtd-score
                              subj-c-1-wtd-score subj-d-1-wtd-score subj-e-1-wtd-score
                              subj-f-1-wtd-score subj-g-1-wtd-score subj-h-1-wtd-score)
(FFIELD 12 " ")
(XDEMON 8 subj-subtot-2 TOTAL subj-a-2-wtd-score subj-b-2-wtd-score
                              subj-c-2-wtd-score subj-d-2-wtd-score subj-e-2-wtd-score
                              subj-f-2-wtd-score subj-g-2-wtd-score subj-h-2-wtd-score)
(NEWLINE 2)  )
```

APPENDIX C

FILE COAAT.LISP

APPENDIX C - FILE COAAT.LISP

CONTENTS

>COAAT>COAAT.LISP - 10/24/89

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Package: USER; Base: 10 -*-
;*****************************
;*      COAAT.LISP           *
;*****************************
; Top level COAAT functions, defines the global variables and contains top level functions to
; load files, initialize data, and start COAAT.
; Three modes of COAAT can be run;
;       TRAINING =  all data entered by the user, *mode* set to nil.
;       EXERCISE =  CE data entered in Mod 1 by the user, canned set of CE's
;                   is read in for Mod 2, Two data sets are available
;                   AB = COA-1 in the north and B-A = COA-1 in the south,
;                   user does wargaming and completes Mod 3, *mode* is set
;                   to exercise.
;       DEMO     =  Canned data is read in for Mod 1 with first two CE's
;                   blank, entire set comes up for Mod 2 including some
;                   wargaming data, weights and scales are included for
;                   Mod 3, *mode* set to demo.
; Different modes are run by selecting the desired mode from a pop-up menu after the initial
; screen is activated.  Also, when EXIT is chosen from the main process menu the choice is
; given to change the running mode.  This will reset the mode flag and initialize all variables.
; If EXERCISE mode is chosen the user must then select the desired data set, COA-1 in the
; north = A-B or COA-1 in the south = B-A.  The RETURN option can be used to return to the
; current mode without initializing variables (all entered data will still be available).


;*****************************************************************************
;  define global variables

;------Data variables-----------------------------------------------

(defvar *special-user-variables*)
(defvar *special-global-variables*)
(defvar *demons*)
(defvar *variable-data-alist*)
                ;;; An assoc list containing every variable entered, by name, with its value.
(defvar *mod1-data-alist*)
                ;;; Copy of *variable-data-alist* saved from Mod1 during an exercise (exercise
                ;   mode running)
(defvar *exer-data-alist*)
                ;;; Copy of *variable-data-alist* saved from Mod2 and Mod3 during an exercise.
(defvar *pointer-variable-alist*)
                ;;; An assoc list; return table pointers with the name of the variable at that
                ;   location required when backing thru the table to determine what variable the
                ;   cursor is on.
(defvar *global-pointer-variable-alist*)
                ;;; An assoc list; for all modules; Mod-#, the *pointer-variable-alist*
(defvar *ce-number-list*)
                ;;; A list of CE numbers as they are entered in Module 1
```

```
(defvar *ce-data-alist*)
        ;;; An assoc list containing ce-type, objective, and comment by base CE
        ;   number; i.e. (ce-type-## xyz)
(defvar *mod1-ce-data-alist*)
        ;;; Copy of *ce-data-alist* saved from Mod1 during exercise to allow reentry
        ;   to Mod1 to show user input data.
(defvar *ce-menu-list*)
        ;;; A menu list contructed from *mod1- or *ce-data-alist* as appropriate,
        ;   with CE numbers in proper sequence and with CE type and objective in
        ;   the documentation line.
(defvar *mission*)
        ;;; The primary mission being analyzed thru the use of COAAT, used to
        ;   construct name of MOD 1 screen file for different CE type menu and
        ;   to change FEBA movement scale values.
(defvar *ce-num*)
        ;;; The current CE number
(defvar *type*)
        ;;; The type of the current CE, used only in WG worksheet
(defvar *objective*)
        ;;; The objective of the current CE, used only in WG worksheet
(defvar *coa*)
        ;;; The current COA, as "COA-#", set by TFIELD and by WARGAME-CE
(defvar *moa-type*)
        ;;; The method of analysis type selected by the user
(defvar *wgws-returns*)
        ;;; Assoc list of WG worksheet return-tables for each CE by CE number
(defvar *wgws-demons*)
        ;;; Assoc list of WG worksheet demon tables for each CE by CE number
(defvar *wgws-cursorpos*)
        ;;; Curscrpos table for WG worksheet, all worksheets are the same
(defvar *exercise-set* nil)
        ;;; Define the data set to use when the exercise version is running:
        ;   AB = COA-1 north; BA = COA-1 south.
(defvar *next-ce-choice* nil)
        ;;; the default next choice CE number for the menu of CE numbers
(defvar *next-mod* 1)
        ;;; The next module which can be exercised, determines the active items
        ;   on the main menu


;------File designation variables----------------------

(defvar *out-file*)
        ;;; The print output file opened in MAIN-LOOP for mod1 and exit
(defvar *debug-out*)
        ;;; File for debug output writes
```

>COAAT>COAAT.LISP - 10/24/89

```
;------Flag variables----------------------------------

(defvar *mode* nil)
                ;;; Flag for the desired run mode of COAAT; TRAINING = nil, EXERCISE,
                ;   or DEMO.
(defvar *change-flag* nil)
                ;;; Flag for MSETS and MROW to not when cursor has been backed up
                ;   (probably to change data), i.e., editing data but not in Edit mode.
(defvar *debug* nil)
                ;;; Flag for debug output, when set to on, debug-out.file is opened as
                ;   *debug-out*, functions OUTPUT and OUTPUT1 write
(defvar *sens-anal-added* nil)
                ;;; Flag to include sensitivity analysis choice in th end of Mod 3 menu.
                ;   Set by function ADD-SENSTITIVITY-ANALYSIS.


;------Window variables---------------------------------

(defvar *coaat-windows*)
(defvar *command-window*)
(defvar *process-menu*)
(defvar *edit-window*)
(defvar *title-pane*)
(defvar *pop-up-menu*)
(defvar *instruction-window*)
(defvar *prompt-window*)
(defvar *prompt-pane*)
(defvar *worksheet-window*)


(shadowing-import 'sys:read-character 'user)
```

>COAAT>COAAT.LISP - 10/24/89

```
; ***************************************************************************
(defun COAAT ( )
;;; Start up the COAAT system. Load files, bring up the window, and get the mode to run.
   (load  ">coaat>coaat-flavor")
   (load  ">coaat>coaat-interpreter")
   (load  ">coaat>coaat-driver")
   (load  ">coaat>coaat-demons")
   (load  ">coaat>coaat-text")
   (load  ">coaat>coaat-sens-anal")
   (START-COAAT)
   ; Get the mode (Training, Exercise, or Demo) for this run
   (let ( (answer (EXIT-COAAT) ) )
      (if (equalp answer 'continue)
         (setq *mode* nil)          ;; Set mode to Training if Return selected
         (setq *mode* answer) ) ) )


; ***************************************************************************
(defun START-COAAT ( )
   (tv:add-select-key #\z 'coaat-window-flavor "COAAT" '(EXPOSE-COAAT) )
   (send *terminal-io* ':set-more-p nil)
   (COAAT-INIT-GLOBALS)
   ; Expose and activate the window with the mission spec menu up.
   (EXPOSE-COAAT) )


; ***************************************************************************
(defun LISP-EVAL-READ-PRINT (&rest ignore)
   (catch 'coaat-exit
      (si:lisp-top-level1 *command-window*) )
   (send *coaat-windows* ':deactivate)      )


; ***************************************************************************
(defun EXPOSE-COAAT ( )
;;; Initialize the display
   (setq *title-pane* (send *coaat-windows* ':get-pane 'main-title-pane) )
   (send *coaat-windows* ':set-configuration 'mission-spec-scrn)
   (send *coaat-windows* ':send-all-exposed-panes :clear-window)
   (MAIN-TITLE)
   (setq *prompt-window*
         (tv:make-window 'COAAT-POP-UP-PROMPT :superior *edit-window*) )
   (send *coaat-windows* ':expose)
   (send *coaat-windows* ':activate) )
```

```
; ***************************************************************************
(defun INIT-COAAT ( )
   (loop
      ; Initialize the global variables
      (COAAT-INIT-GLOBALS)
      ; Get the mission type for this run
      (setq *mission* (SELECT-MISSION) )
      (MY-ASSERT '*variable-data-alist* '*mission* *mission*)
      (setq *moa-type* (SELECT-MOA) )
      (MY-ASSERT '*variable-data-alist* '*moa-type* *moa-type*)
      ; Put up the main process menu and start
      (START-IT)
      (if (equalp *mode* 'exit)
         (return) )  ) )


; ***************************************************************************
(defun COAAT-INIT-GLOBALS ( )
   (setq *command-window* (send *coaat-windows* ':get-pane 'command-window) )
   (setq *edit-window* (send *coaat-windows* ':get-pane 'edit-window) )
   (setq *title-pane* (send *coaat-windows* ':get-pane 'main-title-pane) )
   (setq *process-menu* (send *coaat-windows* ':get-pane 'process-menu) )
   (setq *prompt-pane* (send *coaat-windows* ':get-pane 'prompt-pane) )
   (setq *variable-data-alist* nil)   (setq *mod1-data-alist* nil)
   (setq *exer-data-alist* nil)   (setq *mod1-ce-data-alist* nil)
   (setq *next-ce-choice* nil)   (setq *global-pointer-variable-alist* nil)
   (setq *ce-number-list* nil)   (setq *ce-menu-list* nil)
   (setq *ce-data-alist* nil)   (setq *wgws-returns* nil)
   (setq *wgws-demons* nil)   (setq *wgws-cursorpos* nil)
   (setq *demons* nil)    (setq *next-mod* 1)
   (MY-ASSERT '*variable-data-alist* 'mod1-data-exists nil)
   (MY-ASSERT '*variable-data-alist* 'mod2-data-exists nil)
   (MY-ASSERT '*variable-data-alist* 'mod3-data-exists nil)  )


; ***************************************************************************
(defun START-IT ( )
;;; Primary reason for this function is to start from crash without initializing variables.
   ; Reset to the main menu
   (RESET-COAAT)
   ; Start up the main menu control loop.
   (MAIN-LOOP)    )
```

```
; ***************************************************************************
(defun RESET-COAAT ( )
;;; Reset screen configuration to main
   (send *process-menu* ':set-item-list (PROCESS-MENU-LIST) )
   (send *coaat-windows* ':set-configuration 'process-scrn)
   (setq *title-pane* (send *coaat-windows* ':get-pane 'main-title-pane) )
   ; Clear any left over stuff from the previous screen
   (send *coaat-windows* ':send-all-exposed-panes :clear-window)
   ; Refresh the main menu
   (send *coaat-windows* ':refresh)
   ; Set the mouse position on menu choice for *next-mod*
   (SET-PROCESS-CHOICE)
   ; Get the title data for the main menu screen
   (MAIN-TITLE)   )


; ***************************************************************************
(defun SELECT-MISSION ( )
;;; Reset screen configuration to mission specification
   (send *coaat-windows* ':set-configuration 'mission-spec-scrn)
   (setq *title-pane* (send *coaat-windows* ':get-pane 'main-title-pane) )
   ; Clear any left over stuff from the previous screen
   (send *coaat-windows* ':send-all-exposed-panes :clear-window)
   ; Refresh the main menu
   (send *coaat-windows* ':refresh)
   ; Put up the starting explanation text
   (EXPLAIN-START-COAAT)
   ; Set the mouse position on the OFFENSE choice
   (send *coaat-windows* ':set-mouse-position 210 655)
   ; Get the title data for the main menu screen
   (MAIN-TITLE)

   (loop as blip = (send *command-window* ':list-tyi)
         as result-value =
            (cond
               ( (and (listp blip) (eq (car blip) ':menu) )
                  (send (fourth blip) ':execute (second blip) ))
               (t nil) )                    ;; ignore keyboard input
         do
      (zl:selectq result-value
         (:offense (return 'offense) )
         (:defense (return 'defense) )
         (:retrograde (return 'retrograde) )) ) )
```

```
; ***************************************************************************
(defun SELECT-MOA ( )
;;; Use a command menu format to get the moa from the user
   ; Reset screen configuration to moa selection
   (send *coaat-windows* ':set-configuration 'moa-scrn)
   (setq *title-pane* (send *coaat-windows* ':get-pane 'main-title-pane) )
   ; Clear any left over stuff from the previous screen
   (send *coaat-windows* ':send-all-exposed-panes :clear-window)
   ; Refresh the main menu
   (send *coaat-windows* ':refresh)
   ; Put up the moa explanation text
   (EXPLAIN-MOA)
   ; Set the mouse position on the AVENUE choice
   (send *coaat-windows* ':set-mouse-position 240 700)
   ; Get the title data for the main menu screen
   (MAIN-TITLE)

   (loop as blip = (send *command-window* ':list-tyi)
         as result-value =
            (cond
               ( (and (listp blip) (eq (car blip) ':menu) )
                  (send (fourth blip) ':execute (second blip) ))
               (t nil) )                          ;; ignore keyboard input
         do
      (zl:selectq result-value
         (:avenue (return 'Avenue) )
         (:belt (return 'Belt) )
         (:box (return 'Box) )) ) )


; ***************************************************************************
(defun PROCESS-MENU-LIST ( )
   (cond
      ( (equalp *next-mod* 1)
         '( ("      Critical Event Assignment      "
            :value :mod-1
            :documentation
            " Assign Critical Events to COA's for Analysis")
           ("      Critical Event War-Gaming      "
            :value nil
            :documentation
            " NOT ALLOWED - Critical Event Assignment must be performed first")
           ("      Course Of Action Comparison    "
            :value nil
            :documentation
            " NOT ALLOWED - Critical Event Assignment must be performed first")
           (" " :no-select t)                     ;; Blank entry to move Exit to center
           (" Exit "
            :value :exit
            :documentation
            " Exit the COAAT System")  ) )
```

```
      ( (equalp *next-mod* 2)
          '( ("      Critical Event Assignment      "
             :value :mod-1
             :documentation
             " Assign Critical Events to COA's for Analysis")
            ("      Critical Event War-Gaming      "
             :value :mod-2
             :documentation
             " War-Game Selected Critical Events")
            ("      Course Of Action Comparison    "
             :value nil
             :documentation
             " NOT ALLOWED - Critical Events must be War-gamed first")
            (" " :no-select t)          ;blank entry to move Exit to center
            (" Exit "
             :value :exit
             :documentation
             " Exit the COAAT System")  ) )

      (t                  ;any other value, all choices valid
          '( ("      Critical Event Assignment      "
             :value :mod-1
             :documentation
             " Assign Critical Events to COA's for Analysis")
            ("      Critical Event War-Gaming      "
             :value :mod-2
             :documentation
             " War-Game Selected Critical Events")
            ("      Course Of Action Comparison    "
             :value :mod-3
             :documentation
             "Compare COA's Based on Critical Event War-Gaming")
            (" " :no-select t)          ;blank entry to move Exit to center
            (" Exit "
             :value :exit
             :documentation
             " Exit the COAAT System")  ) ) ))


; ***************************************************************************
(defun SET-PROCESS-CHOICE ( )
;;; Set the process menu to the choice of the next mod, after all have been run, set
;   *next-mod* to four, any value other than 1-3, default choice is "exit".
   (cond
      ( (equalp *next-mod* 1)
         (send *process-menu* :set-mouse-position 300 35) )
      ( (equalp *next-mod* 2)
         (send *process-menu* :set-mouse-position 650 35) )
      ( (equalp *next-mod* 3)
         (send *process-menu* :set-mouse-position 775 35) )
      (t (send *process-menu* :set-mouse-position 500 60) ) ) )
```

>COAAT>COAAT.LISP - 10/24/89

```
; ***************************************************************************
;   Sensitivity analysis -- Special arrangements to keep sensitivity
;   analysis as special feature and not a standard feature of COAAT.
;   Allows running exercises without sensitivity analysis so they will
;   be consistent with exercises already run.
;   All files are loaded, sensitivity analysis is added by adding it to the
;   end of Module 3 menu, MOD3-DONE.  The flag *sens-anal-added* controls
;   whether that choice is on the menu or not.  Thus, MOD3-DONE is the
;   only routine that requires modification to add sensitvity analysis as
;   a permanent part of COAAT and these functions can be deleted.


(defun ADD-SENSITIVITY-ANALYSIS ( )
;;; This function can be run from the Lisp Listener or from the COAAT
;   command window prior to running INIT-COAAT.  It sets the flag to add
;   the sensitivity analysis choice to the MOD3-DONE menu.
   (setq *sens-anal-added* t) )

(defun DELETE-SENSITIVITY-ANALYSIS ( )
;;; This function can be run from the Lisp Listener or from the COAAT
;   command window prior to running INIT-COAAT.  It sets the flag to
;   eliminate the sensitivity analysis choice from the MOD3-DONE menu.
   (setq *sens-anal-added* nil) )

;---------------------------------------------------------------------
```

APPENDIX D

FILE COAAT-DRIVER.LISP

APPENDIX D - FILE COAAT-DRIVER.LISP

CONTENTS

>COAAT>COAAT-DRIVER.LISP - 10/24/89

```
;;; -*- Mode: LISP; Base: 10; Syntax: Common-lisp; Package: USER -*-
;*******************************
;*     COAAT-DRIVER.LISP       *
;*******************************
; Second level functions for COAAT, the Main-Loop to control modules, the drivers for
; each module execution.
; Uses the variable *mode* to determine when and what canned data to read in for
; Training, Exercise (variable *exercise-set* determines data set to be read), and
; Demo modes.
; Output for debug purposes is sent to files debug-out#.file (#=mod number) when flag
; *debug* is set on, do this in the listener (setq *debug* 'on) and use functions
; OUTPUT or OUTPUT1 for output, *debug* must be reset (turned off) manually also.

; ********************************************************************************
(defun MAIN-LOOP ( )
;;; The main driver loop for COAAT, calls modules based on choice from the
;   main process menu.

   (let ( (mod1-return-table nil) (mod1-user-variables nil)
          (mod1-global-variables nil) (mod1-cursorpos-table nil)
          (mod1-demons nil)
          (mod2-return-table nil) (mod2-user-variables nil)
          (mod2-global-variables nil) (mod2-cursorpos-table nil)
          (mod2-demons nil) (mod2-summary-scrn nil)
          (mod3-return-table nil) (mod3-user-variables nil)
          (mod3-global-variables nil) (mod3-cursorpos-table nil)
          (mod3-demons nil)  )

      ; For the demo mode, if we don't have data read in the canned set
      (when (equalp *mode* 'demo)
         (if (not *variable-data-alist*)
            (let ( (variable-data-file
                        (open ">coaat>demo-variable-data-list.file" :direction :input) ) )
               (setq *variable-data-alist* (read variable-data-file) )
               (close variable-data-file) ) ) )

   ; Look for a blip, a blip is the list returned by :list-tyi
   ; Choosing an item causes a list of the following form to be sent to the io buffer
   ; (:menu chosen-item button-mask window)

   (loop as blip = (send *command-window* ':list-tyi)
         as result-value =
            (cond
               ( (and (listp blip) (eq (car blip) ':menu) )
                  (send (fourth blip) ':execute (second blip) ) )
               (t nil) )                        ;; ignore keyboard input
         do
      (zl:selectq result-value
```

>COAAT>COAAT-DRIVER.LISP - 10/24/89

```
;---- Module 1 ----------------------------------------------------
   (:mod-1
      ; Exercise mode - get mod1 data saved from prior entry in Mod1,
      ; if nothing there OK.
      (when (equalp *mode* 'exercise)
         (setq *variable-data-alist* *mod1-data-alist*)
         (setq *ce-data-alist* *mod1-ce-data-alist*) )
         (MY-ASSERT '*variable-data-alist* 'start-mod1  (format nil "~\\datime\\") )

      ; For debug output, use functions OUTPUT or OUTPUT1, set *debug* on from listener
      (when (equalp *debug* 'on)
         (setq *debug-out* (open ">coaat>debug-out1.file" :direction :output) ) )

      ; Now run module 1
      (multiple-value-setq
            (mod1-return-table mod1-user-variables mod1-global-variables
                               mod1-cursorpos-table mod1-demons)
            (START-MOD-1 mod1-return-table mod1-user-variables mod1-global-variables
                                           mod1-cursorpos-table mod1-demons) )

      (when (equalp *debug* 'on)   (close *debug-out*) )
      (MY-ASSERT '*variable-data-alist* 'exit-mod1  (format nil "~\\datime\\") )

      ; For EXERCISE Mode save the data for use next time in Mod1, write out the mod1
      ; data (user input CE's), initialize *variable-data-alist* with previously saved exercise
      ; data (mod2 & mod3), if none exists then read in canned data.
      ; Required data file is determined by variable *exercise-set* (AB or BA).
      (when (equalp *mode* 'exercise)
         (setq *mod1-data-alist* *variable-data-alist*)
         (setq *mod1-ce-data-alist* *ce-data-alist*)
         (with-open-file (*out-file* ">coaat>MOD1-OUT.data" :direction :output)
            (prin1 *ce-number-list* *out-file*) (fresh-line *out-file*)
            (prin1 *mod1-data-alist* *out-file*)  )
         ; Get the previously created exercise data, if any.
         (setq *variable-data-alist* *exer-data-alist*)
         (let ( (variable-data-file
                  (open (string-append ">coaat>EXER-variable-data-list-"
                                       *exercise-set* ".file") :direction :input) )
                (ce-menu-file
                  (open (string-append ">coaat>EXER-ce-menu-list-"
                                       *exercise-set* ".file") :direction :input) )
                (ce-data-file
                  (open ">coaat>EXER-ce-data-list.file" :direction :input) )   )

            ; If *variable-data-alist* is empty, read in the canned data
            (when (not *variable-data-alist*)
               (setq *variable-data-alist* (read variable-data-file) ) )
            (close variable-data-file)
            (setq *ce-menu-list* (read ce-menu-file) )
            (close ce-menu-file)
            (setq *next-ce-choice* nil)
            (setq *ce-data-alist* (read ce-data-file) )
            (close ce-data-file)  )  )
```

```
      ; For DEMO Mode read in the canned data set
      (when (equalp *mode* 'demo)
         (let ( (ce-menu-file (open ">coaat>DEMO-ce-menu-list.file" :direction :input) ) )
            (setq *ce-menu-list* (read ce-menu-file) )
            (close ce-menu-file) ) )

      (RESET-COAAT) )


;---- Module 2 ----------------------------------------------------------
   (:mod-2
      (setq *next-ce-choice* nil)
      (MY-ASSERT '*variable-data-alist* 'start-mod2  (format nil "~\\datime\\") )
      (when (equalp *debug* 'on)
         (setq *debug-out* (open ">coaat>debug-out2.file" :direction :output) ) )

      (multiple-value-setq
            (mod2-return-table mod2-user-variables mod2-global-variables
                               mod2-cursorpos-table mod2-demons mod2-summary-scrn)
            (START-MOD-2 mod2-return-table mod2-user-variables mod2-global-variables
                         mod2-cursorpos-table mod2-demons mod1-user-variables
                         mod1-global-variables mod2-summary-scrn) )

      (when (equalp *debug* 'on)  (close *debug-out*) )
      (MY-ASSERT '*variable-data-alist* 'exit-mod2  (format nil "~\\datime\\") )
      (setq *exer-data-alist* *variable-data-alist*)
      (RESET-COAAT)  )


;---- Module 3 ----------------------------------------------------------
   (:mod-3
      (MY-ASSERT '*variable-data-alist* 'start-mod3 (format nil "~\\datime\\") )
      (when (equalp *debug* 'on)
         (setq *debug-out* (open ">coaat>debug-out3.file" :direction :output) ) )

      (multiple-value-setq
            (mod3-return-table mod3-user-variables mod3-global-variables
                               mod3-cursorpos-table mod3-demons)
            (START-MOD-3 mod3-return-table mod3-user-variables
                         mod3-global-variables mod3-cursorpos-table mod3-demons) )

      (when (equalp *debug* 'on)  (close *debug-out*) )
      (setq *next-mod* 4)

      (MY-ASSERT '*variable-data-alist* 'exit-mod3  (format nil "~\\datime\\") )
      (setq *exer-data-alist* *variable-data-alist*)
      (RESET-COAAT)  )
```

```
;---- Exit COAAT ----------
   (:exit
      ; For exercise mode, write out the data, if data is to be saved then archive it.
      (when (equalp *mode* 'exercise)
         (with-open-file (*out-file* ">coaat>EXIT-OUT.DATA" :direction :output)
            (prin1 *ca-menu-list* *out-file*) (fresh-line *out-file*)
            (prin1 *exer-data-alist* *out-file*) )
         (if (SAVE-EXERCISE-DATA?)
            (ARCHIVE-EXERCISE-DATA) ) )
      (let ( (answer (EXIT-COAAT) ) )
         (if (equalp answer 'continue)    ;; Return option, don't reset mode
            (RESET-COAAT)                 ;; just restart at mission selection menu
            (progn                     ;; Other options reset *mode* and initialize
               (setq *mode* answer)
               (return nil) ) ) ) )
      ) )  ) )


; ********************************************************************************
(defun START-MOD-1 (mod1-return-table mod1-user-variables mod1-global-variables
                    mod1-cursorpos-table mod1-demons)
   (setq *special-user-variables* mod1-user-variables)
   (setq *special-global-variables* mod1-global-variables)
   (setq *demons* mod1-demons)
   (if (VALOF '*global-pointer-variable-alist* 'mod-1)
      (setq *pointer-variable-alist* (VALOF '*global-pointer-variable-alist* 'mod-1) )
      (setq *pointer-variable-alist* nil) )

   ; Change the window configuration to module 1
   (setq *title-pane* (send *coaat-windows* ':get-pane 'title-pane) )
   (send *coaat-windows* :set-configuration 'mod-1-scrn)
   (send *coaat-windows* :send-all-exposed-panes :clear-window)
   (MODULE-1-TITLE)
   (let ( (table nil) (ask-mod-done nil)
         (screen-file (string-append ">coaat>coaat-mod-1-" *mission* ".scrn") )  )
      (tv:with-mouse-usurped
         (EXPLAIN-MODULE-1) ;While this is up, load the screen description file
         (with-open-file (in-file screen-file :direction :input)
            (setq table (read in-file) ) )
         (EXIT-EXPLAIN-MOD-1) )  ;put up "press any key" message and exit when pressed
      (send *edit-window* :home-cursor)

      ; If there is no data do input-table, if there is data do redisplay from return table
      (if (not (VALOF '*variable-data-alist* 'mod1-data-exists) )
         (progn
            (multiple-value-setq
                  (mod1-return-table mod1-user-variables mod1-global-variables
                                     mod1-cursorpos-table mod1-demons)
                  (INPUT-TABLE *edit-window* table) )
            (MY-ASSERT '*variable-data-alist* 'mod1-data-exists 't)
            (setq ask-mod-done 't) )
         (REDISPLAY-RETURN-TABLE *edit-window* table mod1-return-table) )
```

```
      (setq *change-flag* nil)              ; Make sure change flag not set for edit phase
      (loop
         (if ask-mod-done
            (if (MOD-DONE)
               (progn
                  (send *edit-window* :clear-window)
                  (if (equalp *next-mod* 1) (setq *next-mod* 2) )
                  (return) ) ) )
         (setq ask-mod-done 't)
         (send *edit-window* :home-cursor)
         (multiple-value-setq
            (mod1-return-table mod1-user-variables mod1-global-variables
                               mod1-cursorpos-table mod1-demons)
            (UPDATE-TABLE *edit-window* table mod1-user-variables
                          mod1-global-variables mod1-demons
                          mod1-return-table mod1-cursorpos-table) ) )  )

   ; Save the pointer-variable-alist to global
   (MY-ASSERT '*global-pointer-variable-alist* 'mod-1 *pointer-variable-alist*)
   ; Save total number of COA's, maximum COA's, equal the current COA number
   (MY-ASSERT '*variable-data-alist* 'max-coa
              (VALOF '*variable-data-alist* 'coa-number) )
   ; Make menu list of CE's for use by MOD-2
   (setq *ce-menu-list* (MAKE-CE-MENU-LIST *ce-number-list*) )

   (values mod1-return-table mod1-user-variables mod1-global-variables
           mod1-cursorpos-table mod1-demons)  )


; ***************************************************************************
(defun START-MOD-2 (mod2-return-table mod2-user-variables mod2-global-variables
                    mod2-cursorpos-table mod2-demons
                    mod1-user-variables mod1-global-variables mod2-summary-scrn)

   ; Change the window configuration to module 2
   (setq *title-pane* (send *coaat-windows* :get-pane 'title-pane) )
   (send *coaat-windows* :set-configuration 'mod-2-scrn)
   (send *coaat-windows* :send-all-exposed-panes :clear-window)
   (MODULE-2-TITLE)
   (let ( (table nil) (ws-table nil) )
      (tv:with-mouse-usurped
         (EXPLAIN-MODULE-2)      ;;explain module, while it's up read input and create table
         (when (not mod2-summary-scrn)  ;;if doesn't exist, build it--else use what already built
            (case *mode*              ;;mode determines file to read, if nil build the screen
               (exercise
                  (with-open-file (summary-scrn-file
                                   (string-append ">coaat>EXER-mod2-summary-"
                                                  *exercise-set* ".scrn")
                                   :direction :input)
                     (setq mod2-summary-scrn (read summary-scrn-file) ) )  )
```

```
          (demo
            (with-open-file (summary-scrn-file ">coaat>DEMO-mod2-summary.scrn"
                                               :direction :input)
              (setq mod2-summary-scrn (read summary-scrn-file) ) )  )
          (otherwise          ;;training mode, build the screen table
            (with-open-file (table-file ">coaat>coaat-mod-2.scrn" :direction :input)
              (setq table (read table-file) ) )
              (setq mod2-summary-scrn
                  (CREATE-MOD2-TABLE mod1-global-variables
                                     mod1-user-variables table) ) ) ) )

      ; Get the worksheet description table
      (with-open-file (ws-table-file ">coaat>coaat-mod-2a.scrn" :direction :input)
        (setq ws-table (read ws-table-file) ) )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; --- To save canned data for DEMO or EXERCISE mode, include following 5 lines --
;    Must be run in training mode to build the table, enter prefix for ????
;      (with-open-file (summary-scrn-file ">coaat>????-mod2-summary.scrn"
;                                         :direction :output
;                                         :if-does-not-exist :create
;                                         :if-exists :new-version)
;        (prin1 mod2-summary-scrn summary-scrn-file) )
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


      (EXIT-EXPLAIN-MOD-2) )  ;put up "press any key" message and exit when pressed
    (setq *special-user-variables* mod2-user-variables)
    (setq *special-global-variables* mod2-global-variables)

    (loop
      (if (VALOF '*global-pointer-variable-alist* 'mod-2)
        (setq *pointer-variable-alist* (VALOF '*global-pointer-variable-alist* 'mod-2) )
        (setq *pointer-variable-alist* nil) )
      (setq *demons* mod2-demons)
      (INIT-SCREEN *edit-window* mod2-summary-scrn)
      (if (MOD2-DONE)
        (progn
          (send *edit-window* :clear-window)
          (if (equalp *next-mod* 2) (setq *next-mod* 3) )
          ; Save the pointer-variable-alist to global
          (MY-ASSERT '*global-pointer-variable-alist* 'mod-2 *pointer-variable-alist*)
          (return) ) )
      (WARGAME-CE ws-table) ) )

  (values mod2-return-table mod2-user-variables mod2-global-variables
        mod2-cursorpos-table mod2-demons mod2-summary-scrn) )
```

```
; ********************************************************************************
(defun START-MOD-3 (mod3-return-table mod3-user-variables
                    mod3-global-variables mod3-cursorpos-table mod3-demons)

   (if (VALOF '*global-pointer-variable-alist* 'mod-3)
      (setq *pointer-variable-alist* (VALOF '*global-pointer-variable-alist* 'mod-3) )
      (setq *pointer-variable-alist* nil) )
   ;  Change the window configuration to module 3
   (send *coaat-windows* :set-configuration 'mod-3-scrn)
   (setq *title-pane* (send *coaat-windows* ':get-pane 'title-pane) )
   (send *coaat-windows* :send-all-exposed-panes :clear-window)
   (MODULE-3-TITLE)
   (let ( (table nil) (prompt-var 'COA-EVAL-FACTOR-WEIGHTS) (end-mod3 nil) )
      (tv:with-mouse-usurped
         (EXPLAIN-MODULE-3)
         (EXIT-EXPLAIN-MOD-3) )      ;; Put up "press any key" message and exit when pressed

      (loop                          ;; Module 3 loop over all functions 3A, 3B, & 3C
         (setq *special-user-variables* mod3-user-variables)
         (setq *special-global-variables* mod3-global-variables)
         (setq *demons* mod3-demons)

;---- Module 3A,  Get the factor weighting values for COA comparison ----------------------
         (with-open-file (in-file ">coaat>coaat-mod-3A.scrn" :direction :input)
            (setq table (read in-file) ) )
         (INIT-SCREEN *edit-window* table)
         (DISPLAY-PROMPT prompt-var 275 500)

         (loop                       ;; Loop 3a, factor weighting
            (send *edit-window* :home-cursor)
            (multiple-value-setq
               (mod3-return-table mod3-user-variables mod3-global-variables
                                  mod3-cursorpos-table mod3-demons)
               (INPUT-TABLE *edit-window* table) )

            (if (MOD-DONE)
               (progn
                  (send *prompt-window* :deactivate)
                  (send *edit-window* :clear-window)
                  (return) ) )
            (send *edit-window* :home-cursor) )

;---- Module 3B,  Get scale values for the subjective measures ----------------------------
         (with-open-file (in-file ">coaat>coaat-mod-3B.scrn" :direction :input)
            (setq table (read in-file) ) )
         (INIT-SCREEN *edit-window* table)

         (loop                       ;; Loop 3b, subjective factor scaling
            (send *edit-window* :home-cursor)
            (multiple-value-setq
               (mod3-return-table mod3-user-variables mod3-global-variables
                                  mod3-cursorpos-table mod3-demons)
               (INPUT-TABLE *edit-window* table) )
```

```
            (if (MOD-DONE)
               (progn
                  (send *edit-window* :clear-input)                    ;; Clear any excess input
                  (send *edit-window* :clear-window)
                  (return) ) ) )

;---- Module 3C,  Display the COA comparison matrix ----------------------------------------
         (with-open-file (in-file ">coaat>coaat-mod-3c.scrn" :direction :input)
            (setq table (read in-file) ) )
         (INIT-SCREEN *edit-window* table)
         (send *edit-window* :home-cursor)

         (loop
            (setq end-mod3 (MOD3-DONE) )
            (if end-mod3
               (if (equalp end-mod3 'sa)
                  (FACTOR-WT-SENS-ANAL)
                  (progn                ;; end-mod3 = T
                     ; Save the pointer-variable-alist to global
                     (MY-ASSERT '*global-pointer-variable-alist* 'mod-3 *pointer-variable-alist*)
                     (send *edit-window* :clear-window)
                     (return-from START-MOD-3
                        (values mod3-return-table mod3-user-variables
                                mod3-global-variables mod3-cursorpos-table mod3-demons) ) ) )
            (return) ) )                ;; end-mod3 = nil, loop all mod 3
      ) ) )


; ***************************************************************************************
(defun WARGAME-CE (ws-table)
   (let* ( (comment nil) (ws-return-table nil) (ws-user-variables)
         (ws-global-variables) (ws-demons)
         (wg-start (string-append "START-WG-" *ce-num*) )
         (wg-exit (string-append "EXIT-WG-" *ce-num*) )  )

      (MY-ASSERT '*variable-data-alist* wg-start (format nil "~\\datime\\") )
      (if (VALOF '*global-pointer-variable-alist* 'wgws)
         (setq *pointer-variable-alist* (VALOF '*global-pointer-variable-alist* 'wgws) )
         (setq *pointer-variable-alist* nil) )
      (send *edit-window* ':set-cursorpos 800 100)
      (setq *ce-num* (CHOOSE-CE-NUMBER *edit-window* *ce-menu-list*) )

      ; Get the COA number from the CE number, for display of the WG worksheet
      (setq *coa* (read-from-string (subseq *ce-num* 3 4) ) )

      (multiple-value-setq
         (*ce-num* *type* *objective* comment) (GET-CE-DATA *ce-num*) )
         ; Type and objective made global to make them available as TFIELD arguments
         ; and avoid making them arguments for calls through INPUT-TABLE.
```

```
(setq *ce-num* (read-from-string *ce-num*) )
(send *worksheet-window* :expose)
(if (not (VALOF '*wgws-returns* *ce-num*) )
   (progn                          ;; No previous data for this CE
      (INIT-SCREEN *worksheet-window* ws-table)
      (send *worksheet-window* :home-cursor)
      (multiple-value-setq
         (ws-return-table ws-user-variables ws-global-variables
                          *wgws-cursorpos* ws-demons)
         (INPUT-TABLE *worksheet-window* ws-table) ) )
   (progn                          ;; Previous data exists for this CE
      (setq ws-return-table (VALOF '*wgws-returns* *ce-num*) )
      (setq ws-demons (VALOF '*wgws-demons* *ce-num*) )
      (REDISPLAY-RETURN-TABLE *worksheet-window* ws-table ws-return-table)
      (send *worksheet-window* :home-cursor)
      (multiple-value-setq
         (ws-return-table ws-user-variables ws-global-variables
                          *wgws-cursorpos* ws-demons)
         (UPDATE-TABLE *worksheet-window* ws-table ws-user-variables
                          ws-global-variables ws-demons ws-return-table
                          *wgws-cursorpos*) ) ) )
(loop
   (if (MOD-DONE)
      (progn
         (send *worksheet-window* :clear-window)
         (send *worksheet-window* :clear-input)   ;; clear extraneous input
         (send *worksheet-window* :deactivate)
         (MY-ASSERT '*wgws-returns* *ce-num* ws-return-table)
         (MY-ASSERT '*wgws-demons* *ce-num* ws-demons)
         ; Save the pointer-variable-alist to global
         (MY-ASSERT '*global-pointer-variable-alist* 'wgws *pointer-variable-alist*)
         (MY-ASSERT '*variable-data-alist* wg-exit (format nil "~\\datime\\") )
         (return) ) )

(send *worksheet window* :home-cursor)
(multiple-value-setq
   (ws-return-table ws-user-variables ws-global-variables
                    *wgws-cursorpos* ws-demons)
   (UPDATE-TABLE *worksheet-window* ws-table ws-user-variables ws-global-variables
                    ws-demons ws-return-table *wgws-cursorpos*) ) ) ) )
```

```
; ***************************************************************************
(defun POP-UP-MENU-CHOICE (menu-list label menu-x menu-y mouse-x mouse-y
                                    &optional center default)
;;; Display a pop-up menu and get the user's selection, center causes menu to center
;   around x-y, more flexible than set-position, default is the default menu choice
   (let ( (response nil) )
      (send *pop-up-menu* ':set-item-list menu-list)
      (send *pop-up-menu* ':set-label label)
      (if center
         (send *pop-up-menu* ':center-around menu-x menu-y)
         (send *pop-up-menu* ':set-position menu-x menu-y) )
      (if default
         (multiple-value-bind (x y) (send *pop-up-menu* ':item-cursorpos default)
            (send *pop-up-menu* ':set-mouse-position x (+ y 20) ) )
         (send *pop-up-menu* ':set-mouse-position mouse-x mouse-y) )
      (loop
         (setq response (send *pop-up-menu* ':choose) )
         (if response
            (progn
               (if (equalp response "f")
                  (setq response nil) )
               (return) ) ) )
      (send *pop-up-menu* ':deactivate)
      response) )


; ***************************************************************************
(defun MOD-DONE ( )
   (let ( (response nil)
          (menu-list '( ("Edit" :value "f") ("Done" :value t) ) )
          (label  '(:string " Choose one: "
                    :style (:swiss :bold :normal) ) )
          (menu-x 950) (menu-y 50) (mouse-x 10) (mouse-y 45)    )
      (setq response
         (POP-UP-MENU-CHOICE menu-list label menu-x menu-y mouse-x mouse-y) ) ) )


; ***************************************************************************
(defun MOD2-DONE ( )
   (let ( (response nil)
          (menu-list '( (" War-Game a Critical Event  " :value "f")
                        (" Exit the War-Gaming Module " :value t) ) )
          (label '(:string " Choose one: "
                   :style (:swiss :bold :normal) ) )
          (menu-x 880) (menu-y 50) (mouse-x 10) (mouse-y 25)    )
      (setq response
         (POP-UP-MENU-CHOICE menu-list label menu-x menu-y mouse-x mouse-y) ) ) )
```

```
; ***************************************************************************
(defun MOD3-DONE ( )
   (let ( (response nil)
          ;; Until sensitivity analysis is added as a permanent part !!!!!
          (menu-list (if *sens-anal-added*
                         '(  (" Return to Factor Weights      " :value "f")
                             (" Sensitivity Analysis          " :value sa)
                             (" Exit COA Comparison           " :value t) )
                         ;; else don't offer sensitivity analysis
                         '(  (" Return to Factor Weights      " :value "f")
                             (" Exit COA Comparison           " :value t) ) ) )
          (label '( :string "   Choose one: "
                    :style (:swiss :bold :normal) ) )
          (menu-x 875) (menu-y 50) (mouse-x 15)
          (mouse-y (if *sens-anal-added* 60 45) )    )
      (setq response
          (POP-UP-MENU-CHOICE menu-list label menu-x menu-y mouse-x mouse-y) ) ) )


; ***************************************************************************
(defun MAKE-CE-MENU-LIST (ce-number-list)
;;; Make a menu from the critical events entered in mod 1 so the user can
;   pick the critical events by course of action and method of analysis in mod 2
   (let* ( (choice-list nil)  (doc-string-list nil)
           (num-elem (length ce-number-list) )
           (doc-list (make-list num-elem :initial-element ':documentation ) )   )

      ; Build the choice list, with :value's, get the ce number and put it together with
      ; the keyword :value and a value, have to use a flavor keyword in order to use a
      ; modifier keyword like :documentation
      (dolist (ce# ce-number-list)
         (setq choice-list
               (append choice-list (list (list ce# ':value ce#) ) ) )
         (setq doc-string-list
               (append doc-string-list
                       (list (string-append
                                 " TYPE: "
                                 (string-trim " " (VALOF '*variable-data-alist*
                                                         (read-from-string
                                                            (string-append "ce-type-" ce#) ) ) )
                                 "   OBJECTIVE: "
                                 (string-trim " " (VALOF '*variable-data-alist*
                                                         (read-from-string
                                                            (string-append "objective-" ce#) ) ) )
                                 "   COMMENT: "
                                 (string-trim " " (VALOF '*variable-data-alist*
                                                         (read-from-string
                                                            (string-append "comment-" ce#) ) ) )
      ) ) ) ) )
   ; Add the documentation keyword
   (setq doc-list (mapcar #'list doc-list doc-string-list) )
   ; Put it all together in the menu list and return that list
   (mapcar  #'append  choice-list  doc-list) ) )
```

>COAAT>COAAT-DRIVER.LISP - 10/24/89

```
; ***************************************************************************
(defun EXIT-COAAT ( )
;;; This function is used to exit the COAAT process or to reset the running mode of
;   COAAT when another mode is being run.  The RETURN option simply returns to the
;   current mode without initializing the data.  Mode choices reset the mode  flag and
;   initialize the global variables for a fresh start.

   (let ( (answer nil)
          (menu-list  '(  (" TRAINING       " :value "f")
                          (" EXERCISE       " :value exercise)
                          (" DEMONSTRATION " :vaiue demo)
                          (" RETURN         " :value continue)
                          (" EXIT COAAT     " :value exit) ) )
          (label  '(:string "Select Choice"
                    :style (:swiss :bold :normal) ) )
          (menu-x 500) (menu-y 400) (mouse-x 10) (mouse-y 80)    )
      (setq answer
          (POP-UP-MENU-CHOICE menu-list label menu-x menu-y mouse-x mouse-y) )

      (if (equalp answer 'exercise)
          (setf *exercise-set* (ASK-EXERCISE-SET) ) )

      answer) )                    ;; return answer


; ***************************************************************************
(defun ASK-EXERCISE-SET ( )
;;; Use pop-up-menu to get the data set mode required for an exercise, AB is COA 1
;   in the north (Avenue A), BA is COA 1 in the south (Avenue B).
   (let ( (answer nil)
          (menu-list  '(  (" Data Set A-B " :value :AB)
                          (" Data Set B-A " :value :BA) ) )
          (label '( :string "Select COA Order"
                    :style (:swiss :bold :normal) ) )
          (menu-x 500) (menu-y 400) (mouse-x 10) (mouse-y 30)    )
      (setq answer
          (POP-UP-MENU-CHOICE menu-list label menu-x menu-y mouse-x mouse-y) ) ) )


; ***************************************************************************
(defun SAVE-EXERCISE-DATA? ( )
;;; When EXIT is chosen from the main proces[illegible]nu and the *mode* is exercise ask if the ;
;   data should be saved.
   (let ( (answer nil)
          (menu-list '( (" YES " :value t) (" NO  " :value "f") ) )
          (label '( :string " Save Exercise Data ? "
                    :style (:swiss :bold :normal) ) )
          (menu-x 500) (menu-y 400) (mouse-x 10) (mouse-y 30)    )
      (setq answer
          (POP-UP-MENU-CHOICE menu-list label menu-x menu-y mouse-x mouse-y) ) ) )
```

```
;*****************************************************************************************
(defun ARCHIVE-EXERCISE-DATA ( )
;;; Copy files output by COAAT for exercise from M:>coaat> to M:>coaat-exercise-data>
;   and rename them to include the exercise id number.

   (let ( (id# nil) (dir nil)  )
      ; Create pop-up input window to get exercise ID number
      (setq *input-window*  (tv:make-window
                              'COAAT-POP-UP-INPUT-WINDOW
                              ':left 500
                              ':top 400
                              ':width 220
                              ':height 50
                              ':label '( :string " ENTER EXERCISE ID: "
                                         :style (:swiss :bold-italic :normal) )
                              ':default-character-style '(:swiss :bold :large)
                              ':borders '(3 2 3 2) ) )
      (send *command-window* :set-more-p nil)     ;; let command echos go
      (send *input-window* :expose)
      (send *input-window* :select)

      ; Get exercise ID number, normally four characters, allow five max, any characters
      (loop
         (multiple-value-setq
            (id# dir)
            (INSERT-COLUMNS *input-window* 'IS-ANY 5 id# 500 400) )
         (if id#                              ;; if got a value go on, otherwise loop
            (return nil) ) )

      (send *input-window* :clear-window)
      (send *input-window* :deactivate)

      (let ( (file1
                (string-append "m:>coaat-exercise-data>mod1-EXERCISE-" ID# ".data") )
             (file2
                (string-append "m:>coaat-exercise-data>exit-EXERCISE-" ID# ".data") ) )
         (cp:execute-command "copy file"  "m:>coaat>mod1-out.data"  file1 )
         (cp:execute-command "copy file"  "m:>coaat>exit-out.data"  file2 ) )

      (MAKE-DBASE-DATA id#)
      (send *command-window* :set-more-p t)
   ) )
```

```
;*****************************************************************************
(defun MAKE-DBASE-DATA (id#)
;;; Get two files output from COAAT exercise and create data for input to dBase
;   Exercise ID number is required, four output files are created; mod1, mod2, scale, weight.

   (let ( (coa nil) (prev-coa nil) (av nil) (prev-av nil) (av-num 0)
          (var-name nil) (val nil) (avenue nil) (max-coa nil) (*variable-data-alist* nil)  )
      ; Process the Mod 1 data
      (let* ((file1 (string-append "m:>coaat-exercise-data>mod1-exercise-" id# ".data") )
             (mod1-data-file (open file1 :direction :input) )
             (mod1-output-file (string-append
                                 "m:>coaat-exercise-data>mod1-DBASE-" id# ".data") )
             (ce-number-list (read mod1-data-file) )
             (*variable-data-alist* (read mod1-data-file) )
             (name-list '("ce-type-" "objective-" "comment-") )    )
         (close mod1-data-file)
         (with-open-file (out-file mod1-output-file :direction :output)
            (dolist (ce-num ce-number-list)
               (princ ce-num out-file) (princ ", " out-file)
               (setq coa (read-from-string (subseq ce-num 3 4) ) )
               (princ coa out-file) (princ ", " out-file)
               (if (not (equalp coa prev-coa) )
                  (progn
                     (setq av-num 0)
                     (setq prev-av nil)
                     (setq prev-coa coa) ) )
               (setq av (read-from-string (subseq ce-num 4 5) ) )
               (if (not (equalp av prev-av) )
                  (progn
                     (setq av-num (+ 1 av-num) )
                     (setq prev-av av) ) )
               (setq var-name (read-from-string
                                 (string-append "COA-"
                                                (format nil "~s" coa)
                                                "-AVENUE-"
                                                (format nil "~s" av-num) ) ) )
               (setq avenue (GET-DATA '*variable-data-alist* var-name) )
               (princ avenue out-file)
               (dolist (name name-list)
                  (princ ", " out-file)       ;; Put before so there won't be one after last field
                  (setq var-name (read-from-string (string-append name ce-num) ) )
                  (setq val (GET-DATA '*variable-data-alist* var-name) )
                  (princ val out-file)  )
               (fresh-line out-file) )  )  )
```

```
; Now do the EXIT data
(let* ((file2 (string-append "m:>coaat-exercise-data>exit-exercise-" id# ".data") )
       (exit-data-file (open file2 :direction :input) )
       (mod2-output-file (string-append
                           "m:>coaat-exercise-data>mod2-DBASE-" id# ".data") )
       (scale-output-file (string-append
                           "m:>coaat-exercise-data>scale-DBASE-" id# ".data") )
       (weight-output-file (string-append
                           "m:>coaat-exercise-data>weight-DBASE-" id# ".data") )
       (ce-menu-list (read exit-data-file) )
       (*variable-data-alist* (read exit-data-file) )
       (name-list '("ce-type-" "objective-" "comment-"
                    "fc-pers-" "fc-equip-" "ec-pers-" "ec-equip-"
                    "re-pol-" "re-ammo-" "feba-mvmt-" "time-rqd-") )
       (ce-num nil)    )
   (close exit-data-file)
   (setq coa nil prev-coa nil av nil prev-av nil av-num 0 )
   (with-open-file (out-file mod2-output-file :direction :output)
      (dolist (ce-dat ce-menu-list)
         (setq ce-num (first ce-dat) )
         (princ ce-num out-file) (princ ", " out-file)
         (setq coa (read-from-string (subseq ce-num 3 4) ) )
         (princ coa out-file) (princ ", " out-file)
         (if (not (equalp coa prev-coa) )
            (progn
               (setq av-num 0)
               (setq prev-av nil)
               (setq prev-coa coa) ) )
         (setq av (read-from-string (subseq ce-num 4 5) ) )
         (if (not (equalp av prev-av) )
            (progn
               (setq av-num (+ 1 av-num) )
               (setq prev-av av) ) )
         (setq var-name (read-from-string
                          (string-append "COA-"
                                         (format nil "~s" coa)
                                         "-AVENUE-"
                                         (format nil "~s" av-num) ) ) )
         (setq avenue (GET-DATA '*variable-data-alist* var-name) )
         (princ avenue out-file)
         (dolist (name name-list)
            (princ ", " out-file);put before so there won't be one after last field
            (setq var-name (read-from-string (string-append name ce-num) ) )
            (setq val (GET-DATA '*variable-data-alist* var-name) )
            (princ val out-file) )
         (fresh-line out-file) )  )                ;; Close mod2-output-file
```

```
    ; Now get the COA scale data
    (setq coa 0)
    (setq max-coa (GET-DATA '*variable-data-alist* 'max-coa) )
    (setq name-list '( "fc-pers-" "fc-equip-" "ec-pers-" "ec-equip-"
                       "re-pol-" "re-ammo-" "feba-mvmt-" "time-rqd-"
                       "subj-a-" "subj-b-" "subj-c-" "subj-d-" "subj-e-"
                       "subj-f-" "subj-g-" "subj-h-") )
    (with-open-file (out-file scale-output-file :direction :output)
       (dotimes (i max-coa)
          (setq coa (+ coa 1) )
          (princ coa out-file)
          (dolist (name name-list)
             (princ ", " out-file)       ;; Put before so there won't be one after last field
             (setq var-name
                (read-from-string
                   (string-append name (format nil "~s" coa) "-scale") ) )
             (setq val (GET-DATA '*variable-data-alist* var-name) )
             (princ val out-file)   )
          (fresh-line out-file)    )  )          ;; Close scale-output-file

    ; Now get the factor weights
    ; First add the names of the five hardwired subjective measures
    (setq name-list
       '(  (*subj-a-name* "Accomplish Mission") (*subj-b-name* "Effective Use of Assets")
          (*subj-c-name* "Flexibility") (*subj-d-name* "Facilitate Future Ops")
          (*subj-e-name* "Risk") )  )
    (dolist (name name-list)
       (setq var-name (first name) )
       (setq val (second name) )
       (MY-ASSERT '*variable-data-alist* var-name val) )
    (setq name-list
        '( *fc-pers-weight* *fc-equip-weight* *ec-pers-weight*
          *ec-equip-weight* *re-pol-weight* *re-ammo-weight*
          *feba-mvmt-weight* *time-rqd-weight* *subj-a-weight*
          *subj-b-weight* *subj-c-weight* *subj-d-weight*
          *subj-e-weight* *subj-f-weight* *subj-g-weight* *subj-h-weight*
          *subj-a-name* *subj-b-name* *subj-c-name* *subj-d-name*
          *subj-e-name* *subj-f-name* *subj-g-name* *subj-h-name*) )
    (with-open-file (out-file weight-output-file :direction :output)
       (dolist (name name-list)
          (setq val (GET-DATA '*variable-data-alist* name) )
          (princ val out-file)
          (if (not (equalp name '*subj-h-name*) )
             (princ ", " out-file) ) ) )          ;; Don't put comma after last entry
) ) )
```

```
; ***************************************************************************
(defun OUTPUT (header arg1)
;;; Function to write debug output, header line followed by data
   (when (equalp *debug* 'on)        ;; Preclude error if debug not on
      (terpri *debug-out*)
      (princ header *debug-out*)
      (fresh-line *debug-out*)
      (prin1 arg1 *debug-out*)
      (fresh-line *debug-out*) ) )

; ***************************************************************************
(defun OUTPUT1 (header arg1)
;;; Function to write debug output on 1 line, header and data
   (when (equalp *debug* 'on)        ;; Preclude error if debug not on
      (terpri *debug-out*)
      (princ header *debug-out*)
      (prin1 arg1 *debug-out*)
      (fresh-line *debug-out*) ) )


; ***************************************************************************
(defun SAVE-DEMO-DATA ( )
;;; Routine to save the global lists of data needed as canned data for the demo mode

   (with-open-file
        (variable-data-file  ">coaat>DEMO-variable-data-list.file"
                             :direction :output
                             :if-does-not-exist :create
                             :if-exists :new-version)
        (prin1 *variable-data-alist* variable-data-file) )
   (with-open-file
       (ce-menu-file ">coaat>DEMO-ce-menu-list.file"
                     :direction :output
                     :if-does-not-exist :create
                     :if-exists :new-version)
       (prin1 *ce-menu-list* ce-menu-file) ) )

; ***************************************************************************
(defun SAVE-EXER-DATA ( )
;;; Routine to save the global lists of data needed as canned data for the EXERCISE mode
;   Files have to be renamed to include -AB or -BA identifiers of data set
   (with-open-file
        (variable-data-file  ">coaat>EXER-variable-data-list.file"
                             :direction :output
                             :if-does-not-exist :create
                             :if-exists :new-version)
        (prin1 *variable-data-alist* variable-data-file) )
   (with-open-file
       (ce-menu-file ">coaat>EXER-ce-menu-list.file"
                     :direction :output
                     :if-does-not-exist :create
                     :if-exists :new-version)
       (prin1 *ce-menu-list* ce-menu-file) ) )
```

APPENDIX E

FILE COAAT-INTERPRETER.LISP

CONTENTS

>COAAT>COAAT-INTERPRETER.LISP - 10/24/89

```
;;; -*- Mode: LISP; Base: 10; Syntax: Common-lisp; Package: USER -*-
;*************************************
;     COAAT-INTERPRETER.LISP     *
;*************************************
;;; Functions to interpret the screen description command table, display the screen and
;   accept input data from the user.

;*****************************************************************************************
(defun INPUT-TABLE (win table)
;;; Input-table is the main control for accepting the description of a screen, gets the user
;   input, and returns the screen's data.  Uses EDIT-TABLE to loop thru the table until the
;   end of the table is reached.

   (setq *special-user-variables* nil)
   (setq *special-global-variables* nil)
   (setq *demons* nil)
   (MY-ASSERT '*special-global-variables* (list 'terpri-count win) 0)
   (let ( (return-value nil)
          (direction nil)
          (current-pointer '(0) )
          (cursorpos-table '(-1 nil) ) )
     (multiple-value-setq
          (return-value cursorpos-table)
          (EDIT-TABLE win table return-value direction current-pointer cursorpos-table) )
     (values  return-value
              *special-user-variables*
              *special-global-variables*
              cursorpos-table
              *demons*) ) )


;*****************************************************************************************
(defun UPDATE-TABLE (win table initial-user-variables initial-global-variables initial-demons
                           initial-return-value initial-cursorpos-table)
;;; Control of the update editing of the table.  Uses EDIT-TABLE to loop thru the entire table.

   (setq *special-user-variables* initial-user-variables)
   (setq *special-global-variables* initial-global-variables)
   (setq *demons* initial-demons)
   (MY-ASSERT '*special-global-variables* (list 'terpri-count win) 0)
   (let ( (return-value (DELETE-TRAILING-NEWLINES table initial-return-value) )
          (direction nil)
          (current-pointer '(0) )
          (cursorpos-table (cons -1 (rest initial-cursorpos-table) ) ) )
     (multiple-value-setq
          (return-value cursorpos-table)
          (EDIT-TABLE win table return-value direction current-pointer cursorpos-table) )
     (values  return-value
              *special-user-variables*
              *special-global-variables*
              cursorpos-table
              *demons*) ) )
```

```
;*****************************************************************************
(defun EDIT-TABLE (win table return-value direction current-pointer cursorpos-table)
;;; Edit loop.  Loop until pointer is incremented beyond end of input table.  Uses
;   COMMAND-EXECUTE to apply the command  function retrieved from the input table.

   (loop
      (multiple-value-setq
         (return-value cursorpos-table direction)      ;; Values returned by command-execute
         ;; Execute the command function from the input table.  See if anything exists at
         ;; this pointer location, pass t or nil to command-execute as argument redo. Pass
         ;; nil as the context argument.
         (COMMAND-EXECUTE win current-pointer current-pointer
                          direction table cursorpos-table return-value
                          (if (ITEM-EXISTS current-pointer return-value)
                             t                                    ;; then redo
                             nil)                                  ;; else redo nil
                          nil) )                                  ;; Set context nil

      (if (equalp direction 'back)                                 ;; If backspace
         (setf current-pointer (DECREMENT-POINTER current-pointer 0) )
         (setf current-pointer (INCREMENT-POINTER current-pointer 0) ) )

      (if (and  (equalp direction 'back)                           ;; If backspace and
                (= (first current-pointer) 0) )                     ;; at the top
         (setf direction nil) )

      (if (>= (first current-pointer) (length table) )        ;; If pointer at or past end of table
         (return (values return-value                              ;; then return, else loop
                         cursorpos-table) ) ) ) )


;*****************************************************************************
(defun REDISPLAY-RETURN-TABLE (win table return-table)
;;; Display data from the return table built by previous input. This function requires
;   the return-table, built from previous input, for data.
   ;; Initialize variables for the display of the return table.

   (send win :clear-window)
   (MY-ASSERT '*special-global-variables* (list 'terpri-count win) 0)
   (REDISPLAY-TABLE-AUX win table return-table) )
```

```
;*****************************************************************************
(defun REDISPLAY-TABLE-AUX (win table return-table)
;;; Redisplay functions use return-table for previous input data, INIT-functions are
;   used with default values where there is no input.
   (let ( (pointer 0) )
      (dolist (command-list table)
         (case (first command-list)
            (MSETS
               (dolist (ret-tab (nth pointer return-table) )
                  (REDISPLAY-TABLE-AUX  win (rest (rest (rest command-list) ) )
                                            (rest (rest (rest ret-tab) ) ) ) ) )
            (MROW
               (dolist (ret-tab (nth pointer return-table) )
                  (REDISPLAY-TABLE-AUX  win (rest (rest (rest command-list) ) )
                                            (rest (rest (rest ret-tab) ) ) ) ) )
            ;; Simply display data from return table and move to next field.
            ((CEFIELD DFIELD FFIELD IFIELD MFIELD MAA-PFIELD MOA-PFIELD PFIELD
                      PTFIELD SIFIELD DEMON WSDEMON XDEMON)
               (DISPLAY-STRING win (second command-list) (nth pointer return-table) ) )

            (TFIELD          ; data may require multiple line output
               (REDISPLAY-TFIELD win (second command-list) (nth pointer return-table) ) )

            (CENTER          ; center string on new line and advance to next line
               (INIT-CENTER win (second command-list) ) )

            (NEWLINE         ; advance require number of lines
               (INIT-NEWLINE win (second command-list) ) )

            (LEFT            ; display string on new line and advance to next line
               (INIT-LEFT win (second command-list) ) )

            (otherwise
               (fresh-line)
               (princ "REDISPLAY-TABLE, Unknown Function - ")
               (princ command-list) ) )
         (setq pointer (+ 1 pointer) ) ) ) )


;*****************************************************************************
(defun INIT-SCREEN (win table)
;;; Initialize a screen using default values or previous input data stored in
;   *variable-data-alist*. No return-table required.  CE numbers are taken from
;   *ce-number-list*, by INIT-CEFIELD. A copy of the list is saved and then restored
;   when done.
   (let ( (ce-numbers (copy-list *ce-number-list*) ) )
      (send win :clear-window)
      (MY-ASSERT '*special-global-variables* (list 'terpri-count win) 0)
      (INIT-SCREEN-AUX win table)
      (setq *ce-number-list* (copy-list ce-numbers) ) ) )
```

```
;*****************************************************************************
(defun INIT-SCREEN-AUX (win table)
;;; Detailed control for initializing a screen.  Functions search *variable-data-alist*
;   for previous input data, if no data then default value is displayed.  Intended for
;   displaying a screen using previously input data where no return-table exists.  Also
;   can initialize a display using only default values.

   (dolist (command-list table)
      (case (first command-list)
         (MSETS
            (let ( (table (rest (rest (rest command-list) ) ) ) )
               (INIT-SCREEN-AUX win table) ) )
         (MROW
            (let ( (table (rest (rest (rest command-list) ) ) ) )
               (INIT-SCREEN-AUX win table) ) )

         (TFIELD
            (INIT-TFIELD  win (second command-list) (third command-list)
                          (rest (rest (rest command-list) ) ) ) )
         ((IFIELD PFIELD)
            (INIT-IFIELD win (second command-list)
                         (fourth command-list) (fifth command-list) ) )
         (MFIELD
            (let ((default " "))
               (INIT-IFIELD win (second command-list)
                            default (fourth command-list) ) ) )
         (SIFIELD
            (INIT-SIFIELD  win (second command-list)
                           (third command-list) (fourth command-list) ) )
         (CEFIELD
            (INIT-CEFIELD win (second command-list) ) )

         ((DFIELD PTFIELD)
            (INIT-DFIELD win (second command-list)
                         (third command-list) (fourth command-list) ) )
         (MAA-PFIELD
            (INIT-MAA-PFIELD win (second command-list) ) )

         (MOA-PFIELD
            (INIT-MOA-PFIELD win (second command-list) ) )

         ((DEMON XDEMON)
            (let ( (args (rest (rest (rest (rest command-list) ) ) ) )
                   (function 'INIT-DEMON) )
               (apply function (append (list win  (second command-list)
                                                  (third command-list)
                                                  (fourth command-list) ) args) ) ) )
```

```
        (WSDEMON
           (let ( (args (rest (rest (rest (rest command-list) ) ) ) )
                  (function 'INIT-WSDEMON) )
              (apply function (append (list win  (second command-list)
                                                 (third command-list)
                                                 (fourth command-list) ) args) ) ) )
        (CENTER
           (INIT-CENTER win (second command-list) ) )

        (NEWLINE
           (INIT-NEWLINE win (second command-list) ) )

        (LEFT
           (INIT-LEFT win (second command-list) (third command-list) ) )

        (FFIELD
           (DISPLAY-STRING win (second command-list) (third command-list) ) )

        (otherwise
           (fresh-line)
           (princ "INIT-SCREEN, Unknown Function - ")
           (princ command-list) ) ) ) )


;*****************************************************************************
(defun DISPLAY-STRING (win len str)
;;; Display a data string and advance the cursor a given number of character spaces
;   (length of field).  If length is nil, remain at end of string.

   (multiple-value-bind (cursor-x cursor-y) (send win :read-cursorpos)
      (princ str win)
      (if len
         (send win :set-cursorpos (+ cursor-x (* len (send win :char-width))) cursor-y) ) ) )


;*****************************************************************************
(defun NEXT-FIELD (win len &optional x y)
   (multiple-value-bind (cursor-x cursor-y) (send win :read-cursorpos)
      (setq cursor-x (if x x cursor-x) ) (setq cursor-y (if y y cursor-y) )
      (send win  :set-cursorpos
                 (+ cursor-x (* len (send win :char-width) ) ) cursor-y) ) )
```

```
;*****************************************************************************
(defun COMMAND-EXECUTE ( win pointer-t pointer-r direction
                                table cursorpos-table ret-val redo context)
;;; COMMAND-EXECUTE executes a command from the table using pointer-into-table,
;   pointer-t, to access the command.  Returns ret-val, cursorpos-table, and direction.
;   Pointer-r is pointer-into-return-table.  Retrieve-from-Table gets the sub-list from the
;   table at this pointer location

   (let ( (command (RETRIEVE-FROM-TABLE pointer-t table) ) )
      ;; Make a list of arguments putting a standard set first and appending the arguments
      ;; which are in the table (input)
      (multiple-value-setq (ret-val cursorpos-table direction)
         (apply (first command) (append (list  win pointer-t pointer-r direction table
                                               cursorpos-table ret-val redo context)
                                        (rest command) ) ) )   )
   (values ret-val cursorpos-table direction) )


;*****************************************************************************
(defun CENTER (win pointer-t pointer-r dir table cursorpos-table ret-val redo context string)
;;; A screen description command function which takes a string input and
;   centers it on the screen.
   (ELIMINATE-COMPILER-WARNINGS-FOR-ARGS-NOT-USED pointer-t table context)
   (if (and (not redo) (not (equalp dir 'back) ) )
      (multiple-value-bind (curr-x) (send win :read-cursorpos)       ;y not used, just get x
         ;; Make sure we are on a clean row
         (if (not (= curr-x 0) )
            (ND-TERPRI win) )
         (setf ret-val (INSERT-AT-POINTER pointer-r ret-val string) )
         (send win :display-centered-string string)
         ;; Advance to next row
         (ND-TERPRI win) ) )
   (values ret-val cursorpos-table dir) )


;*****************************************************************************
(defun INIT-CENTER (win string)
;;; Quick initial display and/or redisplay of the center function
   (multiple-value-bind (curr-x) (send win :read-cursorpos)
      ;; Be sure on a clean row
      (if (not (= curr-x 0) )
         (ND-TERPRI win) )
      (send win :display-centered-string string)
      ;; Advance to next row
      (ND-TERPRI win) ) )
```

```
;*****************************************************************************
(defun NEWLINE  ( win pointer-t pointer-r dir table cursorpos-table
                  ret-val redo context &optional (number 1) )
;;; A screen description command function which advances the cursor a number of lines
;   on the display, default number of lines is one (1).
   (ELIMINATE-COMPILER-WARNINGS-FOR-ARGS-NOT-USED pointer-t table context)
   (when (and (not redo) (not (equalp dir 'back) ) )
         (setf ret-val (INSERT-AT-POINTER pointer-r ret-val t) )
         (dotimes (i number)
           (ND-TERPRI win) ) ) )
   (values ret-val cursorpos-table dir) )


;*****************************************************************************
(defun INIT-NEWLINE (win number)
;;; Quick initial display and redisplay, no data recorded
   (dotimes (i number)
      (ND-TERPRI win) ) )


;*****************************************************************************
(defun LEFT ( win pointer-t pointer-r dir table cursorpos-table ret-val
              redo context string &optional type variable)
;;; A screen description command function which displays a string at the left margin of
;   the display with an optional alphabetic or numeric sequence character added.
   (ELIMINATE-COMPILER-WARNINGS-FOR-ARGS-NOT-USED pointer-t table)
   (if (and (not redo) (not (equalp dir 'back) ) )
      (multiple-value-bind (curr-x) (send win :read-cursorpos)
         ;; Make sure we are on a clean row
         (if (not (= curr-x 0) )
            (ND-TERPRI win) )
         (send win :string-out string)
         (let* ((sequence (DISPLAY-ITEM-NUMBER win type variable context) )
                (save-string (if sequence
                                 (string-append string sequence)
                                 string) ) )
            (setf ret-val (INSERT-AT-POINTER pointer-r ret-val save-string) )
            (if variable
               (progn
                  (MY-ASSERT '*variable-data-alist* variable save-string)
                  (MY-ASSERT '*pointer-variable-alist* pointer-r variable)
                  (if (equalp variable '*coa*)
                     (progn
                        (setq *coa* (string-trim " " save-string) )
                        (MY-ASSERT  '*variable-data-alist* 'coa-number
                                    (read-from-string sequence) )
                        ;; Assume that start of COA starts new list of MOA groups
                        (MY-ASSERT '*variable-data-alist* 'moa-number 0) ) ) ) ) )
         ;; Advance to next line
         (ND-TERPRI win) ) )
   (values ret-val cursorpos-table dir) )
```

```
;*****************************************************************************
(defun INIT-LEFT (win string &optional variable)
;;; Quick initial display and redisplay
   (multiple-value-bind (curr-x) (send win :read-cursorpos)
      ;; Make sure we are on a clean row
      (if (not (= curr-x 0) )
         (ND-TERPRI win) ) )
   (let ((new-string (if variable
                         (if (not (equalp (GET-DATA '*variable-data-alist* variable) " ") )
                            (GET-DATA '*variable-data-alist* variable) )
                      string) ) )
      (send win :string-out new-string)
      (if (equalp variable *coa*)
         (progn
            (setq *coa* (string-trim " " new-string) )
            ;; Initialize moa number
            (MY-ASSERT '*variable-data-alist 'moa-number 0) ) ) )
   ;; Next line
   (ND-TERPRI win) ) )


;*****************************************************************************
(defun TFIELD  ( win pointer-t pointer-r dir table cursorpos-table
                 ret-val redo context len str &rest stuff)
;;; A screen description command function which displays a text field.  Field may include
;   multiple lines indicated by % and the names of variables, delimited by &,  for data to be
;   included in the display.
   (ELIMINATE-COMPILER-WARNINGS-FOR-ARGS-NOT-USED pointer-t table)
   (let* ((prev-loc (if (= (first cursorpos-table) -1)
                        nil
                        (nth (first cursorpos-table) (second cursorpos-table) ) ) )
          (prev-x (first prev-loc) ) (prev-y (second prev-loc) )
          (string-value "") (var "") (var? nil) (terpri-count 0)
          (new-cursorpos nil) )
      ;; On redo get cursor position from table and move to that location
      (if redo
         (progn
            (setq new-cursorpos (nth (1+ (first cursorpos-table) )
                                     (second cursorpos-table) ) )
            (send win :set-cursorpos (first new-cursorpos)
                                     (second new-cursorpos) )
            (WIPE-FIELD win len) ) )

      (multiple-value-bind (curr-x curr-y) (send win :read-cursorpos)
         (let ( (new-curr-x curr-x) (new-curr-y curr-y) )
            (dotimes (i (length str) )
               (let ( (ch (subseq str i (1+ i) ) ) )
                  (cond
                     ( (string-equal ch "%")                          ;; Next line indicator
                       (setf new-curr-y (+ new-curr-y 14) )
                       (setf terpri-count (1+ terpri-count) )
                       (send win :set-cursorpos new-curr-x new-curr-y)
                       (setf string-value (string-append string-value ch) ) )
```

```
                ( (string-equal ch "&")                          ;; Variable name indicator
                  (setf var? (TOGGLE var?) )    ;; 1st pass var? is nil, TOGGLE sets to T
                  (if  (not var?)                    ;; 2nd pass TOGGLE sets var? to nil
                       (if (VALOF '*variable-data-alist* var)
                          (setq var (read-from-string (VALOF '*variable-data-alist* var) ) )
                          ;; When sure all are in *variable-data-alist* eliminate this
                          (progn
                             (if (FETCH (read-from-string var)
                                        '*special-global-variables*
                                        '*special-user-variables*)
                                 (setq var (FETCH (read-from-string var)
                                                  '*special-global-variables*
                                                  '*special-user-variables*) )
                                 (setq var (eval (read-from-string var) ) ) )
                             (princ var win)
                             (if (stringp var)
                                (setq string-value (string-append string-value var) )
                                (setq string-value (string-append string-value
                                                                  (format nil "~S" var) ) ) )
                             (setf var "") ) ) ) )

                (t  (if var?                                              ;; otherwise
                       (setf var (string-append var ch) )
                       (progn (setq string-value
                                    (string-append string-value ch) )
                              (princ ch win) ) ) ) ) ) )

(if (> terpri-count (VALOF '*special-global-variables* (list 'terpri-count win) ) )
   (MY-ASSERT '*special-global-variables* (list 'terpri-count win) terpri-count) )

(let ( (type nil) (variable nil) )
   (setf type (first stuff) ) (setf stuff (rest stuff) )
   (setf variable (first stuff) ) (setf stuff (rest stuff) )
   (let ( (sequence (DISPLAY-ITEM-NUMBER win type variable context) ) )
      (if sequence (setf string-value (string-append string-value sequence) ) )
      (if variable
         (progn (MY-ASSERT '*variable-data-alist* variable string-value)
                (MY-ASSERT '*pointer-variable-alist* pointer-r variable)
                (if (equalp variable '*coa*)
                   (progn
                    (setq *coa* (string-trim " " string-value) )
                    (MY-ASSERT  '*variable-data-alist* 'coa-number
                                (read-from-string sequence) )
            ;; Initialize moa number
            (MY-ASSERT '*variable-data-alist* 'moa-number 0) ) ) ) ) ) )
(setf ret-val (INSERT-AT-POINTER pointer-r ret-val string-value) )
```

```
      (if (equalp dir 'back)
         (progn
            (send win :set-cursorpos prev-x prev-y)
            (setf cursorpos-table (PUSH-CURSORPOS (list curr-x curr-y) cursorpos-table) )
            (setf cursorpos-table (POP-CURSORPOS cursorpos-table) )
            (setf cursorpos-table (POP-CURSORPOS cursorpos-table) ) )
         (progn
            (send win :set-cursorpos (+ curr-x (* len (send win :char-width) ) ) curr-y)
            (setf cursorpos-table (PUSH-CURSORPOS (list curr-x curr-y) cursorpos-table) ) ) ) ) )
   (values ret-val cursorpos-table dir) )


;*****************************************************************************
(defun INIT-TFIELD (win len str &rest stuff)
;;; Quick initial display of TFIELD, l      n variable data, if none, use default value
   (multiple-value-bind (curr-x curr-y   nd win :read-cursorpos)
      (let ( (terpri-count 0) (new-curr-x curr-x) (new-curr-y curr-y)
            (var "") (var? nil) (string-value nil)   )
         (dotimes (i (length str) )
            (let ( (ch (subseq str i (1+ i) ) ) )
               (cond
                  ( (string-equal ch "%")                              ;; New line indicator
                    (setf new-curr-y (+ new-curr-y 14) )
                    (setf terpri-count (1+ terpri-count) )
                    (send win :set-cursorpos new-curr-x new-curr-y) )

                  ( (string-equal ch "&")                              ;; Variable name indicator
                    (setf var? (TOGGLE var?) )        ;; 1st pass var? is nil, TOGGLE sets to T
                    (if (not var?)                       ;; 2nd pass TOGGLE sets var? to nil
                       (progn
                          (if (VALOF '*variable-data-alist* var)
                             (setq var (VALOF '*variable-data-alist* var) )
                             (progn
                                (if (FETCH (read-from-string var)
                                           '*special-global-variables*
                                           '*special-user-variables*)
                                   (setq var (FETCH (read-from-string var)
                                                    '*special-global-variables*
                                                    '*special-user-variables*) )
                                   (setq var (eval (read-from-string var) ) ) ) ) )
                          (princ var win)
                          (setq string-value (string-append  string-value
                                                             (format nil "~S" var) ) )
                          (setf var "") ) ) )
                  (t (if  var?
                          (setf var (string-append var ch) )
                          (progn   (setq string-value
                                         (string-append string-value ch) )
                                   (princ ch win) ) ) ) ) )

         (if (> terpri-count (VALOF '*special-global-variables* (list 'terpri-count win) ) )
            (MY-ASSERT '*special-global-variables* (list 'terpri-count win) terpri-count) )
```

```
        (let ( (type nil) (variable nil) )
           (setf type (first stuff) ) (setf stuff (rest stuff) )
           (setf variable (first stuff) ) (setf stuff (rest stuff) )
           (if (equalp variable '*coa*)
              (progn
                 (setq *coa* (string-trim " " string-value) )
              ;; Reset counter for moa number
              (MY-ASSERT '*variable-data-alist* 'moa-number 0) ) ) )
        (send win :set-cursorpos (+ curr-x (* len (send win :char-width) ) ) curr-y) ) ) )


;*****************************************************************************
(defun REDISPLAY-TFIELD (win len str)
;;; Display data from return-table, record only line counts
   (multiple-value-bind (curr-x curr-y) (send win :read-cursorpos)
      (let ( (terpri-count 0) (new-curr-x curr-x) (new-curr-y curr-y) )
         (dotimes (i (length str) )
            (let ( (ch (subseq str i (1+ i) ) ) )
               (cond
                  ( (string-equal ch "%")                          ;; New line indicator
                    (setf new-curr-y (+ new-curr-y 14) )
                    (setf terpri-count (1+ terpri-count) )
                    (send win :set-cursorpos new-curr-x new-curr-y) )
                  ;; Don't bother with variables, data is included in the string from return-table.
                  (t (princ ch win) ) ) ) )
         (if (> terpri-count (VALOF '*special-global-variables* (list 'terpri-count win) ) )
            (MY-ASSERT '*special-global-variables* (list 'terpri-count win) terpri-count) )
         (send win :set-cursorpos (+ curr-x (* len (send win :char-width) ) ) curr-y) ) ) )


;*****************************************************************************
(defun FFIELD (win pointer-t pointer-r dir table cursorpos-table
               ret-val redo context len str)
;;; A screen description command function which displays a fixed string.
   (ELIMINATE-COMPILER-WARNINGS-FOR-ARGS-NOT-USED pointer-t table context)
   (if (and (not redo) (not (equalp dir 'back) ) )
      (progn
         (DISPLAY-STRING win len str)
         (setf ret-val (INSERT-AT-POINTER pointer-r ret-val str) )
   (values ret-val cursorpos-table dir) )
```

```
;*****************************************************************************
(defun IFIELD (win pointer-t pointer-r dir table cursorpos-table
               ret-val redo context len type &optional default variable)
;;; A screen description command function which accepts input for a field.  Type of data
;   acceptable is specified in the screen description file.
   (ELIMINATE-COMPILER-WARNINGS-FOR-ARGS-NOT-USED pointer-t table)
   (let* ( (test (case type
                   (any 'IS-ANY)
                   (alpha 'IS-ALPHA)
                   (decimal 'IS-DECIMAL)
                   (numeric 'IS-NUMBER)
                   (natural 'IS-NATURAL-NUMBER) ) )
          ;; If going backwards lookup variable at this pointer; otherwise change the variable
          ;; name to include the CE number unless a global variable; global variables, denoted
          ;; by leading and trailing asterisks *xyz* will not be renamed
          (ce-var (if (equalp dir 'back)
                      (VALOF '*pointer-variable-alist* pointer-r)
                      (if (and  (string-equal (subseq (string variable) 0 1) "*")
                                (string-equal (subseq (string variable)
                                                      (- (string-length variable) 1)
                                                      (string-length variable) ) "*") )
                          variable                                    ;; global variables
                          (read-from-string
                             (string-append variable "-" *ce-num*) ) ) ));; make others CE specific

          (value (GET-DATA  '*variable-data-alist* ce-var
                            (if (> (length default) len) (subseq default 0 len) default) ) )
          ;; Get cursorpos of the previous field, unless this is first (pointer = -1)
          (prev-loc (if (= (first cursorpos-table) -1)
                        nil
                        (nth (first cursorpos-table) (second cursorpos-table) ) ) )
          (prev-x (first prev-loc) )   (prev-y (second prev-loc) )
          (new-cursorpos nil) (direction nil)   )

      ;; Get the CE number from the variable name when going backward
      (if (equalp dir 'back)
          (setq *ce-num* (subseq (string ce-var)
                                 (search "ce-" (string ce-var) :from-end t :test #'equalp) ) ) )
      ;; Put cursor at recorded position if redoing
      (if redo
          (progn
             (setq new-cursorpos (nth (1+ (first cursorpos-table) )
                                      (second cursorpos-table) ) )
             (send win :set-cursorpos (first new-cursorpos)
                                      (second new-cursorpos) ) ) )
      ;; Make sure we have a cursorpos to record
      (multiple-value-bind (curr-x curr-y) (send win :read-cursorpos)
          ;; Display either default or the previous input value
          (WIPE-FIELD win len )
          (princ value win)
          (send win :set-cursorpos curr-x curr-y)                 ;; Cursor back to start of field
```

```
        (loop
           (MY-ASSERT '*special-global-variables* 'first-ch t)
           (MY-ASSERT '*special-global-variables* 'decimal-pt nil)
           (multiple-value-setq
              (value direction)
              (INSERT-COLUMNS win test len value prev-x prev-y) )
           (if (and (member type '(decimal numeric natural) )
                    (not (equalp value "") )
                    (or (equalp value ".")
                        (not (numberp (read-from-string value nil nil) ) ) ) )
              (progn  (send win :set-cursorpos curr-x curr-y)
                      (WIPE-FIELD win len) )
              (return nil) ) )

        (PUT-IN-SYMBOL-TABLE variable context value)
        (MY-ASSERT '*variable-data-alist* ce-var value)
        (MY-ASSERT '*pointer-variable-alist* pointer-r ce-var)
        ;; Special case for CE type, objective, and comment
        (if (member variable '(ce-type objective comment) )
           (MY-ASSERT  '*ce-data-alist*
                       (read-from-string
                          (string-append variable "-"
                                         (PARSE-BASE-CE-NUM *ce-num*) ) ) value) )
        (setf ret-val (INSERT-AT-POINTER pointer-r ret-val value) )
        (if (equalp direction 'back)
           (progn
              (setf cursorpos-table (PUSH-CURSORPOS (list curr-x curr-y) cursorpos-table) )
              (setf cursorpos-table (POP-CURSORPOS cursorpos-table) )
              (setf cursorpos-table (POP-CURSORPOS cursorpos-table) ) )
           (setf cursorpos-table (PUSH-CURSORPOS (list curr-x curr-y) cursorpos-table) ) )
        (if (member type '(decimal numeric natural) )  ;; Only numbers can be demon args
           (setf ret-val (UPDATE-DEMONS win ret-val ce-var) ) )
        (values ret-val cursorpos-table direction) ) ) )



;*****************************************************************************
(defun INIT-IFIELD (win len def var)
;;; Initialize IFIELD, PFIELD, and MFIELD using recorded data or default values
   (let* ( (ce-var (if (and (string-equal (subseq (string var) 0 1) "*")
                            (string-equal (subseq (string var)
                                                  (- (string-length var) 1)
                                                  (string-length var) ) "*") )
                       var                                          ;; global variables
                       (read-from-string (string-append var "-" *ce-num*) ) ) )  ;; make others CE specific
          (value (GET-DATA '*variable-data-alist* ce-var def) ) )
      (DISPLAY-STRING win len value) ) )
```

```
;*****************************************************************************
(defun SIFIELD ( win pointer-t pointer-r dir table cursorpos-table
                 ret-val redo context len &optional default variable)
;;; A screen description command function which accepts and displays signed numerical
;   input for a field.
   (ELIMINATE-COMPILER-WARNINGS-FOR-ARGS-NOT-USED pointer-t table)
   (let* ( (test 'IS-NATURAL-NUMBER)
         ;; If going backwards look up variable at this pointer; otherwise change the variable
         ;; name to include the CE number unless a global variable; global variables, denoted
         ;; by leading and trailing asterisks *xyz* will not be renamed
         (ce-var (if  (equalp dir 'back)
                      (VALOF '*pointer-variable-alist* pointer-r)
                      (if (and  (string-equal (subseq (string variable) 0 1) "*")
                                (string-equal (subseq (string variable)
                                                      (- (string-length variable) 1)
                                                      (string-length variable) ) "*") )
                          variable                                          ;; global variables
                          (read-from-string
                             (string-append variable "-" *ce-num*) ) ) ) )  ;; CE specific
         (value (GET-DATA  '*variable-data-alist* ce-var
                           (if (> (length default) len) (subseq default 0 len) default) ) )
         ;; Get cursorpos of the previous field, unless this is first (pointer = -1)
         (prev-loc (if (= (first cursorpos-table) -1)
                       nil
                       (nth (first cursorpos-table) (second cursorpos-table) ) ) )
         (prev-x (first prev-loc) ) (prev-y (second prev-loc) )
         (new-cursorpos nil) (direction nil)     )

      ;; Put cursor at recorded position if redoing
      (if redo
         (progn  (setq new-cursorpos (nth (1+ (first cursorpos-table) )
                                          (second cursorpos-table) ) )
                 (send win :set-cursorpos (first new-cursorpos)
                                          (second new-cursorpos) ) ) )
      ;; Make sure we have a cursorpos to record
      (multiple-value-bind (curr-x curr-y) (send win :read-cursorpos)
         ;; Display either default or the previous input value
         (WIPE-FIELD win len )  (princ value win)
         (send win :set-cursorpos curr-x curr-y)              ;; Cursor back to start of field

         (loop
            (MY-ASSERT '*special-global-variables* 'first-ch t)
            (MY-ASSERT '*special-global-variables* 'decimal-pt nil)
            (multiple-value-setq
               (value direction)
               (INSERT-COLUMNS win test len value prev-x prev-y #\-) )
            (if (and  (member type '(decimal numeric natural) )
                      (not (equalp value "") )
                      (or (equalp value ".")
                          (not (numberp (read-from-string value nil nil) ) ) ) )
               (send win :set-cursorpos curr-x curr-y) (WIPE-FIELD win len)
               (return nil) ) )
```

```
        (PUT-IN-SYMBOL-TABLE variable context value)
        (MY-ASSERT '*variable-data-alist* ce-var value)
        (MY-ASSERT '*pointer-variable-alist* pointer-r ce-var)
        (setf ret-val (INSERT-AT-POINTER pointer-r ret-val value) )
        (if (equalp direction 'back)
           (progn
              (setf cursorpos-table (PUSH-CURSORPOS (list curr-x curr-y) cursorpos-table) )
              (setf cursorpos-table (POP-CURSORPOS cursorpos-table) )
              (setf cursorpos-table (POP-CURSORPOS cursorpos-table) ) )
           (setf cursorpos-table (PUSH-CURSORPOS (list curr-x curr-y) cursorpos-table) ) )
        (setf ret-val (UPDATE-DEMONS win ret-val ce-var) )
        (values ret-val cursorpos-table direction) ) ) )


;*****************************************************************************
(defun INIT-SIFIELD (win len def var)
;;; Initialize display of SIFIELD using recorded data or default value
   (let* ( (def (if (and  (> (length def) 0)
                          (or (string-equal (subseq def 0 1) "-")
                              (string-equal (subseq def 0 1) "+") ) )
                    def                                    ;; has a plus or minus first character
                    (string-append " " def) win) )         ;; put a leading space for alignment
           (ce-var (if  (and  (string-equal (subseq (string var) 0 1) "*")
                              (string-equal (subseq (string var)
                                                    (- (string-length var) 1)
                                                    (string-length var) ) "*") )
                        var                                                  ;; global variables
                        (read-from-string (string-append var "-" *ce-num*) ) ) )      ;; CE specific
           (value (GET-DATA '*variable-data-alist* ce-var def) ) )
      (DISPLAY-STRING win len value) ) )


;*****************************************************************************
(defun MFIELD (win pointer-t pointer-r dir table cursorpos-table ret-val
               redo context len menu-list variable)
;;; A screen description command function which is a menu choice field, input arguments are
;   width, menu-list, and variable.  The variable name is required to define the menu
;   function to be called.  Menu function must be built for each MFIELD variable.
   (ELIMINATE-COMPILER-WARNINGS-FOR ARGS-NOT-USED pointer-t table)
   (let* ( (new-value nil) (new-cursorpos nil)
           (test 'IS-ANY)                          ;; Only applies to the "other" option of a menu
           ;; If variable name is global (*xx*) use as is, else chang variable name to
           ;; include the CE number.
           (var (if  (and  (string-equal (subseq (string variable) 0 1) "*")
                           (string-equal (subseq (string variable)
                                                 (- (string-length variable) 1)
                                                 (string-length variable) ) "*") )
                     variable
                     (read-from-string (string-append variable "-" *ce-num*) ) ) )
```

```
        ;; Get previous input data if there is any
        (value  (if (VALOF '*variable-data-alist* var)
                    (VALOF '*variable-data-alist* var)
                    nil) )
        ;; Build name of the function to be called for this variable
        (menu-function (read-from-string
                         (string-append "CHOOSE-" (string-trim "*" variable) ) ) )
        (prev-loc  (if  (= (first cursorpos-table) -1)
                        nil
                        (nth (first cursorpos-table) (second cursorpos-table) ) ) )
        (prev-x (first prev-loc) )   (prev-y (second prev-loc) ) )

    ;; If this function is entered with direction = back then skip, otherwise have to take
    ;; some kind of action twice, only way to change direction in this function is choose
    ;; "Other" and type an entry followed by the new direction (backspace or return)
    (if (equalp dir 'back)
        (progn (send win :set-cursorpos prev-x prev-y)                    ;; move backward
               (setf cursorpos-table (POP-CURSORPOS cursorpos-table) )
               (return-from MFIELD (values ret-val cursorpos-table dir) ) ) )

    ;; On redo get cursor position from table and move to that location
    (if redo
        (progn (setq new-cursorpos (nth (1+ (first cursorpos-table) )
                                        (second cursorpos-table) ) )
               (send win :set-cursorpos (first new-cursorpos)
                                        (second new-cursorpos) ) ) )

    ;; If this is not a redo or is first, i.e. no cursorpos table, get the actual location
    (multiple-value-bind (curr-x curr-y) (send win :read-cursorpos)
        ;; If we have a value from previous input, display it so he knows what it is
        (if value  (progn (WIPE-FIELD win len)
                          (princ value win)
                          ;; Put cursor back at first of the field
                          (send win :set-cursorpos curr-x curr-y) ) )

        (multiple-value-setq (new-value dir)
            (funcall menu-function win menu-list value len curr-x curr-y prev-x prev-y) )

        ;; Now record the data
        (PUT-IN-SYMBOL-TABLE variable context new-value)
        (MY-ASSERT '*variable-data-alist* var new-value)
        (MY-ASSERT '*pointer-variable-alist* pointer-r var)
        (setf ret-val (INSERT-AT-POINTER pointer-r ret-val new-value) )
        ;; Record the cursor position
        (if (equalp dir 'back)
            ;; In case this is the last field and not recorded yet put it on the list
            ;; then pop it and the next off to go backwards
            (progn
                (setf cursorpos-table (PUSH-CURSORPOS (list curr-x curr-y) cursorpos-table) )
                (setf cursorpos-table (POP-CURSORPOS cursorpos-table) )
                (setf cursorpos-table (POP-CURSORPOS cursorpos-table) ) )
            (setf cursorpos-table (PUSH-CURSORPOS (list curr-x curr-y) cursorpos-table) ) )
        (values ret-val cursorpos-table dir) ) ) )
```

```
;*****************************  ***************************************************
(defun PARSE-BASE-CE-NUM (instring)
;;; Return the base CE number from a string, assumes that COA number is always one (1)
;   digit, but that the CE sequence number may be two digits.
;   Returns nil if length not 6 or 7.
   (cond
      ( (= (length instring) 6)
         (subseq instring 4 6) )
      ( (= (length instring) 7)
         (subseq instring 4 7) )
      (t nil) ) )


;*******************************************************************************
(defun CEFIELD (win pointer-t pointer-r dir table cursorpos-table
                    ret-val redo context len)
;;; A screen description command function which is a special modification of IFIELD for the
;   input of CE number on the assignment worksheet.  Screen description input requires only
;   length of field; type is always "alphanumeric", there is no default value, and variable is
;   "ce-number".  Automatically put up "CE-", add the COA number, if there is a value in the
;   return-table, get it and use it, then accept input of the base CE number (ACA +
;   sequence-number).
   (ELIMINATE-COMPILER-WARNINGS-FOR-ARGS-NOT-USED pointer-t dir table)
   (let* ( (coa# (read-from-string (subseq *coa* 4 5) ) )
           ; Get cursorpos of the previous field, unless this is first (pointer = -1)
           (prev-loc  (if  (= (first cursorpos-table) -1)
                           nil
                           (nth (first cursorpos-table) (second cursorpos-table) ) ) )
           (prev-x (first prev-loc) )   (prev-y (second prev-loc) )
           ; If this is redo, find out where cursor should be
           (start-cursorpos (if  redo
                                 (nth (1+ (first cursorpos-table) )
                                      (second cursorpos-table) )
                                 nil) )
           (direction nil)  (value nil)  (test 'IS-ALPHANUMERIC)
           (new-ce-num nil) (old-ce-num nil)  )

      (setq *ce-num* (string-append "CE-" (string (digit-char coa#) ) ) )
      (multiple-value-bind (start-x start-y) (send win :read-cursorpos)
         (if start-cursorpos (progn  (setq start-x (first start-cursorpos) )
                                     (setq start-y (second start-cursorpos) ) ) )
         (send win :set-cursorpos start-x start-y)          ;; make sure in the right place

         ; If a previous value exists, get it and use it
         (if (RETRIEVE-FROM-TABLE pointer-r ret-val)
            (progn (setq old-ce-num (RETRIEVE-FROM-TABLE pointer-r ret-val) )
                   (setq value (PARSE-BASE-CE-NUM old-ce-num) )
                   (setq *ce-num* old-ce-num)
                   ;; Insure right COA #, in case backed up to prior COA
                   (setq coa# (read-from-string (subseq *ce-num* 3 4) ) ) ) )
```

```
(WIPE-FIELD win len )
(princ *ce-num* win)
; Position the cursor just after the "CE-#", even if we have the full number
(send win :set-cursorpos
      (+ start-x (* 4 (send win :char-width) ) ) start-y)

(let ( (base-len 3)                              ;; Accept a maximum of 3 characters
      (start-val value) )                                ;; Save starting value
   (loop
      (setq new-ce-num (string-append "CE-" (string (digit-char coa#) ) ) )
      (setq value start-val)                  ;; Reset to starting value in case looping

      (multiple-value-setq
         (value direction)
         (INSERT-COLUMNS win test base-len value prev-x prev-y) )

      (send *PROMPT-WINDOW* :deactivate)                  ;; In case it is displayed
      (if (equalp value nil)  (setq value "") )
      (setq value (string-upcase (string-trim " " value) ) )
      (setq new-ce-num (string-append new-ce-num value) )
      ; If the CE number has been changed, delete the old one from the list
      (if (not (equalp old-ce-num new-ce-num) )
         (setq *ce-number-list* (delete old-ce-num *ce-number-list* :test #'equalp) ) )

      ; Don't accept null/blank, or single character, or first character not alpha,
      ; or 2nd/3rd character not a number, or duplicate CE and not the same as
      ; previous input value
      (if (or (equalp value "") (< (length value) 2)
            (not (IS-ALPHA (character (subseq value 0 1) ) ) )
            (not (IS-NUMBER (character (subseq value 1 2) ) ) )
            (and (= (length value) 3)
                 (not (IS-NUMBER (character (subseq value 2 3) ) ) ) )
            (and (DUPLICATE-CE-NUMBER new-ce-num)
                 (not (equalp new-ce-num *ce-num*) ) ) )
         (progn
            (DISPLAY-PROMPT 'CE-NAME-PROMPT (+ start-x 175) start-y)
            (send win :set-cursorpos start-x start-y)
            (WIPE-FIELD win len)
            (princ *ce-num* win)
            (send win :set-cursorpos
                  (+ start-x (* 4 (send win :char-width) ) ) start-y) )
         (return nil) ) ) )

(set 1 *ce-num* new-ce-num)
(if (not (DUPLICATE-CE-NUMBER *ce-num*) )
   (setq *ce-number-list* (append *ce-number-list* (list *ce-num*) ) ) )
(PUT-IN-SYMBOL-TABLE 'ce-number context *ce-num*)
(MY-ASSERT '*variable-data-alist* 'ce-number *ce-num*)
(MY-ASSERT '*pointer-variable-alist* pointer-r *ce-num*)
```

```
        ; Check for previous input for this base CE, assume if we have type we have other.
        ; Store the data in *variable-data-alist* to be recovered as needed
        (if (VALOF '*ce-data-alist*
                   (read-from-string
                      (string-append "ce-type-" (PARSE-BASE-CE-NUM *ce-num*) ) ) )
           (progn
              (let ( (base-ce-num (PARSE-BASE-CE-NUM *ce-num*) ) )
                 (MY-ASSERT  '*variable-data-alist*
                             (read-from-string (string-append "ce-type-" *ce-num*) )
                             (VALOF '*ce-data-alist*
                                    (read-from-string
                                       (string-append "ce-type-" base-ce-num) ) ) )
                 (MY-ASSERT  '*variable-data-alist*
                             (read-from-string (string-append "objective-" *ce-num*) )
                             (VALOF '*ce-data-alist*
                                    (read-from-string
                                       (string-append "objective-" base-ce-num) ) ) )
                 (MY-ASSERT  '*variable-data-alist*
                             (read-from-string (string-append "comment-" *ce-num*) )
                             (VALOF '*ce-data-alist*
                                    (read-from-string
                                       (string-append "comment-" base-ce-num) ) ) ) ) ) )
        (setf ret-val (INSERT-AT-POINTER pointer-r ret-val *ce-num*) )
        (if (equalp direction 'back)
           (progn
              (setf cursorpos-table (PUSH-CURSORPOS (list start-x start-y) cursorpos-table) )
              (setf cursorpos-table (POP-CURSORPOS cursorpos-table) )
              (setf cursorpos-table (POP-CURSORPOS cursorpos-table) ) )
           (setf cursorpos-table (PUSH-CURSORPOS (list start-x start-y) cursorpos-table) ) )
        (values ret-val cursorpos-table direction) ) ) )


;*****************************************************************************
(defun INIT-CEFIELD (win len)
;;; initialize CEFIELD display
   (if *ce-number-list*
      ;; If there is a list of CE numbers
      (progn
         (setq *ce-num* (first *ce-number-list*) )
         (setq *ce-number-list* (rest *ce-number-list*) ) )
      (setq *ce-num* "CE-") )                ; otherwise just set to CE-

   (DISPLAY-STRING win len *ce-num*) )
```

```
;*****************************************************************************
(defun DFIELD (win pointer-t pointer-r dir table cursorpos-table
               ret-val redo context len default variable)
;;; A screen description command function which is used to display a data field using
;   previously entered data which is stored in *variable-data-alist*.  If going backwards
;   or redoing skip completely, data is already displayed.
   (ELIMINATE-COMPILER-WARNINGS-FOR-ARGS-NOT-USED pointer-t table context)
   (if (and (not redo) (not (equalp dir 'back) ) )
      ;; Get value of the variable from data table if its there, else use default value
      (let* ( (value (GET-DATA  '*variable-data-alist* variable
                                (if (> (length default) len)
                                   (subseq default 0 len)
                                   default) ) ) )
         (DISPLAY-STRING win len value)
         (setf ret-val (INSERT-AT-POINTER pointer-r ret-val value) ) ) )
   (values ret-val cursorpos-table dir) )


;*****************************************************************************
(defun INIT-DFIELD (win len default var-name)
;;; Get value of the var-name from data list if its there, else use default value
   (let ( (value (GET-DATA   '*variable-data-alist* var-name
                             (if (> (length default) len)
                                (subseq default 0 len)
                                default) ) ) )
      (DISPLAY-STRING win len value) ) )


;*****************************************************************************
(defun PFIELD (win pointer-t pointer-r dir table cursorpos-table
               ret-val redo context len type default variable)
;;; A screen description command function which is basically IFIELD with a prompt window
;   determined by the input name of the variable. The variable name is made CE specific
;   by adding the CE number to the end, unless it is a global (*xyz*) name which will not
;   be changed.  DISPLAY-PROMPT calls a function named "variable-PROMPT" to write
;   the prompt in the pop-up window.
   (ELIMINATE-COMPILER-WARNINGS-FOR-ARGS-NOT-USED pointer-t table)
   (let* ( (test (case type
                   (any 'IS-ANY)
                   (alpha 'IS-ALPHA)
                   (decimal 'IS-DECIMAL)
                   (numeric 'IS-NUMBER)
                   (natural 'IS-NATURAL-NUMBER) ) )
          ;; Build prompt function name without *, if any there
          (prompt-var (read-from-string (string-append (string-trim "*" variable) "-PROMPT") ) )
```

```
        ;; If it is a global variable name "*__*" then use name with no modification,
        ;; otherwise add CE number to end for new name
        (variable  (if  (equalp dir 'back)
                        (VALOF '*pointer-variable-alist* pointer-r)
                        (if (and  (string-equal (subseq (string variable) 0 1) "*")
                                  (string-equal (subseq (string variable)
                                                        (- (string-length variable) 1)
                                                        (string-length variable) ) "*") )
                            variable
                            (read-from-string (string-append variable "-" *ce-num*) ) ) ) )
        ;; Get value of the variable from data table if its there, else use default value
        (value (GET-DATA  '*variable-data-alist* variable
                          (if (> (length default) len) (subseq default 0 len) default) ) )
        ;; Get cursorpos of the previous field, unless this is first (pointer = -1)
        (prev-loc  (if  (= (first cursorpos-table) -1)
                        nil
                        (nth (first cursorpos-table) (second cursorpos-table) ) ) )
        (prev-x (first prev-loc) )   (prev-y (second prev-loc) )
        (new-cursorpos nil) (direction nil)      )

    ;; Put cursor at recorded position if redoing
    (if redo
        (progn
            (setq new-cursorpos (nth (1+ (first cursorpos-table) )
                                     (second cursorpos-table) ) )
            (send win :set-cursorpos  (first new-cursorpos)
                                      (second new-cursorpos) ) ) )
    ;; Make sure we have a cursorpos to record
    (multiple-value-bind (curr-x curr-y) (send win :read-cursorpos)
        ;; Display either default or the previous input value
        (WIPE-FIELD win len )   (princ value win)
        (send win :set-cursorpos curr-x curr-y)                    ;; Cursor back to start of field
        (DISPLAY-PROMPT prompt-var (+ curr-x 175) curr-y)

        (loop
            (MY-ASSERT '*special-global-variables* 'first-ch t)
            (MY-ASSERT '*special-global-variables* 'decimal-pt nil)
            (multiple-value-setq
                (value direction)
                (INSERT-COLUMNS win test len value prev-x prev-y) )
            (if (and  (member type '(decimal numeric natural) )
                      (not (equalp value "") )
                      (or (equalp value ".")
                          (not (numberp (read-from-string value nil nil) ) ) ) )
                (send win :set-cursorpos curr-x curr-y) (WIPE-FIELD win len)
                (return nil) ) )
        (send *PROMPT-WINDOW* :deexpose)
        (send *PROMPT-WINDOW* :deactivate)
        (PUT-IN-SYMBOL-TABLE variable context value)
        (MY-ASSERT '*variable-data-alist* variable value)
        (MY-ASSERT '*pointer-variable-alist* pointer-r variable)
        (setf ret-val (INSERT-AT-POINTER pointer-r ret-val value) )
```

```
        (if (equalp direction 'back)
           (progn
              (setf cursorpos-table (PUSH-CURSORPOS (list curr-x curr-y) cursorpos-table) )
              (setf cursorpos-table (POP-CURSORPOS cursorpos-table) )
              (setf cursorpos-table (POP-CURSORPOS cursorpos-table) ) )
           (setf cursorpos-table (PUSH-CURSORPOS (list curr-x curr-y) cursorpos-table) ) )
        (if (member type '(decimal numeric natural) )  ;; Only numbers can be demon args
           (setf ret-val (UPDATE-DEMONS win ret-val variable) ) )
        (values ret-val cursorpos-table direction) ) ) )


;*****************************************************************************
(defun PTFIELD (win pointer-t pointer-r dir table cursorpos-table
                ret-val redo context len default variable)
;;; A screen description command function which is a special case of PFIELD built
;   specifically for the Module 3 scale value variables which are input with names ending
;   in "-#-scale", where # is the COA number. Those 8 characters are deleted from the
;   name and the first part is prefixed with "scale-" and used to call the prompt window.
   (ELIMINATE-COMPILER-WARNINGS-FOR-ARGS-NOT-USED pointer-t dir table)
   (let* ( (test 'IS-SCALE-VAL)                                    ;; Limit values to legal scale
         ;; Build prompt function name by deleting the last 8 characters of variable name
         ;; and adding "scale-" to the front of the result
         (prompt-var (read-from-string
                        (string-append "scale-"
                                       (subseq (string variable)
                                               0 (- (string-length variable) 8) ) ) ) )
         ;; Get value of the variable from data table if its there, else use default value
         (value (GET-DATA  '*variable-data-alist* variable
                           (if (> (length default) len) (subseq default 0 len) default) ) )
         (prev-loc  (if  (= (first cursorpos-table) -1)
                         nil
                         (nth (first cursorpos-table) (second cursorpos-table) ) ) )
         (prev-x (first prev-loc) )   (prev-y (second prev-loc) )
         (new-cursorpos nil )   (direction nil)    )

      ;; Put cursor at recorded position if redoing
      (if redo (progn
                  (setq new-cursorpos (nth (1+ (first cursorpos-table) )
                                           (second cursorpos-table) ) )
                  (send win :set-cursorpos (first new-cursorpos)
                                           (second new-cursorpos) ) ) )
      ;; Make sure we have a cursorpos to record
      (multiple-value-bind (curr-x curr-y) (send win :read-cursorpos)
         ;; Display either default or the previous input value
         (WIPE-FIELD win len)   (princ value win)
         (send win :set-cursorpos curr-x curr-y)                   ;; Cursor back to start of field
         (DISPLAY-PROMPT prompt-var (+ curr-x 175) curr-y)

         (loop
            (MY-ASSERT '*special-global-variables* 'first-ch t)
            (MY-ASSERT '*special-global-variables* 'decimal-pt nil)
```

```
            (multiple-value-setq
               (value direction)
               (INSERT-COLUMNS win test len value prev-x prev-y) )
            (if (and  (member type '(decimal numeric natural) )
                      (not (equalp value "") )
                      (or (equalp value ".")
                          (not (numberp (read-from-string value nil nil) ) )
                          (> (read-from-string value nil nil) 9) ) )          ;; Single digit only
               (progn (send win :set-cursorpos curr-x curr-y)
                      (WIPE-FIELD win len) )
               (return nil) ) )
         (send *PROMPT-WINDOW* :deexpose)
         (send *PROMPT-WINDOW* :deactivate)
         (MY-ASSERT '*variable-data-alist* variable value)
         (MY-ASSERT '*pointer-variable-alist* pointer-r variable)
         (PUT-IN-SYMBOL-TABLE variable context value)
         (setf ret-val (INSERT-AT-POINTER pointer-r ret-val value) )
         (if (equalp direction 'back)
            (progn
               (setf cursorpos-table (PUSH-CURSORPOS (list curr-x curr-y) cursorpos-table) )
               (setf cursorpos-table (POP-CURSORPOS cursorpos-table) )
               (setf cursorpos-table (POP-CURSORPOS cursorpos-table) ) )
            (setf cursorpos-table (PUSH-CURSORPOS (list curr-x curr-y) cursorpos-table) ) )
         (if (member type '(decimal numeric natural) )   ;; Only numbers can be demon args
            (setf ret-val (UPDATE-DEMONS win ret-val variable) ) )
         (values ret-val cursorpos-table direction) ) ) )


;*****************************************************************************
(defun MAA-PFIELD (win pointer-t pointer-r dir table cursorpos-table
                   ret-val redo context len)
;;; A screen description command function which is a special case of PFIELD built
;   specifically for the Module 1 solicitation of the main axis of attack for each COA.
;   Variable name is "main-atk-axis" plus *coa*.  Only alphabetic characters allowed.
   (ELIMINATE-COMPILER-WARNINGS-FOR-ARGS-NOT-USED pointer-t table)
   (let* ( (test 'IS-ALPHA)
           (prompt-var 'ATK-AXIS-PROMPT)
           ;; Build the variable name (main-atk-axis) to include the COA number
           (variable (if (equalp dir 'back)
                         (VALOF '*pointer-variable-alist* pointer-r)
                         (read-from-string (string-append "main-atk-axis-" *coa*) ) ) )
           ;; Get value of the variable from data table if its there, else use blank
           (value (GET-DATA '*variable-data-alist* variable) )
           (prev-loc (if (= (first cursorpos-table) -1)
                         nil
                         (nth (first cursorpos-table) (second cursorpos-table) ) ) )
           (prev-x (first prev-loc) )   (prev-y (second prev-loc) )
           (new-cursorpos nil )   (direction nil)    )
```

```
    ;; Put cursor at recorded position if redoing
    (if redo (progn (setq new-cursorpos  (nth (1+ (first cursorpos-table) )
                                             (second cursorpos-table) ) )
                    (send win :set-cursorpos (first new-cursorpos)
                                             (second new-cursorpos) ) ) )
    ;; Make sure we have a cursorpos to record
    (multiple-value-bind (curr-x curr-y) (send win :read-cursorpos)
       ;; Display either default or the previous input value
       (WIPE-FIELD win len)   (princ value win)
       (send win :set-cursorpos curr-x curr-y)                ;; Cursor back to start of field
       (send *PROMPT-WINDOW* :set-size 215 215)
       (DISPLAY-PROMPT prompt-var (+ curr-x 175) curr-y)

       (multiple-value-setq
          (value direction)
          (INSERT-COLUMNS win test len value prev-x prev-y) )

       (send *PROMPT-WINDOW* :deexpose)
       (send *PROMPT-WINDOW* :deactivate)
       (PUT-IN-SYMBOL-TABLE variable context value)
       (MY-ASSERT '*variable-data-alist* variable value)
       (MY-ASSERT '*pointer-variable-alist* pointer-r variable)
       (setf ret-val (INSERT-AT-POINTER pointer-r ret-val value) )

       (if (equalp direction 'back)
          (progn
             (setf cursorpos-table (PUSH-CURSORPOS (list curr-x curr-y) cursorpos-table) )
             (setf cursorpos-table (POP-CURSORPOS cursorpos-table) )
             (setf cursorpos-table (POP-CURSORPOS cursorpos-table) ) )
          (setf cursorpos-table (PUSH-CURSORPOS (list curr-x curr-y) cursorpos-table) ) )

       (values ret-val cursorpos-table direction) ) ) )



;*****************************************************************************
(defun INIT-MAA-PFIELD (win len)
;;; Initialize the main avenue of attack field

   (let* ((variable (read-from-string (string-append "main-atk-axis" *coa*) ) )
          (string (GET-DATA '*variable-data-alist* variable) ) )
     (DISPLAY-STRING win len string) ) )
```

```
;*****************************************************************************
(defun MOA-PFIELD (win pointer-t pointer-r dir table cursorpos-table
                   ret-val redo context len)
;;; A screen description command function which is a special case of PFIELD built
;   specifically for the Module 1 solicitation of the avenue/belt/box name. Variable name
;   is built from COA number, the moa type (av/blt/box) and the moa number in this COA.
  (ELIMINATE-COMPILER-WARNINGS-FOR-ARGS-NOT-USED pointer-t table)
  (let* ( (test 'IS-ANY)                                  ;; Any standard characters accepted
        (prompt-var 'MOA-NAME-PROMPT)
        ;; MOA-number is zerood to start each COA, increment each time in here unless
        ;; redoing.  MOA-number will always be 0 when editing from the top but is not
        ;; needed during edit since it's already known.
        (moa-number (if (or redo (equalp dir 'back) )
                        (VALOF '*variable-data-alist* 'moa-number)
                        (+ 1 (VALOF '*variable-data-alist* 'moa-number) ) ) )
        ;; Build the variable name
        (variable (if (or redo (equalp dir 'back) )
                      (VALOF '*pointer-variable-alist* pointer-r)
                      (read-from-string (string-append *coa* "-" *moa-type* "-"
                                                       (format nil "~s" moa-number) ) ) ) )
        ;; Get value of the variable from data list if its there, else use blank
        (value (GET-DATA '*variable-data-alist* variable) )
        (prev-loc (nth (first cursorpos-table) (second cursorpos table) ) )
        (prev-x (first prev-loc) )   (prev-y (second prev-loc) )
        (new-cursorpos nil )   (direction nil)    )

    ;; Put cursor at recorded position if redoing
    (if redo (progn (setq new-cursorpos (nth (1+ (first cursorpos-table) )
                                             (second cursorpos-table) ) )
                    (send win :set-cursorpos (first new-cursorpos)
                                             (second new-cursorpos) ) ) )
    ;; Make sure we have a cursorpos to record
    (multiple-value-bind (curr-x curr-y) (send win :read-cursorpos)
       ;; Display either default or the previous input value
       (WIPE-FIELD win len)   (princ value win)
       (send win :set-cursorpos curr-x curr-y)                ;; Cursor back to start of field
       (if (not redo)                                    ;; Don't bother with prompt on redo
          (progn (send *PROMPT-WINDOW* :set-size 215 215)
                 (DISPLAY-PROMPT prompt-var (+ curr-x 175) curr-y) ) )

       (multiple-value-setq
          (value direction)
          (INSERT-COLUMNS win test len value prev-x prev-y) )

       (send *PROMPT-WINDOW* :deactivate)                                ;; In case it is up
       (PUT-IN-SYMBOL-TABLE variable context value)
       (MY-ASSERT '*variable-data-alist* variable value)
       (MY-ASSERT '*variable-data-alist* 'moa-number moa-number)
       (MY-ASSERT '*pointer-variable-alist* pointer-r variable)
       (setf ret-val (INSERT-AT-POINTER pointer-r ret-val value) )
```

```
        (if (equalp direction 'back)
           (progn
              (setf cursorpos-table (PUSH-CURSORPOS (list curr-x curr-y) cursorpos-table) )
              (setf cursorpos-table (POP-CURSORPOS cursorpos-table) )
              (setf cursorpos-table (POP-CURSORPOS cursorpos-table) ) )
           (setf cursorpos-table (PUSH-CURSORPOS (list curr-x curr-y) cursorpos-table) ) )

        (values ret-val cursorpos-table direction) ) ) )


;*****************************************************************************
(defun INIT-MOA-PFIELD (win len)
;;; Initialize the MOA field
   (let* ((moa-number (+ 1 (VALOF '*variable-data-alist* 'moa-number) ) )
          ;; Build variable name
          (variable (read-from-string (string-append *coa* "-" *moa-type* "-"
                                                     (format nil "~S" moa-number) ) ) )
          ;; Get value from the data list, else use blank
          (string (GET-DATA '*variable-data-alist* variable) ) )
     (MY-ASSERT 'variable-data-alist* 'moa-number moa-number)
     (DISPLAY-STRING win len string) ) )


;*****************************************************************************
(defun MSETS (win pointer-t pointer-r dir table cursorpos-table
                  ret-val redo context name description &rest commands)
;;; A screen description command function which allows the definition of collection of
;   fields, rows, or nested sets. This provides flexibility to the system and allows the
;   screen display to be determined by the input rather than be fixed in advance.

   (let ( (new-context (list 'MSETS name) ) )
     (multiple-value-setq (ret-val cursorpos-table dir)
        (SET-OR-ROW  win pointer-t pointer-r dir table cursorpos-table ret-val
                     redo context new-context name description &rest commands) ) )
   (values ret-val cursorpos-table dir) )



;*****************************************************************************
(defun MROW (win pointer-t pointer-r dir table cursorpos-table
                 ret-val redo context name description &rest commands)
;;; A screen description command function which provides for a collection of fields which
;   make up a row of the display.  Rows can then be combined into sets defined by MSETS.

   (let ( (new-context (list 'MROW name) ) )
     (multiple-value-setq (ret-val cursorpos-table dir)
        (SET-OR-ROW  win pointer-t pointer-r dir table cursorpos-table ret-val
                     redo context new-context name description &rest commands) ) )
     (values ret-val cursorpos-table dir) )
```

```
;*******************************************************************************
(defun SET-OR-ROW (win pointer-t pointer-r dir table cursorpos-table
                   ret-val redo context new-context description commands)
;;; This function contains the recursive loops used by both MSETS and MROW to execute
;   the commands in a set (row) and iterate the set (row).
   (let* ( (return-value ret-val)  (desc nil)  (direction dir)
         ;; The number of commands included in this set plus 2 for name and description args
         (max-command-ptr (+ (length commands) 2) )
         ;; Next command is at location 3 unless going back, if going back when get here
         ;; the next command to execute is the last one of this set.
         (command-ptr (if (equalp dir 'back) max-command-ptr 3) )
         ;; Get values from the current return table at this pointer location
         (existing-values (RETRIEVE-FROM-TABLE pointer-r ret-val) )
         ;; If retrieved value is a list then use the length of that list, else 0.
         (max-position (if (listp existing-values) (length existing-values) 0) )
         ;; If going back set position at end of the list, otherwise set at 1
         (position (if (equalp dir 'back) max-position 1) )
         ;; Set pointer into the input table for the current command
         (new-pointer-t (append pointer-t (list command-ptr) ) )
         ;; Set pointer into the return table for the current command
         (new-pointer-r (append pointer-r (list (1- position) command-ptr) ) ) )

      ;; Store new-context label with the input argument context values
      (MY-ASSERT '*special-global-variables* (append new-context '(context) ) context)
      ;; Check the description argument to see if it is a variable or just a string or nothing
      (setq desc (string-trim " " description) )                    ;; Trim spaces off the ends

      ;; If no description input set to nil
      ;; If variable name input get the value, else use as input
      (if (> (length desc) 0)
         (if (equalp (subseq desc 0 1) "&")                  ;; If first char = &, it is a variable
            (progn (setq desc (string-trim "&" desc) )                    ;; Strip & off
                   (setq description (eval (read-from-string desc) ) ) ) )  ;; Get value of variable
         (setq description nil) )                                  ; If nothing there set it to nil.

      ;; Store description name in list with any previous set descriptors
      (MY-ASSERT '*special-global-variables* 'desc
                 (append (VALOF '*special-global-variables* 'desc) (list description) ) )

      ;; Loop all the iterations of this command, MSETS or MROW.
      (loop
         ;; Store iteration count and maximum count for this set context
         (MY-ASSERT '*special-global-variables* (append new-context '(iter) ) position)
         (MY-ASSERT '*special-global-variables*
                    (append new-context '(max) ) (max position max-position) )
```

```
;; Loop over all the commands in this set
(loop
   (multiple-value-setq
      (return-value cursorpos-table direction)
      (COMMAND-EXECUTE win new-pointer-t new-pointer-r direction
                       table cursorpos-table return-value
                       (if (ITEM-EXISTS new-pointer-r return-value) t nil)
                       new-context) )

   ;; Set up pointer for the next command
   (if (equalp direction 'back)
      ;; If going back, decrement the current pointer
      (let ( (r-len (1- (length new-pointer-r) ) )
             (t-len (1- (length new-pointer-t) ) ) )
         (setf new-pointer-r (DECREMENT-POINTER new-pointer-r r-len) )
         (setf new-pointer-t (DECREMENT-POINTER new-pointer-t t-len) )
         (setf command-ptr (1- command-ptr) )
         ;; Set change flag when backup during initial input
         (if (not redo) (setq *change-flag* T) ) )
      ;; Going forward, increment the current pointer
      (let ( (r-len (1- (length new-pointer-r) ) )
             (t-len (1- (length new-pointer-t) ) ) )
         (setf new-pointer-r (INCREMENT-POINTER new-pointer-r r-len) )
         (setf new-pointer-t (INCREMENT-POINTER new-pointer-t t-len) )
         (setf command-ptr (1+ command-ptr) ) ) )

   ;; If pointer is less than 3 or greater than maximum, exit loop
   (if (or (< command-ptr 3) (> command-ptr max-command-ptr) )
      (return nil) ) )                                          ; Exit loop

(if (equalp direction 'back)
   ;; If going backwards decrease position count by one, reset command pointer
   (progn (setf command-ptr max-command-ptr)
          (setf position (1- position) ) )
   ;; Going forward set command pointer to 3, set position count plus one
   (progn (setf command-ptr 3)
          (setf position (+ position 1) )
          ; Put up ASK-END menu if appropriate
          (if  (or (not description)
                   (and redo (not *change-flag*)
                             (not (MORE-DATA cursorpos-table) ) ) )
               ;; If no menu description exists OR if redoing and not changing
               ;; and no more data, then no menu.  Force it out of loop.
               (setq direction 1)
               (if (or (MORE-DATA cursorpos-table)
                       (and (numberp direction) (eq direction 0) ) )
                  ;; If there is more data (redo or change) OR direction=0, no menu
                  (setq direction 'tab)
                  (if (not (numberp direction) )
                     ;; On initial input, at end of data if changing, and
                     ;; direction not a number, put up the menu.
                     (setq direction (ASK-END win) ) ) ) ) ) )
```

```
;; Set new pointers into return and input tables
(setf new-pointer-r
      (SUBSTITUTE-POS (- (length new-pointer-r) 2) (- position 1)
                      (SUBSTITUTE-POS (1- (length new-pointer-r) )
                                      command-ptr new-pointer-r) ) )
(setf new-pointer-t
      (SUBSTITUTE-POS (1- (length new-pointer-t) ) command-ptr
                      new-pointer-t) )

;; When position less than one (backing up) or all elements this set have been
;; done for a redo, but more data exists.
(if (or (< position 1)
        (and redo (> position max-position) (MORE-DATA cursorpos-table) ) )
   ;; Drop last descriptor from list, going to the next higher level,
   ;; and if direction is a number, decrement by one and return
   (progn
      (MY-ASSERT '*special-global-variables* 'desc
                 (butlast (VALOF '*special-global-variables* 'desc) ) )
      (return (values return-value cursorpos-table
                 (if (numberp direction)
                    (- direction 1)
                    direction) ) ) ) )

;; If direction is greater than 0, back out to next level
(if (and (numberp direction) (> direction 0) )
   (progn
      (MY-ASSERT '*special-global-variables* 'desc
                 (butlast (VALOF '*special-global-variables* 'desc) ) )
      (return (values return-value cursorpos-table (- direction 1) ) ) ) )

;; Otherwise loop at this level.
(if (> position max-position) (setf max-position position) ) ) ) )
```

```
;*****************************************************************************
(defun MORE-DATA (cursorpos-table)
;;; Used by MSETS and MROW, on redo or when changing, to determine if more data
;   exists for processing.  If more cursor positions recorded than current pointer,
;   then there must be more data.

   (> (length (second cursorpos-table) )
      (1+ (first cursorpos-table) ) ) )


;*****************************************************************************
(defun INSERT-COLUMNS (win test width value back-x back-y &optional sp-char)
;;; Function to accept keyboard input from the user.  Each character input is checked for
;   correct type as specified by the calling function.  The only other input accepted are
;   the keys return, tab, end, back-space, and rubout.  Returns the input as a string and
;   the cursor movement direction.

   (send win :clear-input)            ;; Clear any extra characters which may have been struck
   (let ( (ret-val "")  (position 0)  (decimal-pos nil)  (change nil) )
      (loop
         (let ( (ch (read-character win) ) )
            (if (or (funcall test ch)
                    (member ch '(#\tab #\return #\back-space #\rubout #\end) ) )
               (case ch
                  ( (#\tab #\return #\end)
                     (if change
                        (progn (dotimes (i (- width position 1) )
                                  (send win :delete-char) (send win :insert-string " ") )
                               (send win :increment-cursorpos (send win :char-width) 0)
                               (return (values ret-val (case ch (#\end 'end) (t 'tab) ) ) ) )
                        (progn (send win :increment-cursorpos ( * (- width position)
                                                                 (send win :char-width) ) 0)
                               (return (values value (case ch (#\end 'end) (t 'tab) ) ) ) ) ) )
                  (#\back-space
                     (if change
                        (progn (dotimes (i (- width position) )
                                  (send win :delete-char)
                                  (send win :insert-string " ") )
                               (send win :set-cursorpos back-x back-y)
                               (return (values ret-val 'back) ) )
                        (progn (send win :set-cursorpos back-x back-y)
                               (return (values value 'back) ) ) ) )

                  (#\rubout
                     (if (eq decimal-pos position)
                        (progn (setf decimal-pos nil)
                               (MY-ASSERT '*special-global-variables* 'decimal-pt nil) ) )
                     (if (not (= position 0) )
                        (progn (send win :increment-cursorpos (* -1 (send win :char-width) ) 0)
                               (send win :clear-char) (setf position (1- position) )
                               (setf ret-val (subseq ret-val 0 (1- (length ret-val) ) ) ) ) )
                     (if (= position 0) (MY-ASSERT '*special-global-variables* 'first-ch t) ) )
```

```
                (t  (if (= position 0)
                        (if (and sp-char (not (char= ch sp-char) ) )
                            (progn (WIPE-FIELD win width)
                                   (send win :delete-char)
                                   (send win :insert-string " ")
                                  (setf position (1+ position) ) )
                            (WIPE-FIELD win width) ) )
                    (send win :delete-char) (send win :insert-string (string ch) )
                    (setf change t) (setf position (1+ position) )
                    (if (char= ch #\.)  (setf decimal-pos position) )
                    (setf ret-val (string-append ret-val (string ch) ) ) ) )
                (beep) )

            (if (= position width)
                (return (values ret-val 'tab) ) ) ) ) ) )


;*****************************************************************************
(defun DISPLAY-VARIABLE (win len var-name)
;;; Display a previously input or calculated variable given either global or generic variable
;   name.  Generic names are assumed to require conversion to CE specific by adding the
;   *ce-num* to the end.

   (let* ( (var-name (if  (and  (string-equal (subseq (string var-name) 0 1) "*")
                                (string-equal (subseq (string var-name)
                                                      (- (string-length var-name) 1)
                                                      (string-length var-name) ) "*") )
                          var-name
                          (read-from-string (string-append var-name "-" *ce-num*) ) ) )
          (value (GET-DATA '*variable-data-alist* var-name) ) )

      (multiple-value-bind (start-x start-y) (send win :read-cursorpos)
         (if (and  (>= (length value) 1)
                   (not (or  (equalp (subseq value 0 1) "-")
                             (equalp (subseq value 0 1) "+") ) ) )
             (progn (send win :delete-char)
                    (send win :insert-string " ")
                    (princ  (if (> (length value) (1- len) )
                                (subseq value 0 (1- len) ) value)
                            win) )
             (princ (if (> (length value) len) (subseq value 0 len) value) win) )
         (NEXT-FIELD win len start-x start-y) ) ) )
```

```
;*****************************************************************************
(defun DISPLAY-XVAR (win len var-name)
;;; Display a previously input or calculated variable, using variable name exactly as given.

   (let* ( (value (GET-DATA '*variable-data-alist* var-name) ) )
      (multiple-value-bind (start-x start-y) (send win :read-cursorpos)
         (if (and (>= (length value) 1)
                  (not (or (equalp (subseq value 0 1) "-")
                           (equalp (subseq value 0 1) "+") ) ) )
            (progn (send win :delete-char)
                   (send win :insert-string " ")
                   (princ (if (> (length value) (1- len) )
                              (subseq value 0 (1- len) ) value)
                          win) )
            (princ (if (> (length value) len) (subseq value 0 len) value) win) )
         (NEXT-FIELD win len start-x start-y) ) ) )


;*****************************************************************************
(defun CREATE-MOD2-TABLE (global-variables user-variables table)
;;; Use the user- and global- variable tables from mod1 to build a screen description
;   table for mod2
   (setq *special-user-variables* user-variables)
   (setq *special-global-variables* global-variables)
   (let ( (coa-num 1)   (new-table table)
          (num-list (list 'zero 'one 'two 'three 'four 'five 'six 'seven 'eight 'nine 'ten) )
          (max-coa (VALOF '*variable-data-alist* 'max-coa) ) )

      (loop                                                          ;; Loop for COA's
         (let* ( (coa-val (string-append "COA-" (format nil "~s" coa-num) ) )
                 (coa-line
                   (string-append coa-val "  Main Attack Axis = "
                                  (VALOF '*variable-data-alist*
                                         (read-from-string
                                           (string-append "main-atk-axis-" coa-val) ) ) ) )
                 (av-blt-box-num 1) (inner-count nil) (mset-in-table nil)
                 (outer-count (read-from-string
                                (string-append "outer-" (nth coa-num num-list) ) ) )
                 (mset-out-table (list 'msets outer-count "") )  )

            (setq mset-out-table (append mset-out-table '( (left ,coa-line) ) ) )
            (loop                                                    ;; Loop for av/blt/box's
               (let* ( (av-blt-box-val
                         (read-from-string
                           (string-append coa-val "-" *moa-type* "-"
                                          (format nil "~s" av-blt-box-num) ) ) )
                       (av-blt-box-name
                         (if (VALOF '*variable-data-alist* av-blt-box-val)
                           (string-append *moa-type* "-"
                                          (VALOF '*variable-data-alist* av-blt-box-val) )
                           nil) )
```

```
                (ce-list (GET-VALUES '*special-user-variables* 'ce-number
                                     '*special-global-variables* coa-num av-blt-box-num) )
                (ce 1)  (row-count nil)  (mrow-only-table nil)  )

          (if (not av-blt-box-name)  (return) )          ;; If no name found, must be end
          (setq inner-count (read-from-string
                              (string-append "inner-" (nth av-blt-box-num num-list) ) ) )
          (setq mset-in-table (list 'msets inner-count "") )
          (setq mset-in-table (append mset-in-table
                                      '( (left ,(string-append "  " av-blt-box-name) ) ) ) )

          (dolist (ce-val ce-list)                              ;; All CE's this av-blt-box
            (let ( (ce-type (first (GET-VALUES '*special-user-variables* 'ce-type
                                               '*special-global-variables*
                                               coa-num av-blt-box-num ce) ) )
                   (objective (first (GET-VALUES '*special-user-variables* 'objective
                                                 '*special-global-variables*
                                                 coa-num av-blt-box-num ce) ) )  )
              (setq row-count (nth ce num-list) )
              (setq mrow-only-table (append (list 'mrow row-count "")
                                            (MAKE-MOD2-TABLE-ROW
                                              ce-val ce-type objective) ) )
              (setq mset-in-table (append mset-in-table (list mrow-only-table) ) )
              (setq ce (1+ ce) ) )  )                                ;; End CE list

          (setq row-count (nth ce num-list) )
          (setq mset-in-table (append mset-in-table
                                      (list (append (list 'mrow row-count "")
                                                    (MAKE-MOD2-TABLE-TOTAL
                                                      coa-num av-blt-box-num) ) ) ) )
          (setq mset-out-table (append mset-out-table (list mset-in-table) ) )
          (setq av-blt-box-num (1+ av-blt-box-num) ) )          ;; End av-blt-box loop

        (setq inner-count (read-from-string
                            (string-append "inner-" (nth av-blt-box-num num-list) ) ) )
        (setq mset-out-table
              (append mset-out-table
                      (list (append (list 'msets inner-count "")
                                    (MAKE-MOD2-TABLE-TOTAL coa-num) ) ) ) )
        (setq new-table (append new-table (list mset-out-table) ) )
        (setq coa-num (1+ coa-num) )
        (if (> coa-num max-coa)              ;; End COA loop, quit if next greater than max
          (return) ) ) )
    new-table ) )
```

```
;*****************************************************************************
(defun MAKE-MOD2-TABLE-ROW (ce ce-type objective)
;;; Build the CE row for the mod 2 screen description table
   (let ( (fc-pers (read-from-string (string-append "fc-pers-" ce) ) )
          (fc-equip (read-from-string (string-append "fc-equip-" ce) ) )
          (ec-pers (read-from-string (string-append "ec-pers-" ce) ) )
          (ec-equip (read-from-string (string-append "ec-equip-" ce) ) )
          (re-pol (read-from-string (string-append "re-pol-" ce) ) )
          (re-ammo (read-from-string (string-append "re-ammo-" ce) ) )
          (feba-mvmt (read-from-string (string-append "feba-mvmt-" ce) ) )
          (time-rqd (read-from-string (string-append "time-rqd-" ce) ) ) )

      '((ffield 6 " ")
        (ffield 8 ,ce)
        (ffield 21 ,ce-type)
        (ffield 20 ,objective)
        (dfield 6 "   0  " ,fc-pers)
        (ffield 1 " ")
        (dfield 6 "   0  " ,fc-equip)
        (ffield 1 " ")
        (dfield 6 "   0  " ,ec-pers)
        (ffield 1 " ")
        (dfield 6 "   0  " ,ec-equip)
        (ffield 1 " ")
        (dfield 6 "   0  " ,re-pol)
        (dfield 6 "   0  " ,re-ammo)
        (ffield 1 " ")
        (dfield 6 "   0  " ,feba-mvmt)
        (ffield 2 " ")
        (dfield 6 "  0.0" ,time-rqd)
        (newline 1) ) ) )


;*****************************************************************************
(defun MAKE-MOD2-TABLE-TOTAL (coa &optional av-blt-box)
;;; Build MOA and COA total rows for mod 2 screen description table
   (let* ( (ce-list
             (if av-blt-box
                (GET-VALUES '*special-user-variables* 'ce-number
                            '*special-global-variables* coa av-blt-box)
                (GET-VALUES '*special-user-variables* 'ce-number
                            '*special-global-variables* coa) ) )
           (variable-postfix
             (if av-blt-box
                (string-append (format nil "~s" coa) (format nil "~s" av-blt-box) )
                (format nil "~s" coa) ) )
           (axis-var (read-from-string (string-append "main-atk-axis-coa-"
                                                      (format nil "~s" coa) ) ) )
           (axis-val (if (VALOF '*variable-data-alist* axis-var)
                         (VALOF '*variable-data-alist* axis-var)
                         nil) )
           (fc-pers nil)  (fc-equip nil)  (ec-pers nil)  (ec-equip nil)
```

```
        (re-pol nil)  (re-ammo nil)  (feba-mvmt nil)  (time-rqd nil)
        (fc-pers-var (read-from-string (string-append "fc-pers-" variable-postfix) ) )
        (fc-equip-var (read-from-string (string-append "fc-equip-" variable-postfix) ) )
        (ec-pers-var (read-from-string (string-append "ec-pers-" variable-postfix) ) )
        (ec-equip-var (read-from-string (string-append "ec-equip-" variable-postfix) ) )
        (re-pol-var (read-from-string (string-append "re-pol-" variable-postfix) ) )
        (re-ammo-var (read-from-string (string-append "re-ammo-" variable-postfix) ) )
        (feba-mvmt-var (read-from-string (string-append "feba-mvmt-" variable-postfix) ) )
        (time-rqd-var (read-from-string (string-append "time-rqd-" variable-postfix) ) )
        (fc-pers-scale (read-from-string (string-append fc-pers-var "-scale") ) )
        (fc-equip-scale (read-from-string (string-append fc-equip-var "-scale") ) )
        (ec-pers-scale (read-from-string (string-append ec-pers-var "-scale") ) )
        (ec-equip-scale (read-from-string (string-append ec-equip-var "-scale") ) )
        (re-pol-scale (read-from-string (string-append re-pol-var "-scale") ) )
        (re-ammo-scale (read-from-string (string-append re-ammo-var "-scale") ) )
        (feba-mvmt-scale (read-from-string (string-append feba-mvmt-var "-scale") ) )
        (time-rqd-scale (read-from-string (string-append time-rqd-var "-scale") ) )
        (total  (if av-blt-box
                    (string-append *moa-type* " TOTAL")
                    "COA TOTAL ") ) )

    (dolist (ce ce-list)
      (setq fc-pers
            (append fc-pers (list (read-from-string (string-append "fc-pers-" ce) ) ) ) )
      (setq fc-equip
            (append fc-equip (list (read-from-string (string-append "fc-equip-" ce) ) ) ) )
      (setq ec-pers
            (append ec-pers (list (read-from-string (string-append "ec-pers-" ce) ) ) ) )
      (setq ec-equip
            (append ec-equip (list (read-from-string (string-append "ec-equip-" ce) ) ) ) )
      (setq re-pol
            (append re-pol (list (read-from-string (string-append "re-pol-" ce) ) ) ) )
      (setq re-ammo
            (append re-ammo (list (read-from-string (string-append "re-ammo-" ce) ) ) ) )
      ;; Calculate FEBA movement and time required only for events on the main attack axis.
      (let ( (axis (subseq (string ce) 4 5) ) )
        (if (string-equal axis axis-val)
            (progn (setq feba-mvmt
                         (append feba-mvmt (list (read-from-string
                                                   (string-append "feba-mvmt-" ce) ) ) ) )
                   (setq time-rqd
                         (append time-rqd (list (read-from-string
                                                  (string-append "time-rqd-" ce) ) ) ) ) ) ) ) )
    (if av-blt-box                                          ;; Return total row only
      '( (ffield 38 " ") (ffield 15 ,total) (ffield 2 " ")
         (demon 6 ,fc-pers-var INT-TOTAL ,@fc-pers) (ffield 1 " ")
         (demon 6 ,fc-equip-var INT-TOTAL ,@fc-equip) (ffield 1 " ")
         (demon 6 ,ec-pers-var INT-TOTAL ,@ec-pers) (ffield 1 " ")
         (demon 6 ,ec-equip-var INT-TOTAL ,@ec-equip) (ffield 1 " ")
         (demon 6 ,re-pol-var INT-TOTAL ,@re-pol)
         (demon 6 ,re-ammo-var INT-TOTAL ,@re-ammo) (ffield 1 " ")
         (demon 6 ,feba-mvmt-var INT-TOTAL ,@feba-mvmt) (ffield 2 " ")
         (demon 6 ,time-rqd-var TOTAL ,@time-rqd) (newline 1) )
```

```
        ; When doing COA return both total row and scaled value row
       '( (ffield 38 " ") (ffield 15 ,total) (ffield 2 " ")
          (demon 6 ,fc-pers-var INT-TOTAL ,@fc-pers) (ffield 1 " ")
          (demon 6 ,fc-equip-var INT-TOTAL ,@fc-equip) (ffield 1 " ")
          (demon 6 ,ec-pers-var INT-TOTAL ,@ec-pers) (ffield 1 " ")
          (demon 6 ,ec-equip-var INT-TOTAL ,@ec-equip) (ffield 1 " ")
          (demon 6 ,re-pol-var INT-TOTAL ,@re-pol)
          (demon 6 ,re-ammo-var INT-TOTAL ,@re-ammo) (ffield 1 " ")
          (demon 6 ,feba-mvmt-var INT-TOTAL ,@feba-mvmt) (ffield 2 " ")
          (demon 6 ,time-rqd-var TOTAL ,@time-rqd) (newline 1)

          (ffield 38 " ") (ffield 15 "SCALED VALUE") (ffield 4 " ")
          (demon 2 ,fc-pers-scale CALC-FC-PERS-SCALE-VAL ,fc-pers-var) (ffield 5 " ")
          (demon 2 ,fc-equip-scale CALC-FC-EQUIP-SCALE-VAL ,fc-equip-var) (ffield 5 " ")
          (demon 2 ,ec-pers-scale CALC-EC-PERS-SCALE-VAL ,ec-pers-var) (ffield 5 " ")
          (demon 2 ,ec-equip-scale CALC-EC-EQUIP-SCALE-VAL ,ec-equip-var) (ffield 5 " ")
          (demon 2 ,re-pol-scale CALC-RE-POL-SCALE-VAL ,re-pol-var) (ffield 4 " ")
          (demon 2 ,re-ammo-scale CALC-RE-AMMO-SCALE-VAL ,re-ammo-var) (ffield 5 " ")
          (demon 2 ,feba-mvmt-scale CALC-FEBA-MVMT-SCALE-VAL ,feba-mvmt-var)
          (ffield 7 " ")
          (demon 2 ,time-rqd-scale CALC-TIME-RQD-SCALE-VAL ,time-rqd-var)
          (newline 1) ) ) ) )


;*****************************************************************************
(defun DELETE-TRAILING-NEWLINES (table return-table &optional (found nil) )
;;; Delete newline indicators from the return-table prior to updating so that new lines
;   may be added.
   (let ( (pointer 0)  (new-table nil)
          (reverse-table (reverse return-table) )
          (hold-table nil) )

      (dolist (command (reverse table) )
         (if (atom command)
            (setq new-table (append new-table (list (nth pointer reverse-table) ) ) )
            (if (not found)
               (case (first command)
                  (MSETS
                     (let ( (temp-table nil) )
                        (dolist (ret-tab (reverse (nth pointer reverse-table) ) )
                           (multiple-value-setq
                              (hold-table found)
                              (DELETE-TRAILING-NEWLINES command ret-tab found) )
                           (setq temp-table (append temp-table (list hold-table) ) ) )
                        (setq new-table (append new-table (list (reverse temp-table) ) ) ) ) )
                  (MROW
                     (let ( (temp-table nil) )
                        (dolist (ret-tab (reverse (nth pointer reverse-table) ) )
                           (multiple-value-setq
                              (hold-table found)
                              (DELETE-TRAILING-NEWLINES command ret-tab found) )
                           (setq temp-table (append temp-table (list hold-table) ) ) )
                        (setq new-table (append new-table (list (reverse temp-table) ) ) ) ) )
```

```
                (NEWLINE
                   (setq new-table (append new-table (list nil) ) ) )
                (IFIELD
                   (setq new-table (append new-table (list (nth pointer reverse-table) ) ) )
                   (setq found t) )
                (SIFIELD
                   (setq new-table (append new-table
                                           (list (nth pointer reverse-table) ) ) )
                   (setq found t) )
                ((DEMON XDEMON WSDEMON)
                   (setq new-table (append new-table (list (nth pointer reverse-table) ) ) )
                   (setq found t) )
                (t  (setq new-table (append new-table (list (nth pointer reverse-table) ) ) ) ) )

             (setq new-table (append new-table (list (nth pointer reverse-table) ) ) ) )
          (setf pointer (1+ pointer) ) ) )
      (values (reverse new-table) found) ) )


;*****************************************************************************
(defun ND-TERPRI (win)
;;; Advance cursor a number of lines, if number is > 0, or to next line, and reset count to zero.
;   e.g., TFIELD records count of lines used for multiple line output, this function advances
;   cursor beyond those lines.
   (let ( (terpri-count (VALOF '*special-global-variables* (list 'terpri-count win) ) ) )
      (if (> terpri-count 0)
         (multiple-value-bind (curr-x curr-y) (send win :read-cursorpos)
            (setq curr-x 0)                          ; use x to avoid compiler warning
            (send win :set-cursorpos 0 (+ curr-y (* (1+ terpri-count) 14) ) ) )
         (terpri win) )
      (MY-ASSERT '*special-global-variables* (list 'terpri-count win) 0) ) )


;*****************************************************************************
(defun WIPE-FIELD (win width)
;;; Clear the display field of any previously displayed data.
   (multiple-value-bind (x y) (send win :read-cursorpos)
      (dotimes (i width)
         (send win :delete-char)
         (send win :insert-string " ") )
      (send win :set-cursorpos x y) ) )


;*****************************************************************************
(defun IS-ANY (ch)
;;; Test for any standard character.
   (standard-char-p ch) )
```

```
;*****************************************************************************
(defun IS-NUMBER (ch)
;;; Test for number.
   (and  (char>= ch #\0)
         (char<= ch #\9) ) )


;*****************************************************************************
(defun IS-SCALE-VAL (ch)
;;; Test for allowed scale values, 1-9 legal currently.
   (and  (char>= ch #\1)
         (char<= ch #\9) ) )


;*****************************************************************************
(defun IS-NATURAL-NUMBER (ch)
;;; Test for natural number.
   (let ( (first-ch (VALOF '*special-global-variables* 'first-ch) ) )
      (if first-ch (MY-ASSERT '*special-global-variables* 'first-ch nil) )
      (or (and first-ch (char= ch #\-) )
          (and first-ch (char= ch #\+) )
          (IS-NUMBER ch) ) ) )


;*****************************************************************************
(defun IS-DECIMAL (ch)
;;; Test for decimal number.
   (let ( (decimal-pt (VALOF '*special-global-variables* 'decimal-pt) ) )
      (if (and (not decimal-pt) (char= ch #\.) )
         (MY-ASSERT '*special-global-variables* 'decimal-pt t) )
      (or (and (char= ch #\.) (not decimal-pt) )
          (and (char>= ch #\0)
               (char<= ch #\9) ) ) ) )


;*****************************************************************************
(defun IS-IN (ch in-list)
;;; Test for character included in a specified list of aliowable characters.
   (member ch in-list) )


;*****************************************************************************
(defun IS-ALPHA (ch)
;;; Test for alphabetic character.
   (or (and (char>= ch #\a)
            (char<= ch #\z) )
       (and (char>= ch #\A)
            (char<= ch #\Z) ) ) )
```

```
;*****************************************************************************
(defun IS-ALPHANUMERIC (ch)
;;; Test for character either alphabetic or numeric.
   (or (IS-NUMBER ch) (IS-ALPHA ch) ) )


;*****************************************************************************
(defun DUPLICATE-CE-NUMBER (ce-num)
;;; Is this CE number already on the CE data list ?  Duplicate CE numbers will not be
;   allowed within a course of action.
   (let ( (answer nil) )
      (dolist (ce# *ce-number-list*)
         (if (string-equal ce# ce-num)
         (setq answer t) ) )
      answer) )


;*****************************************************************************
(defun GET-CE-DATA (ce-num)
;;; Given CE number, find the base data; type, objective, and comment for that CE.
   (let* ( (ce# (PARSE-BASE-CE-NUM ce-num) )
           (type (VALOF '*ce-data-alist* (read-from-string
                                          (string-append "ce-type-" ce#) ) ) )
           (obj (VALOF '*ce-data-alist* (read-from-string
                                          (string-append "objective-" ce#) ) ) )
           (com (VALOF '*ce-data-alist* (read-from-string
                                          (string-append "comment-" ce#) ) ) ) )
      (values ce-num type obj com) ) )


;*****************************************************************************
(defun ASK-END (win)
;;; Pop-up Menu at the end of a row or set for the user to choose the appropriate context
;   for the next row to be entered.  Menu list is the list of description arguments for MSETS
;   and MROW plus "Finished."
   (multiple-value-bind (cur-x cur-y) (send win :read-cursorpos)
      (let* ((response 0)  (count 1)  (mouse-x 20)  (mouse-y 25)
             (menu-x 150)  (menu-y (+ cur-y 125) )
             (label '(:string "  Start new:  "
                      :style (:swiss :bold :normal) ) )
             ; Reverse list of descriptions, add Finished to make list of choices
             (choice-list (append (reverse (VALOF '*special-global-variables* 'desc) )
                                  (list "FINISHED") ) )
             ; Put the first item with its value on the menu-list
             (menu-list (list (list (first choice-list) ':value 'tab) ) ) )
```

```
        (setq cur-x 0)                          ; use x to avoid compiler warning
        ;; Remove the first choice from the list, since already on the menu list.
        (setq choice-list (rest choice-list) )

        (dolist (choice choice-list)
           (setq menu-list (append menu-list (list (list choice  ':value  count) ) ) )
           (setq count (+ count 1) ) )

        (setq response (POP-UP-MENU-CHOICE
                         menu-list label menu-x menu-y mouse-x mouse-y t) ) ) ) )


;*****************************************************************************
(defun CHOOSE-CE-TYPE (win menu-list current-val len cursor-x cursor-y back-x back-y)
;;; Pop-up menu for the user to choose the type of CE from the menu list input as an
;   argument to MFIELD.  Add "Other" and "No Change" to the menu list.

      (let ( (choice nil)  (dir nil)  (default current-val)
           (test 'IS-ANY)            ; applies only for "other" option
           (menu-list (append menu-list
                              (list '("Other" :value "Other"
                                      :documentation " Enter your own type description") )
                              (list '("No Change" :value "No Change"
                                      :documentation " Type remains as previously input") ) ) )
           (label '(:string " Select Critical Event Type "
                    :style (:swiss :bold :normal) ) )
           (menu-x (+ cursor-x 700) )                ;; Put the menu over on the right side
           (menu-y (+ cursor-y 100) ) )

     ; If there is no previous value (arg default = nil) set the cursor in the middle.
     ; if there is a previous value set the cursor on "No Change".
      (if (not default)
         (let ( (num nil) )
            (if (integerp (/ (length menu-list) 2) )          ;; If list length is even number
               (setq num (/ (- (length menu-list) 2) 2) )     ;; then subtract 2 and get middle
               (setq num (/ (- (length menu-list) 1) 2) ) )   ;; else subtract 1 and get middle
            (setq default (nth num menu-list) ) )
         (setq default (first (last menu-list) ) ) )


         (loop
            ;; Put up the CE type menu and get the user choice
            (setq choice
                  (POP-UP-MENU-CHOICE menu-list label menu-x menu-y nil nil t default) )
            ;; If choice was No Change and no previous value exists, go back to menu
            (if (equalp choice "No Change")
               (if current-val
                  ;; Set choice to the previous input value and return out of the menu loop
                  (progn (setq choice current-val)
                         (return) ) )
               (return) ) )
```

```
        (if (equalp choice "")                                  ;; Choice "Other" value is ""
           ;; Choice was "Other", use insert-columns to process entry and go to next field
           (progn  (WIPE-FIELD win len)
                   (multiple-value-setq (choice dir)
                      (INSERT-COLUMNS win test len choice back-x back-y) ) )

           ;; For CE choices including "No Change" display and move to next field
           (progn  (WIPE-FIELD win len)
                   (princ choice win)
                   (if (equalp dir 'back)
                      (send win :set-cursorpos back-x back-y)              ;; backward
                      (send win :set-cursorpos                              ;; forward
                            (+ curr-x (* len (send win :char-width) ) )  curr-y) ) ) )

     ;; Record chosen CE-type by base CE number in the CE data list
     (MY-ASSERT  '*ce-data-alist*
                   (read-from-string
                      (string-append "ce-type-"
                                     (PARSE-BASE-CE-NUM *ce-num*) ) ) choice)

     (values choice dir) ) )


;*****************************************************************************
(defun CHOOSE-CE-NUMBER (win menu-list)
;;; Pop-up menu with a list of all CE's for the user to choose one for war-gaming.
   (multiple-value-bind (cursor-x cursor-y) (send win :read-cursorpos)
      (let ( (choice nil)  (n 0)
           (label '(:string " Select Critical Event "
                    :style (:swiss :bold :normal) ) )
           (menu-x (+ cursor-x 300) )
           (menu-y (+ cursor-y 100) ) )

        (setq choice (POP-UP-MENU-CHOICE
                        menu-list label menu-x menu-y '20 '30 t *next-ce-choice*) )

        ; Save the next item on the menu list, or this one if at the end, for next choice.
        (dolist (item menu-list)
           (setq n (+ n 1) )
           (when (equalp choice (first item) )
              (if (equal n (length menu-list) )
                 (setq *next-ce-choice* item)
                 (setq *next-ce-choice* (nth n menu-list) ) )
              (return) ) )
        choice) ) )
```

```
;*****************************************************************************
(defun ABSTRACT-DATA (x context context-set context-val)
;;; Get data for variable x in the given context.
   (if (equalp context (VALOF '*special-user-variables* x) )
      (VALOF '*special-user-variables*
             (if context-set
                (list context-set x context-val)
                (list x context-val) ) )
      (FETCH  x '*special-global-variables*
               '*special-user-variables*
             (let ( (cval (VALOF '*special-user-variables* x) ) )
                (if (and cval (listp cval) )
                   (append context-set  cval  (list context-val) )
                   context-set) ) ) ) )


;*****************************************************************************
(defun POP-CURSORPOS (table)
;;; Decrement the pointer for the cursor position table by one (1).
   (let ( (ptr (first table) )   (cursorpos-table (second table) ) )
      (if (= ptr -1)
         table
         (list (1- ptr) cursorpos-table) ) ) )


;*****************************************************************************
(defun PUSH-CURSORPOS (item table)
;;; Increment the pointer for the cursor position table by one (1).  If this cursor position
;   element is not in the table, add it.
   (let ( (ptr (first table) )   (cursorpos-table (second table) ) )
      (cond
         ( (= ptr (1- (length cursorpos-table) ) )
            (list (1+ ptr) (append cursorpos-table (list item) ) ) )
         (t (list (1+ ptr) cursorpos-table) ) ) ) )


;*****************************************************************************
(defun DISPLAY-ITEM-NUMBER (win type variable context)
;;; Display an alphabetic or numeric character in sequence within the current context.
;   Value is displayed, stored in *special-user-variables*, and returned.
   (if type
      (let ( (position (VALOF '*special-global-variables* (append context '(iter) ) ) ) )
         (case type
            (numeric
               (send win :string-out (format nil "~s" position) )
               (PUT-IN-SYMBOL-TABLE variable context position)
               (format nil "~s" position) )
            (alpha
               (send win :string-out (format nil "~s" (ALPHA-SEQUENCE position) ) )
               (PUT-IN-SYMBOL-TABLE variable context (ALPHA-SEQUENCE position) )
               (format nil "~s" (ALPHA-SEQUENCE position) ) ) ) )
      nil) )
```

```
;*****************************************************************************
(defun FETCH (key context-loc key-loc &optional in-context-set)
;;; Search the symbol table, key-loc, for the variable, key, within the current context,
;   obtained from context-loc.  Returns the value of key.

   (let* ( (context (VALOF key-loc key) )   (context-set nil)   (current-context context) )

      (if (not (listp context) )
         context
         (progn
            (if context
               (loop
                  (let ( (next-context (VALOF context-loc
                                              (append current-context '(context) ) ) ) )
                     (if (not next-context)
                        (return nil) )
                     (setf context-set (append context-set
                                               (append next-context
                                                       (list
                                                          (if in-context-set
                                                             (FIND-ITER next-context
                                                                        in-context-set)
                                                             (VALOF context-loc
                                                                    (append next-context
                                                                            '(iter) ) ) ) ) ) ) )
                     (setf current-context next-context) ) ) )
            (let* ( (context-val (if in-context-set
                                     (FIND-ITER context in-context-set)
                                     (VALOF context-loc (append context '(iter) ) ) ) )
                    (value (VALOF key-loc (if context-set
                                              (list context-set key context-val)
                                              (list key context-val) ) ) ) )
               value) ) ) ) )


;*****************************************************************************
(defun FIND-ITER (context context-set)
;;; Get the iteration number of the current context within its containing context.
   (let ( (working-set context-set) )
      (loop
         (setf working-set (member (first context) working-set :test #'equal) )
         (if (equalp (second working-set) (second context) )
            (return (third working-set) ) )
         (if (not working-set)
            (return nil) )
         (setf working-set (rest (rest (rest working-set) ) ) ) ) ) )
```

```
;*****************************************************************************
(defun PUT-IN-SYMBOL-TABLE (variable context value)
;;; Store value of variable in the symbol table within the given context.
   (if variable
      (if context
         (let ( (context-set nil)   (current-context context) )
            (loop
               (let ( (next-context (VALOF '*special-global-variables*
                                           (append current-context '(context) ) ) ) )
                  (if (not next-context)
                     (return nil) )
                  (setf context-set (append context-set
                                            (append next-context
                                                    (list
                                                      (VALOF '*special-global-variables*
                                                             (append next-context
                                                                     '(iter) ) ) ) ) ) )
                  (setf current-context next-context) ) )
            (MY-ASSERT '*special-user-variables* variable context)
            (MY-ASSERT '*special-user-variables*
                       (if context-set
                          (list context-set
                                variable
                                (VALOF '*special-global-variables*
                                       (append context '(iter) ) ) )
                          (list variable
                                (VALOF '*special-global-variables*
                                       (append context '(iter) ) ) ) )
                       value) )
         (MY-ASSERT '*special-user-variables* variable value) ) ) )


;*****************************************************************************
(defun TOGGLE (flag)
;;; Toggle a flag between nil and T.
   (if flag nil t) )


;*****************************************************************************
(defun RETRIEVE-FROM-TABLE (pointer table)
;;; Get the value at a given pointer location in the return-table.
   (let ( (ret-val table) )
      (dolist (i pointer)
         (if (not (listp ret-val) )
            (progn
               (setf ret-val nil)
               (return nil) )
            (setf ret-val (nth i ret-val) ) ) )
      ret-val) )
```

```
;*****************************************************************************
(defun ITEM-EXISTS (pointer table)
;;; Determine if a non-nil value exists at a given pointer location in the return table.
;   Actually only provides a different name for RETRIEVE-FROM-TABLE to increase readablilty
;   of the code.
   (RETRIEVE-FROM-TABLE pointer table) )


;*****************************************************************************
(defun INSERT-AT-POINTER (pointer table item)
;;; Store an item at a given pointer location in a table.
   (let ( (pos 0)  (val table)  (prev-val nil) )
      (dolist (i pointer)
         (setf prev-val val)
         (setf val (nth i val) )
         (if (not val)
            (return nil) )
         (setf pos (1+ (ZERO-IF-NIL pos) ) ) )

      (let* ( (insert-pointer (subseq pointer 0  pos) )
              (insert-item (RETRIEVE-FROM-TABLE insert-pointer table) )
              (increase-pos (nth pos pointer) )
              (create-pointer (if (> (1+ (ZERO-IF-NIL pos) ) (length pointer) )
                                  nil
                                  (subseq pointer (1+ (ZERO-IF-NIL pos) ) ) ) ) )
         (if (atom insert-item)
            (let* ( (new-insert-item (RETRIEVE-FROM-TABLE (butlast insert-pointer) table) )
                    (new-increase-item
                         (SUBSTITUTE-POS  (first (last insert-pointer) )
                                          (INCREASE-ITEM increase-pos
                                                         prev-val
                                                         (CREATE-ITEM create-pointer
                                                                      item) )
                                          new-insert-item) ) )
               (subst new-increase-item  new-insert-item  table) )
            (subst (INCREASE-ITEM increase-pos
                                  prev-val  (CREATE-ITEM create-pointer item) )
                   insert-item  table) ) ) )


;*****************************************************************************
(defun ZERO-IF-NIL (item)
;;; Convert a nil item to zero (0).
   (if (not item)
      0
      item) )
```

```
;*****************************************************************************
(defun CREATE-ITEM (pointer item)
;;; Make a list item for use by INSERT-AT-POINTER.
   (if (not pointer)
      item
      (let* ( (item-loc (first (last pointer) ) )
              (item-struct (append (make-list item-loc) (list item) ) ) )
         (dolist (i (reverse (butlast pointer) ) )
            (setf item-struct (append (make-list i) (list item-struct) ) ) )
         item-struct) ) )


;*****************************************************************************
(defun INCREASE-ITEM (pos lis item)
;;; Add item to a list (lis) at position (pos).
   (if (not pos)
      item
      (if (= (1+ pos) (length lis) )
         (SUBSTITUTE-POS pos item lis)
         (append lis (make-list (- pos (length lis) ) ) (list item) ) ) ) )



;*****************************************************************************
(defun INCREMENT-POINTER (pointer i)
;;; Increment the ith element of the pointer list
   (SUBSTITUTE-POS i (1+ (ZERO-IF-NIL (nth i pointer) ) ) pointer) )



;*****************************************************************************
(defun DECREMENT-POINTER (pointer i)
;;; Decrement the ith element of the pointer list
   (if (= (nth i pointer) 0)
      pointer
      (SUBSTITUTE-POS i (1- (ZERO-IF-NIL (nth i pointer) ) ) pointer) ) )



;*****************************************************************************
(defun SUBSTITUTE-POS (position item sequence)
;;; Replace an item at a given position in a sequence.
   (if (and (not sequence)
            (not position) )
      item
      (substitute-if item #'(lambda (x) t) sequence :start (ZERO-IF-NIL position)
                                                    :end (1+ (ZERO-IF-NIL position) ) ) ) )
```

```
;*****************************************************************************
(defun GET-DATA (alist variable &optional default)
;;; Get value from an association list given the alist name and variable name.
;   If no value exists then return blank or the given (optional) default value.
   (if (VALOF alist variable)
      (VALOF alist variable)
      (if default
         default
         " ") ) )


;*****************************************************************************
(defun VALOF (obj attr)
;;; Find the first association pair in the list "obj" which has a key equal to "attr."
   (second (assoc attr (eval obj) :test #'equal) ) )


;*****************************************************************************
(defun MY-ASSERT (o a v)
;;; Set the attribute, a, of object, o, to be value, v, for an association list.
   (cond
      ( (eq v (second (assoc a (eval o) :test #'equal) ) )  nil)
      ( (assoc a (eval o) :test #'equal)  (setf (second (assoc a (eval o) :test #'equal) ) v) )
      (t (set o (cons (list a v) (eval o) ) ) ) ) )


;*****************************************************************************
(defun ASSERT-AT-END (o a v)
;;; Append association pair; attribute a and value v, to the list object o.
   (cond ( (assoc a (eval o) :test #'equal)
            (setf (second (assoc a (eval o) :test #'equal) )
            (append (second (assoc a (eval o) :test #'equal) ) (list v) ) ) )
         (t  (set o (cons (list a (list v) ) (eval o) ) ) ) ) )


;*****************************************************************************
(defun ALPHA-SEQUENCE (position)
;;; Return an alphabetic character determined by the given position indicator.
   (multiple-value-bind (div zl:rem) (truncate position 26)
      (let ( (letter (nth  (if (= zl:rem 0)
                               25
                               (1- zl:rem) )
                           '(#\A #\B #\C #\D #\E #\F #\G #\H #\I #\J #\K #\L #\M
                             #\N #\O #\P #\Q #\R #\S #\T #\U #\V #\W #\X #\Y #\Z) ) ) )
         (read-from-string (string (make-string
                                     (if (= zl:rem 0)
                                        div
                                        (1+ div) ) :initial-element letter) ) ) ) ) )
```

```
;*****************************************************************************
(defun SUB-CONTEXT-P (context1 context2 context-loc)
;;; Determine if one context, context1, is contained in another, context2.
   (if (not context2)
      t
      (let* ( (current-context context1) )
         (loop
            (let ( (next-context (VALOF context-loc (append current-context '(context) ) ) ) )
               (if (equalp current-context context2)
                  (return t) )
               (if (not next-context)
                  (return nil) )
               (setf current-context next-context) ) ) ) ) )


;*****************************************************************************
(defun ALL-VALUES (obj attr &optional context-loc)
;;; Find all values of the attribute "attr" in the list "obj".
;   If context-loc is given, then return all values in the current context.
   (reverse
      (if context-loc
         (let* ( (context (VALOF obj attr) )  (context-set nil)  (current-context context) )
            (if context
               (loop
                  (let ( (next-context (VALOF context-loc
                                               (append current-context '(context) ) ) ) )
                     (if (not next-context)
                        (return nil) )
                     (setf context-set
                           (append context-set
                                   (append next-context
                                           (list (VALOF  context-loc
                                                         (append next-context
                                                                 '(iter) ) ) ) ) ) )
                     (setf current-context next-context) ) ) )
            (remove-if #'(lambda (x) (not x) )
                       (mapcar #'(lambda (x) (if (or  (and  (equal context-set nil)
                                                            (listp (first x) )
                                                            (equal (first (first x) ) attr)
                                                            (<= (second (first x) )
                                                                (VALOF context-loc
                                                                       (append context
                                                                               '(iter) ) ) ) )
                                                      (and  (listp x)
                                                            (listp (first x) )
                                                            (equal (first (first x) ) context-set)
                                                            (eq (second (first x) ) attr)
                                                            (<= (third (first x) )
                                                                (VALOF context-loc
                                                                       (append context
                                                                               '(iter) ) ) ) ) )
                                                 (second x) ) )
                               (eval obj) ) ) )
```

```
        (remove-if #'(lambda (x) (not x) )                                    ;; else
                   (mapcar #'(lambda (x) (if (or (and (listp (first x) )
                                                      (equal (first (first x) ) attr) )
                                                 (and (listp x)
                                                      (listp (first x) )
                                                      (eq (second (first x) ) attr) ) )
                                             (second x) ) )
                           (eval obj) ) ) ) ) )


;*****************************************************************************
(defun GET-VALUES (obj attr context-loc &rest number)
;;; Get the value of the attribute, attr, in the list, obj, in the current context.
   (reverse
      (let* ((context (VALOF obj attr) )  (context-set nil)  (current-context context)  (count 0) )
         (if context
            (loop
               (let ( (next-context (VALOF context-loc (append current-context '(context) ) ) ) )
                  (if (not next-context)
                     (return nil) )
                  (setf context-set (append context-set (append next-context '(0) ) ) )
                  (setf current-context next-context)
                  (setf count (1+ count) ) ) ) )
         (dolist (i number)
            (setq context-set (reverse (substitute i 0  (reverse context-set)
                                                   :count 1 :test #'zl:equal) ) ) )
         (loop
            (let ( (new-context-set (member 0 context-set) ) )
               (if new-context-set
                  (setf context-set (rest new-context-set) )
                  (return nil) ) ) )
         (remove-if #'(lambda (x) (not x) )
                    (mapcar #'(lambda (x) (if (and  (listp x)
                                                    (listp (first x) )
                                                    (or (and (equal context-set nil)
                                                             (eq (first (first x) ) attr)
                                                             (if (nth count number)
                                                                ( = (second (first x) )
                                                                   (nth count number) )
                                                                t) )
                                                        (and  (CONTEXT-EQUAL (first (first x) )
                                                                             context-set)
                                                              (eq (second (first x) ) attr)
                                                              (if (nth count number)
                                                                 ( = (third (first x) )
                                                                    (nth count number) )
                                                                 t) ) ) )
                                                  (second x) ) )
                            (eval obj) ) ) ) ) )
```

```
;*****************************************************************************
(defun CONTEXT-EQUAL (x context-set)
;;; Compare two contexts to determine if they are the same.
   (if (listp x)
      (let ( (x-list (if (equal (length x) (length context-set) )
                         x
                         (REVERSE-BY-THREE x) ) )
             (return-val t) )
         (if (listp x-list)
            (progn
               (dolist (item context-set)
                  (if (not (equal item (first x-list) ) )
                     (progn
                        (setq return-val nil)
                        (return nil) ) )
                  (setf x-list (rest x-list) ) )
               return-val)
            (equal x context-set) ) )
      (equal x context-set) ) )


;*****************************************************************************
(defun REVERSE-BY-THREE (lis)
;;; Reverse a list in groups of three elements.  The three elements of each
;   group are not reversed.
   (let ( (ret-val (reverse lis) ) )
      (dotimes (i (truncate (length lis) 3) )
         (let ( (j (* i 3) ) )
            (setf ret-val
                  (append (subseq ret-val 0 j)
                          (reverse (subseq ret-val j (+ j 3) ) )
                          (subseq ret-val (+ j 3) ) ) ) ) )
      ret-val) )


;*****************************************************************************
(defun ELIMINATE-COMPILER-WARNINGS-FOR-ARGS-NOT-USED
                          (&optional arg1 arg2 arg3 arg4 arg5)
;;; This function does nothing. It is used to eliminate compiler warnings for standard
;   arguments which are not used, e.g., COMMAND-EXECUTE applies all screen description
;   commands with a standard set of arguments, not all of these are always used.
;   Call this function to use them.

   (let ((test nil))
      (when test
         (ELIMINATE-COMPILER-WARNINGS-FOR-ARGS-NOT-USED
                                          arg1 arg2 arg3 arg4 arg5) ) ) )
```

APPENDIX F

FILE COAAT-DEMONS.LISP

APPENDIX F - FILE COAAT-DEMONS.LISP

CONTENTS

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Package: USER; Base: 10 -*-
;*********************************
;*      COAAT-DEMONS.LISP        *
;*********************************
; Functions which perform calculations using the data input to COAAT.

;*****************************************************************************
(defun DEMON ( win pointer-t pointer-r dir table cursorpos-table ret-val redo context
               len var function &rest args)
;;; Screen description command function to calculate and display a field calculated from
;   other input data.
   ;; If back-up or redo, skip; let UPDATE-DEMON take care of it when data is changed
   (if (and (not (equalp dir 'back) ) (not redo) )
      (progn
         (ELIMINATE-COMPILER-WARNINGS-FOR-ARGS-NOT-USED pointer-t table)
         ;; Convert variable names (unless global) to CE specific names
         (multiple-value-setq (var args)
            (apply 'DEMON-VARS (append (list var) args) ) )
         (multiple-value-bind (curr-x curr-y) (send win :read-cursorpos)
            ;; Get context information
            (let* ((context-set nil) (current-context context)
                   (context-val (if context
                                    (VALOF '*special-global-variables*
                                           (append context '(iter) ) ) ) )
                   (postfix (if  context-val
                                 (list var context-val)
                                 (list var) ) ) )
               (if context
                  (loop
                        (let ( (next-context (VALOF '*special-global-variables*
                                                    (append current-context '(context) ) ) ) )

                        (if (not next-context) (return nil) )
                        (setf context-set
                              (append context-set
                                      (append next-context
                                              (list (VALOF '*special-global-variables*
                                                           (append next-context '(iter) ) ) ) ) ) )
                        (setf current-context next-context) ) ) )
               (if (not (VALOF '*demons* (if  context-set
                                              (append (list  context-set) postfix '(exists) )
                                              (append postfix '(exists) ) ) ) )
                  (let ( (prefix (if context-set                                    ;; then
                                     (append (list context-set) postfix)
                                     postfix) ) )
                     ;; Record all information in the *demon* table for future reference
                     (if (not (VALOF '*demons* (append postfix '(exists) ) ) )
                        (progn
                           (if (not (member var (VALOF '*demons* '*demons*) :test #'equal) )
                              (MY-ASSERT  '*demons* '*demons*
                                          (append (VALOF '*demons* '*demons*) (list var) ) ) )
                           (MY-ASSERT '*demons* (append postfix '(exists) ) t)
```

```
                        (MY-ASSERT '*demons* (list var 'context) context)
                        (MY-ASSERT '*demons* (list var 'function) function)
                        (MY-ASSERT '*demons* (list var 'args) args)
                        (MY-ASSERT '*demons* (list var 'length) len) ) )
                    (MY-ASSERT '*demons* (append prefix '(exists) ) t)
                    (MY-ASSERT '*demons* (append prefix '(loc) ) (list curr-x curr-y) )
                    (MY-ASSERT '*demons* (append prefix '(pointer) ) pointer-r)
                    ;; Apply the specified calculation function
                    (let ( (value (apply  function
                                          (append (list context context-set context-val)
                                                  args) ) ) )
                      ;; Store result value in the *demons* table
                      (MY-ASSERT '*demons* (append prefix '(value) ) value)
                      ;; Store value in the return table
                      (setf  ret-val
                             (INSERT-AT-POINTER pointer-r ret-val value) )
                      ;; Store value in the *variable-data-alist*, store 0 rather than blanks
                      (if (or (equalp value "") (equalp value " ") )
                        (setq value (format nil "~5F" 0) ) )
                      (MY-ASSERT '*variable-data-alist* var value)
                      (MY-ASSERT '*pointer-variable-alist* pointer-r var) ) ) )
                ;; Display the result and advance cursor to next field
                (DISPLAY-DEMON var context-set context-val win)
                (send win :set-cursorpos (+ curr-x (* len (send win :char-width) ) ) curr-y) ) ) ) )
    (values ret-val cursorpos-table dir) )


;*****************************************************************************
(defun WSDEMON (win pointer-t pointer-r dir table cursorpos-table ret-val redo context
                    len var function &rest args)
;;; Screen description command function to calculate and display a field calculated from
;   other input data.  Special case of DEMON function for wargaming worksheet, does not
;   require context info.
    ;; If back-up or redo, skip; let UPDATE-DEMON take care of it when data is changed
    (if (and (not (equalp dir 'back) ) (not redo) )
        (progn
          (ELIMINATE-COMPILER-WARNINGS-FOR-ARGS-NOT-USED pointer-t table context)
          (multiple-value-setq (var args)
            ;; Convert variable names (unless global) to CE specific names
            (apply 'DEMON-VARS (append (list var) args) ) )
          (multiple-value-bind (curr-x curr-y) (send win :read-cursorpos)
            ;; If it doesn't exist already, then record all info and calculate.
            (if (not (VALOF '*demons* (append (list var) '(exists) ) ) )
                (progn
                  (if (not (member var (VALOF '*demons* '*demons*) :test #'equal) )
                      (MY-ASSERT  '*demons* '*demons*
                                  (append (VALOF '*demons* '*demons*) (list var) ) ) )
                  (MY-ASSERT '*demons* (append (list var) '(exists) ) t)
                  (MY-ASSERT '*demons* (list var 'function) function)
                  (MY-ASSERT '*demons* (list var 'args) args)
                  (MY-ASSERT '*demons* (list var 'length) len)
                  (MY-ASSERT '*demons* (append (list var) '(loc) ) (list curr-x curr-y) )
                  (MY-ASSERT '*demons* (append (list var) '(pointer) ) pointer-r)
```

```
                ;; Apply the specified calculation function
                (let ( (value (apply  function
                                      (append (list 'nil 'nil 'nil) args) ) ) )
                   ;; Store result value in the *demons* table
                   (MY-ASSERT '*demons* (append (list var) '(value) ) value)
                   ;; Store value in the return table
                   (setf  ret-val
                          (INSERT-AT-POINTER pointer-r ret-val value) )
                   ;; Store value in the *variable-data-alist*, store 0 rather than blanks
                   (if  (or (equalp value "") (equalp value " ") )
                      (setq value (format nil "~5F" 0) ) )
                   (MY-ASSERT '*variable-data-alist* var value)
                   (MY-ASSERT '*pointer-variable-alist* pointer-r var) ) ) )
             ;; Display the result value
             (DISPLAY-DEMON var 'nil 'nil win)
             (send win :set-cursorpos (+ curr-x (* len (send win :char-width) ) ) curr-y) ) ) )
      (values ret-val cursorpos-table dir) )


;*****************************************************************************
(defun XDEMON (win pointer-t pointer-r dir table cursorpos-table ret-val redo context
                   len var function &rest args)
;;; Screen description command function to calculate and display a field calculated from
;   other input data.  Special case of DEMON function, eXact DEMON, variable names are
;   NOT changed to CE specific they are used exactly as input
   (if (and (not redo) (not (equalp dir 'back) ) )
      (multiple-value-bind (curr-x curr-y) (send win :read-cursorpos)
         (ELIMINATE-COMPILER-WARNINGS-FOR-ARGS-NOT-USED pointer-t table)
         (let* ( (context-set nil) (current-context context)
                 (context-val (if context
                                  (VALOF '*special-global-variables*
                                         (append context '(iter) ) ) ) )
                 (postfix (if  context-val
                               (list var context-val)
                               (list var) ) ) )
            (if context
               (loop
                  (let ( (next-context (VALOF  '*special-global-variables*
                                               (append current-context '(context) ) ) ) )
                     (if (not next-context) (return nil) )
                     (setf  context-set
                            (append context-set
                                    (append next-context
                                            (list (VALOF '*special-global-variables*
                                                         (append next-context '(iter) ) ) ) ) ) )
                     (setf current-context next-context) ) ) )
            (if (not (VALOF '*demons* (if context-set
                                          (append (list  context-set) postfix '(exists) )
                                          (append postfix '(exists) ) ) ) )
               (let ( (prefix (if context-set
                                  (append (list context-set) postfix)
                                  postfix) ) )
```

```
                ;; Record demon data for future reference
                (if (not (VALOF '*demons* (append postfix '(exists) ) ) )
                   (progn
                      (MY-ASSERT '*demons* (append postfix '(exists) ) t)
                      (MY-ASSERT '*demons* (list var 'context) context)
                      (MY-ASSERT '*demons* (list var 'function) function)
                      (MY-ASSERT '*demons* (list var 'args) args)
                      (MY-ASSERT '*demons* (list var 'length) len)
                      (if (not (member var (VALOF '*demons* '*demons*) :test #'equal) )
                         (MY-ASSERT  '*demons* '*demons*
                                     (append  (VALOF '*demons* '*demons*)
                                              (list var) ) ) ) ) )
                (MY-ASSERT '*demons* (append prefix '(exists) ) t)
                (MY-ASSERT '*demons* (append prefix '(loc) ) (list curr-x curr-y) )
                (MY-ASSERT '*demons* (append prefix '(pointer) ) pointer-r)
                (let ( (value (apply  function
                                      (append (list context context-set context-val)
                                              args) ) ) )
                   ;; Store value in the *demons* table
                   (MY-ASSERT '*demons* (append prefix '(value) ) value)
                   ;; Store value in the return table
                   (setf  ret-val
                          (INSERT-AT-POINTER pointer-r ret-val value) )
                   ;; Store value in the *variable-data-alist*, store 0 rather than blanks
                   (if (or (equalp value "") (equalp value " ") )
                      (setq value (format nil "~5F" 0) ) )
                   (MY-ASSERT '*pointer-variable-alist* pointer-r var)
                   (MY-ASSERT '*variable-data-alist* var value) ) ) )
             ;; Display the result value
             (DISPLAY-DEMON var context-set context-val win)
             (send win :set-cursorpos (+ curr-x (* len (send win :char-width) ) ) curr-y) ) ) )
      (values ret-val cursorpos-table dir) )


;*****************************************************************************
(defun UPDATE-DEMONS (win ret-val variable)
;;; Function to update the displayed demon result value when the data for that demon is
;   changed.
   ;; Make a list of all the demon args lists in the *demons* table
   (let* ( (current-demons (remove-if-not
                           #'(lambda (x) (and (listp x)
                                             (listp (first x) )
                                             (equalp (second (first x) ) 'args) ) )
                           *demons*) )
          ; Build a list of those demons, and their arg lists, which include this variable
          (applicable-demons (remove-if-not
                              #'(lambda (x) (and (listp (second x) )
                                                 (ELEMENT-MATCH variable (second x) ) ) )
                              current-demons) ) )
```

```
      ;; Now update each demon containing this variable, record and display new value.
      (dolist (update-var applicable-demons)
         (let* ( (var (first (first update-var) ) )
                 (function (VALOF '*demons* (list var 'function) ) )
                 (args (second update-var) )
                 (value (VALOF '*demons* (append (list var) '(value) ) ) )
                 (pointer-d (VALOF '*demons* (append (list var) '(pointer) ) ) )
                 (new-val (apply function (append (list 'nil 'nil 'nil) args) ) ) )

            (if (not (equalp new-val value) )                    ;; If value not changed, forget it
               (progn
                  (if (or (equalp new-val "") (equalp new-val " ") )
                     (setq new-val (format nil "~5F" 0) ) )
                  (MY-ASSERT '*demons* (append (list var) '(value) ) new-val)
                  (setf ret-val (INSERT-AT-POINTER pointer-d ret-val new-val) )
                  (MY-ASSERT '*variable-data-alist* var new-val)
                  (DISPLAY-DEMON var 'nil 'nil win t) ) ) ) ) )
   ret-val)


;*****************************************************************************
(defun ELEMENT-MATCH (variable list)
;;; Does the variable match any item on the list ? If so return T, else nil.
   (dolist (element list)
      (if (equalp element variable)
         (return t) ) ) )


;*****************************************************************************
(defun DISPLAY-DEMON (var context-set context-val win &optional store-position)
   (let* ((context (VALOF '*demons* (list var 'context) ) )
          (loc (VALOF '*demons* (if context
                                    (if context-set
                                       (list context-set var context-val 'loc)
                                       (list var context-val 'loc) )
                                    (list var 'loc) ) ) )
          (value (VALOF '*demons* (if context
                                      (if context-set
                                         (list context-set var context-val 'value)
                                         (list var context-val 'value) )
                                      (list var 'value) ) ) )
          (len (VALOF '*demons* (list var 'length) ) ) )
      (if store-position
         ;; Store current cursor position, do demon and return to the variable field
         (multiple-value-bind (old-x old-y) (send win :read-cursorpos)
            (send win :set-cursorpos (first loc) (second loc) )    ;; Location of the demon field
            (WIPE-FIELD win len)                                   ;; Clear anything that might be in the field
```

```
          (if (and (not (equalp value "") )
                   (not (or  (equalp (subseq value 0 1) "-")
                             (equalp (subseq value 0 1) "+") ) ) )
              (progn
                 (send win :delete-char 1)
                 (send win :insert-string " ")
                 (princ (if (> (length value) (1- len) )
                            (subseq value 0 (1- len) ) value) win) )
              (princ (if (> (length value) len) (subseq value 0 len) value) win) )
          (send win :set-cursorpos old-x old-y) )

       ;; When store position flag is nil, just put it at current cursor position
       (progn
          (WIPE-FIELD win len)                    ;; Clear anything that might be in the field
          (if (and (>= (length value) 1)
                   (not (or  (equalp (subseq value 0 1) "-")
                             (equalp (subseq value 0 1) "+") ) ) )
              (progn
                 (send win :delete-char 1)
                 (send win :insert-string " ")
                 (princ (if (> (length value) (1- len) )
                            (subseq value 0 (1- len) ) value) win) )
              (princ (if (> (length value) len) (subseq value 0 len) value) win) ) ) ) ) )


;*****************************************************************************
(defun INIT-DEMON (win len var function &rest args)
;;; Short, quick version of XDEMON for initializing screen.  Eliminated all context info
;   and reference to return-table.  Calculates value, stores it in *variable-data-alist*,
;   no other data is recorded.  None of the variable names are changed by this function.

   ;; Apply the appropriate demon function
   (let ( (value (apply function (append (list 'nil 'nil 'nil) args) ) ) )
      ;; Change blanks to 0.0
      (if (or (equalp value "") (equalp value " ") )
          (setq value (format nil "~5F" 0) ) )
      ;; Store value in the *variable-data-alist*
      (MY-ASSERT '*variable-data-alist* var value) )

   ;; Get the value from *variable-data-alist* and display , then move to next field
   (DISPLAY-XVAR win len var) )
```

```
;*****************************************************************************
(defun INIT-WSDEMON (win len var function &rest args)
;;; Short, quick version of WSDEMON for initializing screen.  Eliminated all context info
;   and reference to return-table.  Calculates value, stores it in *variable-data-alist*,
;   no other data is recorded.  Uses DEMON-VARS to convert names to CE specific names.

   (multiple-value-setq (var args) (apply 'DEMON-VARS (append (list var) args) ) )

   ;; Apply the appropriate demon function
   (let ( (value (apply function (append (list 'nil 'nil 'nil) args) ) ) )
      ;; Change blanks to 0.0
      (if (or (equalp value "") (equalp value " ") )
         (setq value (format nil "~5F" 0) ) )
      ;; Store value in the *variable-data-alist*
      (MY-ASSERT '*variable-data-alist* var value) )

   ;; Get the value from *variable-data-alist* and display , then move to next field
   (DISPLAY-XVAR win len var) )


;*****************************************************************************
(defun DEMON-VARS (var &rest args)
;;; Changes the names of DEMON input variable and its argument variables to include
;   the CE number (CE specific variable names) so data can be stored into and retrieved
;   from the *variable-data-alist*. The variable being calculated is assumed to always
;   be CE specific, the argument variables may be either CE specific or global.
;   Global variables are noted by leading and trailing asterisks (*xyz*) and their names
;   will not be changed.
   (let ( (var (read-from-string (string-append var "-" *ce-num*) ) )
          (arg-var nil) (new-args nil)  )
      (loop
         (setq arg-var (first args) )
         (setq args (rest args) )
         (setq arg-var  (if (and  (string-equal (subseq (string arg-var) 0 1) "*")
                                  (string-equal (subseq (string arg-var)
                                                        (- (string-length arg-var) 1)
                                                        (string-length arg-var) ) "*") )
                            arg-var                          ;; Global variables, no change
                            (read-from-string (string-append arg-var "-" *ce-num*) ) ) )
         (setq new-args (append new-args (list arg-var) ) )
         (if (not args)
            (return) ) )
      (values var new-args) )  )
```

```
;*******************************************************************************
(defun WEIGHTED-VALUE (dum1 dum2 dum3 &rest val-weight)
;;; Get the total (sum) weighted value of a set of values and weights
   (ELIMINATE-COMPILER-WARNINGS-FOR-ARGS-NOT-USED dum1 dum2 dum3)
   (let ( (val nil)
          (weight nil)
          (val*weight nil) )
      (loop
         (if (not val-weight)
            (return nil) )
         (if (VALOF '*variable-data-alist* (first val-weight) )
            (setq val (append val (list (VALOF '*variable-data-alist*
                                               (first val-weight) ) ) ) )
            "0")
         (setq val-weight (rest val-weight) )
         (if (VALOF '*variable-data-alist* (first val-weight) )
            (setq weight (append weight (list (VALOF  '*variable-data-alist*
                                                      (first val-weight) ) ) ) )
            "0")
         (setq val-weight (rest val-weight) ) )
      ;; if any member of either list is a blank then return blank
      (if (or (member "" val :test #'equal)
              (member "" weight :test #'equal) )
         ""                                                    ;; Then, blank space
         (progn                                                ;; Else
            (setq val (mapcar #'zl:read-from-string val) )
            (setq weight (mapcar #'zl:read-from-string weight) )
            (setq val*weight (mapcar #'* val weight) )
            (format nil "~5,1F" (apply #'+ val*weight) ) ) ) ) )


;*******************************************************************************
(defun TOTAL (dum1 dum2 dum3 &rest vars)
;;; Get the sum of a given list variables
   (ELIMINATE-COMPILER-WARNINGS-FOR-ARGS-NOT-USED dum1 dum2 dum3)
   (let* ( (sum 0.0) (values nil) )
      (dolist (variable vars)
         (setq values (append  values
                               (list (if   (VALOF '*variable-data-alist* variable)
                                           (read-from-string
                                              (VALOF '*variable-data-alist* variable ) )
                                           0.0) ) ) ) )
      (setq sum (apply #'+ values) )
      (format nil "~5,1F" sum) ) )
```

```
;*******************************************************************************
(defun INT-TOTAL (dum1 dum2 dum3 &rest vars)
;;; Total integer numbers, no decimal places
   (ELIMINATE-COMPILER-WARNINGS-FOR-ARGS-NOT-USED dum1 dum2 dum3)
   (let* ( (sum 0) (values nil) )
      (dolist (variable vars)
         (setq values (append  values
                               (list (if   (VALOF '*variable-data-alist* variable)
                                           (read-from-string
                                              (VALOF '*variable-data-alist* variable ) )
                                           0) ) ) ) )
      (setq sum (apply #'+ values) )
      (format nil "~5,0F" sum) ) )


;*******************************************************************************
(defun GET-DEMON-DATA (var context context-set context-val)
;;; Get the data values for a list variables
   (let ( (values (mapcar  #'ABSTRACT-DATA var
                           (make-list (length var) :initial-element context)
                           (make-list (length var) :initial-element context-set)
                           (make-list (length var) :initial-element context-val) ) ) )
      values) )


;*******************************************************************************
(defun ROUND-N-PLACES (number &optional (n 0) )
;;; Round off a value to a specified number of places
   (let ( (div (expt 10.0 n) ) )
      (/  (round (* number div) ) div) ) )


;*******************************************************************************
(defun READ-FROM-STRING-IF-VALUE (val)
   (if  (and val (stringp val) (not (string-equal val "")) )
      (read-from-string val nil nil) 0) )


;*******************************************************************************
(defun MULTIPLY-VALUES (&rest args)
;;; Multiply a set of values together, changing nil values to one (1).
   (let ( (new-args (mapcar #'ONE-IF-NIL args) ) )
      (apply #'* new-args) ) )


;*******************************************************************************
(defun ONE-IF-NIL (item)
;;; Change value of nil items to one (1).
   (if  (and item (stringp item) (not (string-equal item "")) )
      item
      1) )
```

```
;*****************************************************************************
(defun NORMALIZE (weights)
;;; Normalize a set of values

   (let ( (multiplier (/ 1 (apply #'+ weights) ) ) )
      (mapcar #'* weights
               (make-list (length weights) :initial-element multiplier) ) ) )


;*****************************************************************************
(defun NORMALIZE-FACTOR-WEIGHTS ( )
;;; Function to normalize weights of COA comparison measures to 100.

   (let* ( (factor-list '(*fc-pers-weight* *fc-equip-weight* *ec-pers-weight*
                          *ec-equip-weight* *re-pol-weight* *re-ammo-weight*
                          *feba-mvmt-weight* *time-rqd-weight* *subj-a-weight*
                          *subj-b-weight* *subj-c-weight* *subj-d-weight*
                          *subj-e-weight* *subj-f-weight* *subj-g-weight* *subj-h-weight*) )
          (table-list (make-list (length factor-list) :initial-element '*variable-data-alist*) )
          (val-list nil) (new-vals nil) )

      (dolist (var factor-list)
         (setq val-list (append  val-list
                                 (list (if (VALOF '*variable-data-alist* var)
                                           (read-from-string (VALOF '*variable-data-alist* var) )
                                           0.0) ) ) ) )

      (setq new-vals (NORMALIZE val-list) )
      (setq new-vals (mapcar #'* new-vals (make-list (length new-vals) :initial-element 100) ) )
      ;; Format values with 1 decimal place
      (setq new-vals (mapcar #'format
                             (make-list (length new-vals) :initial-element nil)
                             (make-list (length new-vals) :initial-element "~5,1F")
                             new-vals) )

      ;; Record the normalized values in *variable-data-alist*
      (mapcar #'MY-ASSERT table-list factor-list new-vals) ) )
```

```
;*****************************************************************************
(defun CALC-FC-PERS-SCALE-VAL (dum1 dum2 dum3 variable)
;;; The args 1, 2, 3, are used for consistency with other demon function calls
;   they should be eliminated when context info is no longer required for any calls
   (ELIMINATE-COMPILER-WARNINGS-FOR-ARGS-NOT-USED dum1 dum2 dum3)

   (let* ( (*fc-pers-auth* 21517)                                    ;; 13 Mar 89, Nick Carter
          (auth *fc-pers-auth*)
          (loss (if (VALOF '*variable-data-alist* variable)
                    (read-from-string (VALOF '*variable-data-alist* variable) )
                    0.0) )
          (percent (* (/ loss auth) 100) ) )

      (cond
         ( (<= percent 2.0) "9")
         ( (and (> percent 2.0) (<= percent 4.0) ) "8")
         ( (and (> percent 4.0) (<= percent 6.0) ) "7")
         ( (and (> percent 6.0) (<= percent 8.0) ) "6")
         ( (and (> percent 8.0) (<= percent 10.0) ) "5")
         ( (and (> percent 10.0) (<= percent 12.0) ) "4")
         ( (and (> percent 12.0) (<= percent 14.0) ) "3")
         ( (and (> percent 14.0) (<= percent 16.0) ) "2")
         (t "1") ) ) )                ;; >16%



;*****************************************************************************
(defun CALC-FC-EQUIP-SCALE-VAL (dum1 dum2 dum3 variable)
   (ELIMINATE-COMPILER-WARNINGS-FOR-ARGS-NOT-USED dum1 dum2 dum3)

   (let* ( (*fc-equip-auth* 945)                                     ;; 31 Mar 89, Nick Carter
          (auth *fc-equip-auth*)
          (loss (if (VALOF '*variable-data-alist* variable)
                    (read-from-string (VALOF '*variable-data-alist* variable) )
                    0.0) )
          (percent (* (/ loss auth) 100) ) )

      (cond
         ( (<= percent 10.0) "9")
         ( (and (> percent 10.0) (<= percent 18.0) ) "8")
         ( (and (> percent 18.0) (<= percent 24.0) ) "7")
         ( (and (> percent 24.0) (<= percent 29.0) ) "6")
         ( (and (> percent 29.0) (<= percent 34.0) ) "5")
         ( (and (> percent 34.0) (<= percent 38.0) ) "4")
         ( (and (> percent 38.0) (<= percent 41.0) ) "3")
         ( (and (> percent 41.0) (<= percent 43.0) ) "2")
         (t "1") ) ) )                ;; >43%
```

```
;*****************************************************************************
(defun CALC-EC-PERS-SCALE-VAL (dum1 dum2 dum3 variable)
   (ELIMINATE-COMPILER-WARNINGS-FOR-ARGS-NOT-USED dum1 dum2 dum3)

   (let* ( (*ec-pers-auth* 14020)                                  ;; 31 Mar 89, Nick Carter
           (auth *ec-pers-auth*)
           (loss (if (VALOF '*variable-data-alist* variable)
                     (read-from-string (VALOF '*variable-data-alist* variable) )
                     0.0) )
           (percent (* (/ loss auth) 100) ) )

      (cond
         ( (<= percent 2.0) "1")
         ( (and (> percent 2.0) (<= percent 4.0) ) "2")
         ( (and (> percent 4.0) (<= percent 6.0) ) "3")
         ( (and (> percent 6.0) (<= percent 8.0) ) "4")
         ( (and (> percent 8.0) (<= percent 10.0) ) "5")
         ( (and (> percent 10.0) (<= percent 12.0) ) "6")
         ( (and (> percent 12.0) (<= percent 14.0) ) "7")
         ( (and (> percent 14.0) (<= percent 16.0) ) "8")
         (t "9") ) ) )                    ;; >16%


;*****************************************************************************
(defun CALC-EC-EQUIP-SCALE-VAL (dum1 dum2 dum3 variable)
   (ELIMINATE-COMPILER-WARNINGS-FOR-ARGS-NOT-USED dum1 dum2 dum3)

   (let* ( (*ec-equip-auth* 879)                                   ;; 31 Mar 89, Nick Carter
           (auth *ec-equip-auth*)
           (loss (if (VALOF '*variable-data-alist* variable)
                     (read-from-string (VALOF '*variable-data-alist* variable) )
                     0.0) )
           (percent (* (/ loss auth) 100) ) )

      (cond
         ( (<= percent 10.0) "1")
         ( (and (> percent 10.0) (<= percent 18.0) ) "2")
         ( (and (> percent 18.0) (<= percent 24.0) ) "3")
         ( (and (> percent 24.0) (<= percent 29.0) ) "4")
         ( (and (> percent 29.0) (<= percent 34.0) ) "5")
         ( (and (> percent 34.0) (<= percent 38.0) ) "6")
         ( (and (> percent 38.0) (<= percent 41.0) ) "7")
         ( (and (> percent 41.0) (<= percent 43.0) ) "8")
         (t "9") ) ) )                    ;; >43%
```

```
;*****************************************************************************
(defun CALC-RE-POL-SCALE-VAL (dum1 dum2 dum3 variable)
   (ELIMINATE-COMPILER-WARNINGS-FOR-ARGS-NOT-USED dum1 dum2 dum3)

   (let* ( (used (if (VALOF '*variable-data-alist* variable)
                     (read-from-string (VALOF '*variable-data-alist* variable) )
                     0.0) ) )

      (cond
         ( (<= used 50.0) "9")
         ( (and (> used 50.0) (<= used 90.0) ) "8")
         ( (and (> used 90.0) (<= used 125.0) ) "7")
         ( (and (> used 125.0) (<= used 150.0) ) "6")
         ( (and (> used 150.0) (<= used 170.0) ) "5")
         ( (and (> used 170.0) (<= used 180.0) ) "4")
         ( (and (> used 180.0) (<= used 190.0) ) "3")
         ( (and (> used 190.0) (<= used 200.0) ) "2")
         (t "1") ) ) )                ;; >200%



;*****************************************************************************
(defun CALC-RE-AMMO-SCALE-VAL (dum1 dum2 dum3 variable)
   (ELIMINATE-COMPILER-WARNINGS-FOR-ARGS-NOT-USED dum1 dum2 dum3)

   (let* ( (used (if (VALOF '*variable-data-alist* variable)
                     (read-from-string (VALOF '*variable-data-alist* variable) )
                     0.0) ) )

      (cond
         ( (<= used 50.0) "9")
         ( (and (> used 50.0) (<= used 90.0) ) "8")
         ( (and (> used 90.0) (<= used 125.0) ) "7")
         ( (and (> used 125.0) (<= used 150.0) ) "6")
         ( (and (> used 150.0) (<= used 170.0) ) "5")
         ( (and (> used 170.0) (<= used 180.0) ) "4")
         ( (and (> used 180.0) (<= used 190.0) ) "3")
         ( (and (> used 190.0) (<= used 200.0) ) "2")
         (t "1") ) ) )                ;; >200%
```

```
;*****************************************************************************
(defun CALC-FEBA-MVMT-SCALE-VAL (dum1 dum2 dum3 variable)
;;; Calculate the FEBA movement scale value based on mission, values should be adjusted
;   for the particular scenario and commander's desires.
   (ELIMINATE-COMPILER-WARNINGS-FOR-ARGS-NOT-USED dum1 dum2 dum3)
   (let* ( (kilometers (if (VALOF '*variable-data-alist* variable)
                           (read-from-string (VALOF '*variable-data-alist* variable) )
                           0.0) ) )
      (case *mission*
         (offense
            (cond
               ( (> kilometers 38.0) "9")
               ( (and (> kilometers 36.0) (<= kilometers 38.0) ) "8")
               ( (and (> kilometers 34.0) (<= kilometers 36.0) ) "7")
               ( (and (> kilometers 31.0) (<= kilometers 34.0) ) "6")
               ( (and (> kilometers 28.0) (<= kilometers 31.0) ) "5")
               ( (and (> kilometers 24.0) (<= kilometers 28.0) ) "4")
               ( (and (> kilometers 18.0) (<= kilometers 24.0) ) "3")
               ( (and (> kilometers 10.0) (<= kilometers 18.0) ) "2")
               (t "1") ) )                    ;;<10 km
         (defense
            (cond
               ( (> kilometers 19.0) "1")
               ( (and (> kilometers 18.0) (<= kilometers 19.0) ) "2")
               ( (and (> kilometers 17.0) (<= kilometers 18.0) ) "3")
               ( (and (> kilometers 15.0) (<= kilometers 17.0) ) "4")
               ( (and (> kilometers 14.0) (<= kilometers 15.0) ) "5")
               ( (and (> kilometers 12.0) (<= kilometers 14.0) ) "6")
               ( (and (> kilometers 9.0) (<= kilometers 12.0) ) "7")
               ( (and (> kilometers 5.0) (<= kilometers 9.0) ) "8")
               (t "9") ) )                    ;;<5 km
         (retrograde
            (cond
               ( (> kilometers 38.0) "1")
               ( (and (> kilometers 36.0) (<= kilometers 38.0) ) "2")
               ( (and (> kilometers 34.0) (<= kilometers 36.0) ) "3")
               ( (and (> kilometers 31.0) (<= kilometers 34.0) ) "4")
               ( (and (> kilometers 28.0) (<= kilometers 31.0) ) "5")
               ( (and (> kilometers 24.0) (<= kilometers 28.0) ) "6")
               ( (and (> kilometers 18.0) (<= kilometers 24.0) ) "7")
               ( (and (> kilometers 10.0) (<= kilometers 18.0) ) "8")
               (t "9") ) )                    ;;<10 km
      ) ) )
```

```
;*****************************************************************************
(defun CALC-TIME-RQD-SCALE-VAL (dum1 dum2 dum3 variable)
   (ELIMINATE-COMPILER-WARNINGS-FOR-ARGS-NOT-USED dum1 dum2 dum3)
   (let* ( (hours (if  (VALOF '*variable-data-alist* variable)
                       (read-from-string (VALOF '*variable-data-alist* variable) )
                       0.0) ) )
      (cond
         ( (<= hours 24.0) "9")
         ( (and (> hours 24.0) (<= hours 26.0) ) "8")
         ( (and (> hours 26.0) (<= hours 29.0) ) "7")
         ( (and (> hours 29.0) (<= hours 32.0) ) "6")
         ( (and (> hours 32.0) (<= hours 36.0) ) "5")
         ( (and (> hours 36.0) (<= hours 42.0) ) "4")
         ( (and (> hours 42.0) (<= hours 50.0) ) "3")
         ( (and (> hours 50.0) (<= hours 60.0) ) "2")
         (t "1") ) ) )                    ;; >60 hrs
```

APPENDIX G

FILE COAAT-TEXT.LISP

APPENDIX G - FILE COAAT-TEXT.LISP

CONTENTS

>COAAT>COAAT-TEXT.LISP - 8/15/89

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Package: USER; Base: 10 -*-
;*********************************
;*      COAAT-TEXT.LISP          *
;*********************************
; Title, prompt, and instruction screens for the COAAT

;*****************************************************************************************
(defun MAIN-TITLE ( )
   (send *TITLE-PANE* ':line-out " ")
   (send *TITLE-PANE* ':line-out " ")
   (send *TITLE-PANE* ':line-out " ")
   (send *TITLE-PANE* ':line-out
         "                                    COURSE  OF  ACTION")
   (send *TITLE-PANE* ':line-out
         "                                     ASSESSMENT  TOOL")
   (send *TITLE-PANE* ':line-out " ")
   (send *TITLE-PANE* ':line-out
         "                                          ( COAAT )" ) )

;*****************************************************************************************
(defun MODULE-1-TITLE ( )
   (send *TITLE-PANE* ':line-out " ")
   (send *TITLE-PANE* ':line-out
         "                                  Critical Event Assignment Worksheet") )

;*****************************************************************************************
(defun MODULE-2-TITLE ( )
   (send *TITLE-PANE* ':line-out " ")
   (send *TITLE-PANE* ':line-out
         "                                    Critical Event War-Gaming") )

;*****************************************************************************************
(defun MODULE-3-TITLE ( )
   (send *TITLE-PANE* ':line-out " ")
   (send *TITLE-PANE* ':line-out
         "                                  Course of Action Comparison") )

;*****************************************************************************************
(defun EXPLAIN-START-COAAT ( )
;;;  A simple explanation of the COAAT process and mission specification
   (send *PROMPT-PANE* ':set-default-character-style '(:fix :bold :normal) )
   (send *PROMPT-PANE* ':line-out " ")
   (send *PROMPT-PANE* ':line-out
     "                  Course of Action Assessment Tool (COAAT)                  ")
   (send *PROMPT-PANE* ':line-out " ")
   (send *PROMPT-PANE* ':set-default-character-style '(:fix :roman :normal) )
```

```
   (send *PROMPT-PANE* ':line-out
      "    The first step in using COAAT is to specify the type of mission being analyzed.")
   (send *PROMPT-PANE* ':line-out
      " Next you must identify the critical events (CE's) which will occur during the     ")
   (send *PROMPT-PANE* ':line-out
      " performance of the mission and select the method you will use for organizing the ")
   (send *PROMPT-PANE* ':line-out
      " battlefield for analysis.  The CE's should be identified using the Sun system in  ")
   (send *PROMPT-PANE* ':line-out
      " conjunction with a map and tactical overlay analysis of the battle area.  CE      ")
   (send *PROMPT-PANE* ':line-out
      " identification should be entered on the tactical overlay in the location at which ")
   (send *PROMPT-PANE* ':line-out
      " you expect the CE to occur (use control measure symbology).  The next COAAT menu")
   (send *PROMPT-PANE* ':line-out
      " will ask for your selection of method of analysis; i.e., the way you desire to    ")
   (send *PROMPT-PANE* ':line-out
      " group CE's for analysis. ")
   (send *PROMPT-PANE* ':line-out " ")
   (send *PROMPT-PANE* ':line-out
      " Subsequent steps in the COAAT process will be:  ")
   (send *PROMPT-PANE* ':line-out " ")
   (send *PROMPT-PANE* ':line-out
      "      Selection and assignment of CE's to the appropriate COA's,")
   (send *PROMPT-PANE* ':line-out
      "      Wargaming of the CE's, and  ")
   (send *PROMPT-PANE* ':line-out
      "      Comparison of courses of action. ")
   (send *PROMPT-PANE* ':line-out " ")
   (send *PROMPT-PANE* ':line-out " ")
   (send *PROMPT-PANE* ':set-default-character-style '(:swiss :bold-italic :normal) )
   (send *PROMPT-PANE* ':line-out
      "-- Using the mouse, select mission type from the menu below --") )


;*******************************************************************************
(defun EXPLAIN-MOA ( )
;;; A simple explanation of the method of analysis selection
   (send *PROMPT-PANE* ':set-default-character-style '(:fix :bold :normal) )
   (send *PROMPT-PANE* ':line-out " ")
   (send *PROMPT-PANE* ':line-out
         "                                War-Game Methods                                ")
   (send *PROMPT-PANE* ':line-out " ")
   (send *PROMPT-PANE* ':set-default-character-style '(:fix :roman :normal) )
   (send *PROMPT-PANE* ':line-out
      "      Select your desired method of organizing the battlefield area for analysis.  ")
   (send *PROMPT-PANE* ':line-out
      " The generally accepted methods are presented below for your consideration.  When  ")
   (send *PROMPT-PANE* ':line-out
      " you assign the CE's to COA's, COAAT will group them in accordance with the method")
   (send *PROMPT-PANE* ':line-out
      " you have selected. ")
   (send *PROMPT-PANE* ':line-out " ")
```

```
  (send *PROMPT-PANE* ':line-out
    " Avenue in Depth:  The avenue-in-depth method focuses on one axis of advance at a ")
  (send *PROMPT-PANE* ':line-out
    " time starting with the main effort.  This method analyzes successive CE's along   ")
  (send *PROMPT-PANE* ':line-out
    " the main attack axis of advance until all CE's have been analyzed and all battle  ")
  (send *PROMPT-PANE* ':line-out
    " results have been assessed.  Supporting attacks are analyzed in the same manner.  ")
  (send *PROMPT-PANE* ':line-out
    " Results along all axes are aggregated to evaluate each Course Of Action. ")
  (send *PROMPT-PANE* ':line-out " ")
  (send *PROMPT-PANE* ':line-out
    " Belt:  The battlefield area is divided into successive belts generally parallel to")
  (send *PROMPT-PANE* ':line-out
    " the FEBA and running the width of the sector.  CE's in each belt are analyzed and ")
  (send *PROMPT-PANE* ':line-out
    " the results of all belts are summed to evaluate each COA.  A modified belt techni-")
  (send *PROMPT-PANE* ':line-out
    " que in which belts are not contiguous may be used. ")
  (send *PROMPT-PANE* ':line-out " ")
  (send *PROMPT-PANE* ':line-out
    " Box:  The box method is a microanalysis of a few CE's or areas, and the battle    ")
  (send *PROMPT-PANE* ':line-out
    " analysis is focused on those CE's.  The assumption is that all but a few CE's can ")
  (send *PROMPT-PANE* ':line-out
    " be handled effectively and successfully, and only a few are analyzed to show their")
  (send *PROMPT-PANE* ':line-out
    " major impact on battle outcome. ")
  (send *PROMPT-PANE* ':line-out " ")
  (send *PROMPT-PANE* ':line-out " ")
  (send *PROMPT-PANE* ':set-default-character-style '(:swiss :bold-italic :normal) )
  (send *PROMPT-PANE* ':line-out
    "-- Using the mouse, select desired method from the menu below --") )


;*****************************************************************************
(defun EXPLAIN-MODULE-1 ( )
;;; A simple explanation of the module 1 process, CE assignment to COA's.
  (send *INSTRUCTION-WINDOW* ':activate)
  (send *INSTRUCTION-WINDOW* ':clear-input)
  (send *INSTRUCTION-WINDOW* ':set-position 250 150)
  (send *INSTRUCTION-WINDOW* ':set-size 700 175)
  (send *INSTRUCTION-WINDOW* ':set-label '(:string "--" :style (:fix :italic :normal) ) )
  (send *INSTRUCTION-WINDOW* ':expose)
  (send *INSTRUCTION-WINDOW* ':line-out " ")
  (send *INSTRUCTION-WINDOW* ':set-default-character-style '(:fix :bold :normal) )
  (send *INSTRUCTION-WINDOW* ':line-out
    "                              Critical Event Assignment                              ")
  (send *INSTRUCTION-WINDOW* ':line-out " ")
  (send *INSTRUCTION-WINDOW* ':set-default-character-style '(:fix :roman :normal) )
  (send *INSTRUCTION-WINDOW* ':line-out
    "     The next step in using COAAT is to assign critical events (CE's) to courses of")
```

```
   (send *INSTRUCTION-WINDOW* ':line-out
      " action (COA's) and to group them in accordance with your method for organizing the")
   (send *INSTRUCTION-WINDOW* ':line-out
      " battlefield for analysis.  COAAT will present the COA's and the groupings you have")
   (send *INSTRUCTION-WINDOW* ':line-out
      " designated.  You must input the CE identification, choose a CE type from a menu,  ")

   (send *INSTRUCTION-WINDOW* ':line-out
      " input the objective for that CE, and add any short comment you desire.")
   (send *INSTRUCTION-WINDOW* ':line-out " ")   )


;*******************************************************************************
(defun EXPLAIN-MODULE-2 ( )
;;; A simple explanation of module 2 in COAAT.
   (send *INSTRUCTION-WINDOW* ':activate)
   (send *INSTRUCTION-WINDOW* ':set-position 250 150)
   (send *INSTRUCTION-WINDOW* :set-size 700 175)
   (send *INSTRUCTION-WINDOW* ':clear-input)
   (send *INSTRUCTION-WINDOW* ':set-label '(:string "--" :style (:fix :italic :normal) ) )
   (send *INSTRUCTION-WINDOW* ':expose)
   (send *INSTRUCTION-WINDOW* ':line-out " ")
   (send *INSTRUCTION-WINDOW* ':set-default-character-style '(:fix :bold :normal) )
   (send *INSTRUCTION-WINDOW* ':line-out
      "                          Critical Event War-Gaming                          ")
   (send *INSTRUCTION-WINDOW* ':line-out " ")
   (send *INSTRUCTION-WINDOW* ':set-default-character-style '(:fix :roman :normal) )
   (send *INSTRUCTION-WINDOW* ':line-out
      "     The next step in using COAAT is to war-game the CE's and to assess selected")
   (send *INSTRUCTION-WINDOW* ':line-out
      " battle results.  Using your own war-gaming technique or an available simulation  ")
   (send *INSTRUCTION-WINDOW* ':line-out
      " you will war-game each CE which you desire to analyze, assessing and recording ")
   (send *INSTRUCTION-WINDOW* ':line-out
      " battle results for each in the available war-gaming worksheet.  COAAT will sum   ")
   (send *INSTRUCTION-WINDOW* ':line-out
      " and scale the battle results for each COA to facilitate your comparison of them.  ")
   (send *INSTRUCTION-WINDOW* ':line-out "  ")   )


;*******************************************************************************
(defun EXPLAIN-MODULE-3 ( )
;;; A simple explanation of module 3 in the COAAT process.
   (send *INSTRUCTION-WINDOW* ':activate)
   (send *INSTRUCTION-WINDOW* ':set-position 250 150)
   (send *INSTRUCTION-WINDOW* ':set-size 700 275)
   (send *INSTRUCTION-WINDOW* ':set-label '(:string "--" :style (:fix :italic :normal) ) )
   (send *INSTRUCTION-WINDOW* ':clear-input)
   (send *INSTRUCTION-WINDOW* ':expose)
   (send *INSTRUCTION-WINDOW* ':line-out " ")
   (send *INSTRUCTION-WINDOW* ':set-default-character-style '(:fix :bold :normal) )
   (send *INSTRUCTION-WINDOW* ':line-out
      "                          Course of Action Comparison                        ")
```

```
   (send *INSTRUCTION-WINDOW* ':line-out " ")
   (send *INSTRUCTION-WINDOW* ':set-default-character-style '(:fix :roman :normal) )
   (send *INSTRUCTION-WINDOW* ':line-out
      "     The last major process using COAAT is the comparison of alternative courses of")
   (send *INSTRUCTION-WINDOW* ':line-out
      " action which have been analyzed so that a COA may be recommended to the
        Commander.")
   (send *INSTRUCTION-WINDOW* ':line-out
      "     You will provide weights for the objective measures (war-game assessments)   ")
   (send *INSTRUCTION-WINDOW* ':line-out
      " which you chose to use in war-gaming, as well as the subjective measures provided.")
   (send *INSTRUCTION-WINDOW* ':line-out
      " The weights are relative among both objective and subjective measures used and    ")
   (send *INSTRUCTION-WINDOW* ':line-out
      " reflect the relative degree to which each measure is deemed to affect mission     ")
   (send *INSTRUCTION-WINDOW* ':line-out
      " accomplishment, as well as the degree to which each provides a basis for comparing")
   (send *INSTRUCTION-WINDOW* ':line-out
      " COA's.  The combination of weights and scales yields an overall quantitative merit")
   (send *INSTRUCTION-WINDOW* ':line-out
      " of each COA and provides a basis for selecting a recommended COA.                 ")
   (send *INSTRUCTION-WINDOW* ':line-out " ")
   (send *INSTRUCTION-WINDOW* ':line-out
      "     This comparison along with a further comparison of advantages and disadvant-  ")
   (send *INSTRUCTION-WINDOW* ':line-out
       " ages provides you the basis to recommend a COA to the Commander. ")  )


;*****************************************************************************
(defun EXIT-EXPLAIN-MOD-1 ( )
;;; Called by driver routine after the screen display file has been loaded
   (send *INSTRUCTION-WINDOW* ':set-label '(:string
      "-- PRESS SPACE BAR TO EXIT THIS SCREEN TO THE CE ASSIGNMENT WORKSHEET--"
   :style (:fix :bold-italic :normal) ) )
   (sys:read-character)
   (send *INSTRUCTION-WINDOW* ':deactivate)
   (send *EDIT-WINDOW* :clear-window)   )


;*****************************************************************************
(defun EXIT-EXPLAIN-MOD-2 ( )
;;; Called by driver routine after the screen display file has been loaded
   (send *INSTRUCTION-WINDOW* :set-label '(:string
      "-- PRESS SPACE BAR TO EXIT THIS SCREEN TO THE WAR-GAMING WORKSHEET --"
   :style (:fix :bold-italic :normal) ) )
   (sys:read-character)
   (send *INSTRUCTION-WINDOW* ':deactivate)
   (send *EDIT-WINDOW* :clear-window)   )
```

```
;*****************************************************************************
(defun EXIT-EXPLAIN-MOD-3 ( )
;;; Called by driver routine after the screen display file has been loaded
   (send *INSTRUCTION-WINDOW* :set-label '(:string
      " PRESS SPACE BAR TO EXIT THIS SCREEN AND BEGIN THE COA COMPARISON
        PROCESS"
   :style (:fix :bold-italic :normal) ) )
   (sys:read-character)
   (send *INSTRUCTION-WINDOW* ':deactivate)
   (send *EDIT-WINDOW* :clear-window)   )


;*****************************************************************************
(defun DISPLAY-PROMPT (prompt-function x y)
;;; Display a prompt for the user, calling routine is responsible for
;    providing x and y for display location
   (send *PROMPT-WINDOW* :activate)
   (send *PROMPT-WINDOW* :center-around X Y)
   (send *PROMPT-WINDOW* :set-label nil)
   (send *PROMPT-WINDOW* :clear-input)
   (send *PROMPT-WINDOW* :expose)
   (funcall prompt-function)  )


;*****************************************************************************
(defun COA-EVAL-FACTOR-WEIGHTS ( )
   (send *PROMPT-WINDOW* :set-size 220 190)
   (send *PROMPT-WINDOW* ':line-out  " ")
   (send *PROMPT-WINDOW* ':set-default-character-style '(:fix :bold :normal) )
   (send *PROMPT-WINDOW* ':line-out  "COA Evaluation Factor")
   (send *PROMPT-WINDOW* ':line-out  "       Weights ")
   (send *PROMPT-WINDOW* ':set-default-character-style '(:fix :roman :normal) )
   (send *PROMPT-WINDOW* ':line-out  "-----------------")
   (send *PROMPT-WINDOW* ':line-out  " Enter whole number   ")
   (send *PROMPT-WINDOW* ':line-out  "weights which reflect ")
   (send *PROMPT-WINDOW* ':line-out  "the relative impact   ")
   (send *PROMPT-WINDOW* ':line-out  "of each factor on ")
   (send *PROMPT-WINDOW* ':line-out  "mission accomplishment")
   (send *PROMPT-WINDOW* ':line-out  "for the COA's.        ")   )
```

```
;*****************************************************************************
(defun SCALE-SUBJ-A ( )
   (send *PROMPT-WINDOW* :set-size 250 280)
   (GEN-SUBJECTIVE-SCALE-PROMPT) )

;*****************************************************************************
(defun SCALE-SUBJ-B ( )
   (send *PROMPT-WINDOW* :set-size 250 280)
   (GEN-SUBJECTIVE-SCALE-PROMPT) )

;*****************************************************************************
(defun SCALE-SUBJ-C ( )
   (send *PROMPT-WINDOW* :set-size 250 280)
   (GEN-SUBJECTIVE-SCALE-PROMPT) )

;*****************************************************************************
(defun SCALE-SUBJ-D ( )
   (send *PROMPT-WINDOW* :set-size 250 280)
   (GEN-SUBJECTIVE-SCALE-PROMPT) )

;*****************************************************************************
(defun SCALE-SUBJ-E ( )
   (send *PROMPT-WINDOW* :set-size 250 280)
   (RISK-SCALE-PROMPT) )

;*****************************************************************************
(defun SCALE-SUBJ-F ( )              ;; User input
   (send *PROMPT-WINDOW* :set-size 250 280)
   (GEN-SUBJECTIVE-SCALE-PROMPT) )

;*****************************************************************************
(defun SCALE-SUBJ-G ( )              ;; User input
   (send *PROMPT-WINDOW* :set-size 250 280)
   (GEN-SUBJECTIVE-SCALE-PROMPT) )

;*****************************************************************************
(defun SCALE-SUBJ-H ( )              ;; User input
   (send *PROMPT-WINDOW* :set-size 250 280)
   (GEN-SUBJECTIVE-SCALE-PROMPT) )
```

```
;*****************************************************************************
(Defun GEN-SUBJECTIVE-SCALE-PROMPT ( )
   (send *PROMPT-WINDOW* ':set-default-character-style '(:fix :bold :normal) )
   (send *PROMPT-WINDOW* ':line-out  "    INPUT SCALE VALUES ")
   (send *PROMPT-WINDOW* ':line-out  " ")
   (send *PROMPT-WINDOW* ':line-out  " Degree that the measure ")
   (send *PROMPT-WINDOW* ':line-out  " is supported by, or  ")
   (send *PROMPT-WINDOW* ':line-out  " incorporated in, the COA.")
   (send *PROMPT-WINDOW* ':set-default-character-style '(:fix :roman :normal) )
   (send *PROMPT-WINDOW* ':line-out  "-------------------")
   (send *PROMPT-WINDOW* ':line-out  " VALUE      DEGREE  ")
   (send *PROMPT-WINDOW* ':line-out  " ---   ------------ ")
   (send *PROMPT-WINDOW* ':line-out  "   9      High  ")
   (send *PROMPT-WINDOW* ':line-out  "   8 ")
   (send *PROMPT-WINDOW* ':line-out  "   7      Moderately High")
   (send *PROMPT-WINDOW* ':line-out  "   6 ")
   (send *PROMPT-WINDOW* ':line-out  "   5      Moderate ")
   (send *PROMPT-WINDOW* ':line-out  "   4 ")
   (send *PROMPT-WINDOW* ':line-out  "   3      Moderately Low")
   (send *PROMPT-WINDOW* ':line-out  "   2 ")
   (send *PROMPT-WINDOW* ':line-out  "   1      Low      ")      )


;*****************************************************************************
(Defun RISK-SCALE-PROMPT ( )
   (send *PROMPT-WINDOW* ':set-default-character-style '(:fix :bold :normal) )
   (send *PROMPT-WINDOW* ':line-out  "    INPUT SCALE VALUES ")
   (send *PROMPT-WINDOW* ':line-out  " ")
   (send *PROMPT-WINDOW* ':line-out  "  Degree of Risk in the  ")
   (send *PROMPT-WINDOW* ':line-out  "  Course of Action. ")
   (send *PROMPT-WINDOW* ':set-default-character-style '(:fix :roman :normal) )
   (send *PROMPT-WINDOW* ':line-out  "-------------------")
   (send *PROMPT-WINDOW* ':line-out  " VALUE      DEGREE  ")
   (send *PROMPT-WINDOW* ':line-out  " ---   ------------ ")
   (send *PROMPT-WINDOW* ':line-out  "   9      Low Risk  ")
   (send *PROMPT-WINDOW* ':line-out  "   8 ")
   (send *PROMPT-WINDOW* ':line-out  "   7      Moderately Low ")
   (send *PROMPT-WINDOW* ':line-out  "   6 ")
   (send *PROMPT-WINDOW* ':line-out  "   5      Moderate Risk")
   (send *PROMPT-WINDOW* ':line-out  "   4 ")
   (send *PROMPT-WINDOW* ':line-out  "   3      Moderately High ")
   (send *PROMPT-WINDOW* ':line-out  "   2 ")
   (send *PROMPT-WINDOW* ':line-out  "   1      High Risk ")      )
```

```
;*****************************************************************************
(defun ATK-AXIS-PROMPT ( )
   (send *PROMPT-WINDOW* ':line-out " ")
   (send *PROMPT-WINDOW* ':set-default-character-style '(:fix :bold :normal) )
   (send *PROMPT-WINDOW* ':line-out "   Main Attack Axis   ")
   (send *PROMPT-WINDOW* ':set-default-character-style '(:fix :roman :normal) )
   (send *PROMPT-WINDOW* ':line-out " ---------- ")
   (send *PROMPT-WINDOW* ':line-out " ")
   (send *PROMPT-WINDOW* ':line-out " Enter the letter    ")
   (send *PROMPT-WINDOW* ':line-out "designator of the axis")
   (send *PROMPT-WINDOW* ':line-out "which will be, or is  ")
   (send *PROMPT-WINDOW* ':line-out "expected to be the    ")
   (send *PROMPT-WINDOW* ':line-out "main attack for this  ")
   (send *PROMPT-WINDOW* ':line-out "COA. ")
   (send *PROMPT-WINDOW* ':line-out " ")
   (send *PROMPT-WINDOW* ':line-out "-- Press Return to    ")
   (send *PROMPT-WINDOW* ':line-out "   complete entry. -- ") )

;*****************************************************************************
(defun MOA-NAME-PROMPT ( )
   (let ( (line1 (string-append "  Name of This " (string-capitalize *moa-type*) ))
          (line3 (string-append "this " (string-capitalize *moa-type*) " of CE's.") ))
   (send *PROMPT-WINDOW* ':line-out " ")
   (send *PROMPT-WINDOW* ':set-default-character-style '(:fix :bold :normal) )
   (send *PROMPT-WINDOW* ':line-out line1)
   (send *PROMPT-WINDOW* ':set-default-character-style '(:fix :roman :normal) )
   (send *PROMPT-WINDOW* ':line-out " ---------- ")
   (send *PROMPT-WINDOW* ':line-out " ")
   (send *PROMPT-WINDOW* ':line-out "  Enter your name for ")
   (send *PROMPT-WINDOW* ':line-out line3)
   (send *PROMPT-WINDOW* ':line-out "There are eight (8)   ")
   (send *PROMPT-WINDOW* ':line-out "character spaces      ")
   (send *PROMPT-WINDOW* ':line-out "available.")
   (send *PROMPT-WINDOW* ':line-out " ")
   (send *PROMPT-WINDOW* ':line-out "-- Press Return to    ")
   (send *PROMPT-WINDOW* ':line-out "   complete entry. -- ") ) )

;*****************************************************************************
(defun CE-NAME-PROMPT ( )
   (send *PROMPT-WINDOW* ':line-out " ")
   (send *PROMPT-WINDOW* ':line-out " ")
   (send *PROMPT-WINDOW* ':set-default-character-style '(:fix :bold :normal) )
   (send *PROMPT-WINDOW* ':line-out "  Critical Event Name ")
   (send *PROMPT-WINDOW* ':set-default-character-style '(:fix :roman :normal) )
   (send *PROMPT-WINDOW* ':line-out " ---------- ")
   (send *PROMPT-WINDOW* ':line-out " ")
   (send *PROMPT-WINDOW* ':line-out "  CE name must be a   ")
   (send *PROMPT-WINDOW* ':line-out "single letter followed")
   (send *PROMPT-WINDOW* ':line-out "by a one or two digit ")
   (send *PROMPT-WINDOW* ':line-out "sequence number.  Any ")
   (send *PROMPT-WINDOW* ':line-out "CE may be used only   ")
   (send *PROMPT-WINDOW* ':line-out "once for a COA. ")
   (send *PROMPT-WINDOW* ':line-out " ")   )
```

APPENDIX H

FILE COAAT-FLAVOR.LISP

CONTENTS

>COAAT>COAAT-FLAVOR.LISP - 8/14/89

```
;;; ... -*- Mode: LISP; Syntax: Common-lisp; Package: USER; Base: 10 -*-
;*******************************
;*     COAAT-FLAVOR.LISP     *
;*******************************
;;; Flavor definitions, methods, and window definitions for COAAT

;*****************************************************************************************
(defflavor COAAT-WINDOW-FLAVOR ( )                              ;; Basic COAAT window
          (tv:bordered-constraint-frame-with-shared-io-buffer
           tv:process-mixin
           tv:window-with-typeout-mixin
           tv:essential-mouse)   )


;*****************************************************************************************
(defflavor COAAT-INTERACTION-PANE ( )                           ;; COAAT pane for interaction
          (tv:window-pane)
   (:default-init-plist
    :save-bits t
    :more-p t
    :borders '(4 2 4 2) )  )


;*****************************************************************************************
(defflavor TITLE-PANE-FLAVOR ( )                                ;; Module title display pane
          (tv:pane-no-mouse-select-mixin
           tv:stream-mixin
           tv:borders-mixin
           tv:minimum-window)
   (:default-init-plist
    :blinker-p nil
    :default-character-style '(:swiss :bold :large)
    :borders '(4 2 4 1)
    :reverse-video-p nil
    :save-bits t
    :expose-p t
    :more-p nil
    :deexposed-typeout-action :permit) )
```

```
;*****************************************************************************
(defflavor MAIN-TITLE-FLAVOR ( )                          ;; Main title display pane
           (tv:pane-no-mouse-select-mixin
            tv:stream-mixin
            tv:borders-mixin
            tv:minimum-window)
   (:default-init-plist
    :blinker-p nil
    :default-character-style '(:eurex :italic :huge)
    :borders '(4 4 4 1)
    :save-bits t
    :expose-p t
    :more-p nil
    :deexposed-typeout-action :permit) )


;*****************************************************************************
(defflavor COAAT-WORKSHEET ( )                        ;; War-gaiming worksheet window
           (tv:borders-mixin
            tv:centered-label-mixin
            tv:top-box-label-mixin
            tv:window) )


;*****************************************************************************
(defflavor COAAT-POP-UP-PROMPT ( )                    ;; Pop-up user prompt window
           (tv:pop-up-text-window)
   (:default-init-plist
    :width 220
    :height 275
    :more-p nil
    :blinker-p nil
    :save-bits nil
    :borders '(8 3 8 3)
    :border-margin-width 10)  )


;*****************************************************************************
(defflavor COAAT-INSTRUCTION-WINDOW ( )          ;; Window for instructions to the user
           (tv:pop-up-text-window)
   (:default-init-plist
    :blinker-p nil
    :save-bits nil
    :more-p nil
    :borders '(10 5 10 5)
    :border-margin-width 5)  )
```

```
;*****************************************************************************
(defflavor COAAT-POP-UP-INPUT-WINDOW ( )          ;; Small pop-up window for user input
        (tv:pop-up-text-window tv:top-box-label-mixin)
   (:default-init-plist
    :save-bits nil
    :more-p nil
    :border-margin-width 5
    :borders '(3 2 3 2) ) )

;*****************************************************************************
(defmethod (:init COAAT-WINDOW-FLAVOR :after) (&rest init-plist)
   (ignore init-plist)
   (setq tv:typeout-window (tv:make-window 'tv:temporary-typeout-window :superior si:self) ) )

;*****************************************************************************
(defmethod (:who-line-documentation-string COAAT-POP-UP-PROMPT) ( )
   "Enter appropriate input as requested." )

;*****************************************************************************
(defmethod (:who-line-documentation-string COAAT-INSTRUCTION-WINDOW) ( )
   "Press space bar to exit this screen." )

;*****************************************************************************
(defmethod (:who-line-documentation-string COAAT-INTERACTION-PANE) ( )
   nil )

;*****************************************************************************
(setq *COAAT-WINDOWS*          ;; Define the COAAT window, its panes and configurations
      (tv:make-window
         'COAAT-WINDOW-FLAVOR
         ':selected-pane 'command-window
         ':process '(LISP-EVAL-READ-PRINT)
         ':save-bits t
         ':default-character-style '(:fix :roman :normal)
         ':panes
         '(  (command-window
               COAAT-INTERACTION-PANE
               :label nil)
             (edit-window
               COAAT-INTERACTION-PANE
               :label nil)
             (prompt-pane
               tv:window-pane
               :label nil
               :blinker-p nil
               :borders (4 nil 4 nil)
               :border-margin-width 175)
```

```
        (title-pane
          TITLE-PANE-FLAVOR)
        (main-title-pane
          MAIN-TITLE-FLAVOR)
        (mission-menu
          tv:command-menu-pane
          :item-list
            ( (" Offense " :value :offense :documentation " Mission is Offense")
              (" Defense " :value :defense :documentation " Mission is Defense")
              (" Retrograde " :value :retrograde :documentation " Mission is Retrograde") )
          :borders (5 5 5 1)
          :vsp 10
          :label (:string
          "                                                MISSION SPECIFICATION MENU "
                  :style (:Swiss :Bold :Large) ) )
        (moa-menu
          tv:command-menu-pane
          :item-list
            ( (" Avenue in Depth " :value :avenue
               :documentation " Use the Avenue in Depth method for war-gaming ")
              ("      Belt      " :value :belt
               :documentation " Use the Belt method for war-gaming ")
              ("      Box       " :value :box
               :documentation " Use the Box method for war-gaming ") )
          :borders (5 5 5 1)
          :vsp 10
          :label (:string
           "                                              METHOD OF ANALYSIS SELECTION MENU"
          :style (:Swiss :Bold :Large) ) )
        (process-menu
          tv:command-menu-pane
          :name "Main Menu"
          ; Item-list is set by RESET-COAAT using the function PROCESS-MENU-LIST
          :borders (5 5 5 1)
          :vsp 10
          :label (:string
           "                                                PROCESS SELECTION MENU "
          :style (:Swiss :Bold :Large) ) )  )
      ':configurations
      '( (mission-spec-scrn
          (:layout
            (mission-spec-scrn  :column main-title-pane prompt-pane
                                mission-menu command-window) )
          (:sizes
            (mission-spec-scrn  (main-title-pane 10 :lines)
                                :then (command-window 5 :lines)
                                :then (mission-menu :ask :pane-size)
                                :then (prompt-pane :even) ) ) )
```

```
        (moa-scrn
          (:layout
            (moa-scrn  :column main-title-pane prompt-pane
                       moa-menu command-window) )
          (:sizes
            (moa-scrn  (main-title-pane 8 :lines)
                       :then (command-window 2 :lines)
                       :then (mca-menu :ask :pane-size)
                       :then (prompt-pane :even) ) ) )
        (process-scrn
          (:layout
            (process-scrn :column main-title-pane process-menu
                          prompt-pane command window) )
          (:sizes
            (process-scrn (main-title-pane 10 :lines)
                          :then (command-window 2 :lines)
                          :then (process-menu :ask :pane-size)
                          :then (prompt-pane :even) )  )  )
        (mod-1-scrn
          (:layout
            (mod-1-scrn :column title-pane edit-window command-window)  )
          (:sizes
            (mod-1-scrn   (title-pane 3 :lines)
                          :then (command-window 2 :lines)
                          :then (edit-window :even) )  )  )
        (mod-2-scrn
          (:layout
            (mod-2-scrn :column title-pane edit-window command-window)  )
          (:sizes
            (mod-2-scrn   (title-pane 3 :lines)
                          :then (command-window 2 :lines)
                          :then (edit-window :even) )  )  )
        (mod-3-scrn
          (:layout
            (mod-3-scrn :column title-pane edit-window command-window)  )
          (:sizes
            (mod-3-scrn   (title-pane 3 :lines)
                          :then (command-window 2 :lines)
                          :then (edit-window :even) )  )  )
  ) ) )


;*****************************************************************************
(setq *POP-UP-MENU* (tv:make-window                          ;; Define the pop-up menu
                      'tv:pop-up-menu
                      ':borders 3
                      ':vsp 5
                      ':border-margin-width 4
                      ':label '( :string "  Choose one:   "
                                 :style (:Swiss :Bold :Normal) )
                      ':default-character-style '(:fix :roman :normal) ) )
```

```
;*****************************************************************************
(setq *INSTRUCTION-WINDOW*
;;; Used to display instructions to the user, display is terminated by the user pressing
;   any key -- *prompt-window* is also used for instructions to the user, but requires a
;   specific response (input) and is terminated by that input
   (tv:make-window
    'COAAT-INSTRUCTION-WINDOW
    ':name "Instruction Window") )


;*****************************************************************************
(setq *worksheet-window*                          ;; Define the war-gaming worksheet
   (tv:make-window
    'COAAT-WORKSHEET
    ':left 200
    ':top 530
    ':width 825
    ':height 225
    ':label '(:string "CRITICAL EVENT WAR-GAMING WORKSHEET")
    ':borders '(3 2 3 2) ) )


;*****************************************************************************
(compile-flavor-methods  COAAT-WINDOW-FLAVOR
                         COAAT-INTERACTION-PANE
                         TITLE-PANE-FLAVOR
                         MAIN-TITLE-FLAVOR
                         COAAT-WORKSHEET
                         COAAT-POP-UP-PROMPT
                         COAAT-POP-UP-INPUT-WINDOW
                         COAAT-INSTRUCTION-WINDOW)
```

APPENDIX I

FILE COAAT-SENS-ANAL.LISP

APPENDIX I - FILE COAAT-SENS-ANAL.LISP

CONTENTS

>COAAT>COAAT-SENS-ANAL.LISP - 10/11/89

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Package: USER; Base: 10 -*-
;*******************************
;*      COAAT-SENS-ANAL      *
;*******************************
;;; Functions required for sensitivity analysis of weights of COA comparison measures

;*****************************************************************************************
(defun FACTOR-WT-SENS-ANAL  ( )
;;; Main control for the sensitivity analysis of weights of COA comparison measures

   (let ( (which-set nil) (this-factor nil) (minimum nil) (maximum nil)
          (factor-list nil) (cur-tot-1 nil) (cur-tot-2 nil) (factor-name nil)
          (current-wt nil) (wt-variable nil) (scale-1 nil) (scale-2 nil)
          (obj-factor-list '(  (fc-pers  "Friendly Cas, Pers")
                               (fc-equip "Friendly Loss, Equip")
                               (ec-pers  "Enemy Cas, Pers")
                               (ec-equip "Enemy Loss, Equip")
                               (re-pol   "POL Expended")
                               (re-ammo  "Ammo Expended")
                               (feba-mvmt "FEBA Movement ")
                               (time-rqd "Time Required") ) )
          (subj-factor-list '( (subj-a "Accomplish Mission")
                               (subj-b "Effective Use of Assets")
                               (subj-c "Flexibility")
                               (subj-d "Facilitates Future Ops")
                               (subj-e "Risk") ) )
          (variable-factors-list '(subj-f subj-g subj-h) )          )

      ; Add the subjective measures with variable names to the list
      (dolist (item variable-factors-list)
         (let ( (add-element (list (list  item
                                          (VALOF  '*variable-data-alist*
                                                  (read-from-string
                                                     (string-append "*" item "-name*") ) ) ) ) ) ) )
            (setq subj-factor-list (append subj-factor-list add-element) ) ) )

      (loop                                                    ;; Loop until exit is chosen
         ; Put up menu for choice of the set of measures or exit sensitivity analysis
         (setq which-set (OBJ-OR-SUBJ-FACTORS-?) )

         (when (equalp which-set 'exit)
            (return) )

         (if (equalp which-set 'obj)
            (progn
               (setq cur-tot-1 (read-from-string
                                  (VALOF '*variable-data-alist* 'wg-subtot-1) ) )
               (setq cur-tot-2 (read-from-string
                                  (VALOF '*variable-data-alist* 'wg-subtot-2) ) )
               (setq factor-list obj-factor-list) )
```

```
    (progn
      (setq cur-tot-1 (read-from-string
                        (VALOF '*variable-data-alist* 'subj-subtot-1) ) )
      (setq cur-tot-2 (read-from-string
                        (VALOF '*variable-data-alist* 'subj-subtot-2) ) )
      (setq factor-list subj-factor-list) ) )

; If either COA score is zero, then no sensitivity can exist, Return to Mod 3.
(when (or (equalp cur-tot-1 0) (equalp cur-tot-2 0) )
   (NOTIFY-NO-SENSITIVITY 'coa)
   (return) )

; Put up the list of measures and return name for the chosen measure
(setq this-factor (CHOOSE-FACTOR factor-list which-set) )

; Get scale values for the factor.
(setq scale-1 (read-from-string
                (if (equalp "" (VALOF '*variable-data-alist*
                                      (read-from-string
                                        (string-append factor "-1-scale") ) ) )
                    "0"                                    ;; Replace "" with zero
                    (VALOF '*variable-data-alist*
                           (read-from-string
                             (string-append factor "-1-scale") ) ) ) ) )
(setq scale-2 (read-from-string
                (if (equalp "" (VALOF '*variable-data-alist*
                                      (read-from-string
                                        (string-append factor "-2-scale") ) ) )
                    "0"                                    ;; Replace "" with zero
                    (VALOF '*variable-data-alist*
                           (read-from-string
                             (string-append factor "-2-scale") ) ) ) ) )
; If the scale values are equal or zero for the factor, then no sensitivity exists, Loop.
(if (or (equalp scale-1 scale-2) (equalp scale-1 0) (equalp scale-2 0) )
   (NOTIFY-NO-SENSITIVITY 'scale)
   (progn                                  ; Else perform sensitivity analysis
      ; Get the display name line for this measure
      (setq factor-name (second (assoc this-factor factor-list :test #'equal) ) )

      ; Get the current weight for this measure
      (setq wt-variable (read-from-string
                          (string-append "*" this-factor "-weight*") ) )
      (setq current-wt (read-from-string
                         (VALOF '*variable-data-alist* wt-variable) ) )
      ; Blank values for minimum and maximum prior to each new input
      (MY-ASSERT '*variable-data-alist* '*sa-minimum* " " )
      (MY-ASSERT '*variable-data-alist* '*sa-maximum* " " )

      ; Get the maximum and minimum weight values to be applied
      (multiple-value-setq (minimum maximum)
         (GET-MIN-MAX factor-name current-wt) )
```

```
            ; Display window for output results with header and current date
            (DISPLAY-SENS-ANAL factor-name current-wt cur-tot-1 cur-tot-2)

            ; Calculate and display the sensitivity analysis results
            (COMPUTE-SENS-ANAL-RESULTS
              this-factor scale-1 scale-2 minimum maximum factor-list) ) ) )      ;; End loop
    ) )


;*****************************************************************************
(defun OBJ-OR-SUBJ-FACTORS-? ( )
;;; Get the user's choice of objective or subjective measures or exit sensitivity analysis
   (let ( (answer nil)
          (menu-list '( ( "   Objective Measures      " :value obj)
                        ( "   Subjective Measures     " :value subj)
                        ( "                           " :no-select t)
                        ( " Exit Sensitivity Analysis " :value exit) ) )
          (label  '(:string " Choose Measures to Analyze "
                    :style (:swiss :bold :normal) ) )
          (menu-x 750) (menu-y 150) (mouse-x 100) (mouse-y 60)   )

      (setq answer (POP-UP-MENU-CHOICE
                       menu-list label menu-x menu-y mouse-x mouse-y) ) ) )

;*****************************************************************************
(defun NOTIFY-NO-SENSITIVITY (why)
;;; Notify the user that sensitivity analysis is not possible either because one of the
;   COA scores is zero or because scale values are equal or one is zero.
   (tv:with-mouse-usurped
      (send *INSTRUCTION-WINDOW*  ':activate)
      (send *INSTRUCTION-WINDOW*  ':clear-input)
      (send *INSTRUCTION-WINDOW*  ':set-position 750 150)
      (send *INSTRUCTION-WINDOW*  ':set-size 290 140)
      (send *INSTRUCTION-WINDOW*  ':set-label '(:string " " :style (:fix :bold-italic :normal) ) )
      (send *INSTRUCTION-WINDOW*  ':expose)
      (send *INSTRUCTION-WINDOW*  ':set-default-character-style '(:fix :bold :normal) )
      (send *INSTRUCTION-WINDOW*  ':line-out " ")
      (send *INSTRUCTION-WINDOW*  ':line-out " ")
      (send *INSTRUCTION-WINDOW*  ':line-out "     NO SENSITIVITY CAN EXIST " )
      (send *INSTRUCTION-WINDOW*  ':line-out " ")
      (when (equalp why 'coa)
         (send *INSTRUCTION-WINDOW*  ':line-out " WHEN EITHER COA SCORE IS ZERO ") )
      (when (equalp why 'scale)
         (send *INSTRUCTION-WINDOW*  ':line-out " WHEN SCALE VALUES ARE EQUAL OR")
         (send *INSTRUCTION-WINDOW*  ':line-out " WHEN EITHER SCALE VALUE IS ZERO") )
      (send *INSTRUCTION-WINDOW*  ':line-out " ")
      (send *INSTRUCTION-WINDOW*  ':line-out " ")
      (send *INSTRUCTION-WINDOW*  ':set-label '(:string " Press Space Bar to Exit "
                                                :style (:fix :bold-italic :normal) ) )
      (sys:read-character)
      (send *INSTRUCTION-WINDOW*  ':deactivate) ) )
```

```
;*****************************************************************************
(defun CHOOSE-FACTOR (factor-list set)
;;; Get user's choice of specific measure for sensitivity analysis
   (let ( (answer nil)
          (label (if (equalp set 'obj)
                     '(:string "  OBJECTIVE (WAR-GAMING) MEASURES  "
                       :style (:swiss :roman :normal) )
                     '(:string "       SUBJECTIVE MEASURES         "
                       :style (:swiss :roman :normal) ) ) )
          (menu-list (MAKE-FACTOR-MENU-LIST factor-list) )
          (menu-x 750) (menu-y 150) (mouse-x 10) (mouse-y 80)   )

      (setq answer (POP-UP-MENU-CHOICE
                          menu-list label menu-x menu-y mouse-x mouse-y) ) ) )


;*****************************************************************************
(defun MAKE-FACTOR-MENU-LIST (factor-list)
;;; Make the list of menu items with values and documentation
   (let* ((num-elem (length factor-list) ) (val-list nil) (menu-list nil)
          (doc-list (make-list num-elem
                               :initial-element
                               (list :documentation " Left: Select Measure") ) )  )

      (dolist (pair factor-list)
         (if (and (first pair) (second pair) )                    ;; If either is nil, skip the pair
            (setq val-list (append val-list
                                   (list (list (format nil "~25a" (second pair) )
                                               :value (first pair) ) ) ) ) ) )
      (setq menu-list (mapcar #'append val-list doc-list) )
      (setq menu-list (append   (list (list "  " :no-select 't) )
                                menu-list) ) ) )


;*****************************************************************************
(defun GET-MIN-MAX (name weight)
;;; Get the minimum and maximum values to be applied for sensitivity analysis
   (let* ((wt (format nil "~4@s" weight) )
          (table
             '(  (NEWLINE)
                 (TFIELD 31 " Enter the minimum and maximum % weight values for the measure:")
                 (NEWLINE)
                 (FFIELD 3 " ")
                 (DFIELD 25 ,name name) (NEWLINE 2)
                 (FFIELD 17 "   Current Value ")
                 (DFIELD 6 ,wt weight) (NEWLINE 2)
                 (FFIELD 6 " ") (FFIELD 8 "Minimum ")
                 (IFIELD 3 numeric " " *sa-minimum*) (NEWLINE)
                 (FFIELD 6 " ") (FFIELD 8 "Maximum ")
                 (IFIELD 3 numeric " " *sa-maximum*) (NEWLINE 2) ) )
          (dum1) (dum2) (dum3) (dum4) (dum5)        ;; Dummy return variables for input-table
          (minimum nil) (maximum nil)  )
```

```
(setq input-window
      (tv:make-window
       'COAAT-POP-UP-INPUT-WINDOW
       ':left 750
       ':top 150
       ':width 260
       ':height 150
       ':label nil) )
(send input-window :expose)
(send input-window :select)

(loop
   (multiple-value-setq (dum1 dum2 dum3 dum4 dum5)
      (INPUT-TABLE input-window table) )
   ; Now get the values from the data alist
   (setq minimum (read-from-string (VALOF '*variable-data-alist* '*sa-minimum*) ) )
   (setq maximum (read-from-string (VALOF '*variable-data-alist* '*sa-maximum*) ) )
   (if (> maximum minimum)                          ;; Make sure max is greater than min
      (return)
      (send input-window :clear-window) ) )                ;; Clear window before looping

(send input-window :deactivate)
(values minimum maximum) ) )



;*******************************************************************************
(defun DISPLAY-SENS-ANAL (name wt score-1 score-2)
;;; Display the window for sensitivity analysis results with header information
   (send *INSTRUCTION-WINDOW* ':activate)
   (send *INSTRUCTION-WINDOW* ':clear-input)
   (send *INSTRUCTION-WINDOW* ':set-position 750 150)
   (send *INSTRUCTION-WINDOW* ':set-size 250 350)
   (send *INSTRUCTION-WINDOW* ':expose)
   (send *INSTRUCTION-WINDOW* ':line-out " ")
   (send *INSTRUCTION-WINDOW* ':set-default-character-style '(:fix :bold :normal) )
   (setq name (format nil "~25:@<~a~>" name) )                    ;; Center name in a string
   (send *INSTRUCTION-WINDOW* ':line-out name)
   (send *INSTRUCTION-WINDOW* ':line-out " ")
   (send *INSTRUCTION-WINDOW* ':set-default-character-style '(:fix :roman :normal) )
   (send *INSTRUCTION-WINDOW* ':line-out "      Current Values")
   (send *INSTRUCTION-WINDOW* ':line-out " Weight    COA-1    COA-2")
   (send *INSTRUCTION-WINDOW* ':line-out
   (string-append "  " (format nil "~3@s" wt) "      "
                  (format nil "~6@s" score-1) "    " (format nil "~6@s" score-2) ) )
   (send *INSTRUCTION-WINDOW* ':line-out " ")
   (send *INSTRUCTION-WINDOW* ':line-out "  Sensitivity Analysis")
   (send *INSTRUCTION-WINDOW* ':line-out " Weight    COA-1    COA-2")  )
```

```
;*****************************************************************************
(defun COMPUTE-SENS-ANAL-RESULTS (factor scale-1 scale-2 min max factor-list)
;;; Compute and display sensitivity analysis results
   (let ( (sa-wt nil) (sa-wtd-score-1 nil) (sa-wtd-score-2 nil)
          (others-wtd-score-1 nil) (others-wtd-score-2 nil) (total-1 nil) (total-2 nil)
          (increment (/ (- max min) 10.0) ) )

      (multiple-value-setq (others-wtd-score-1 others-wtd-score-2)
         (SUM-OTHER-SCORES factor factor-list) )
      (dotimes (i 11)                                    ;; Zero plus 10 increments
         (setq sa-wt (+ min (* i increment) ) )
         (setq sa-wtd-score-1 (* sa-wt scale-1) )
         (setq sa-wtd-score-2 (* sa-wt scale-2) )
         (setq total-1 (+ sa-wtd-score-1 others-wtd-score-1) )
         (setq total-2 (+ sa-wtd-score-2 others-wtd-score-2) )
         ; Output each result, asterisk identfies the higher
         (if (> total-1 total-2)
            (send *INSTRUCTION-WINDOW* ':line-out
                  (string-append  " " (format nil "~5,1F" sa-wt)
                                  "   " (format nil "~5,1F" total-1)
                                  "*  " (format nil "~5,1F" total-2) ) )
            (send *INSTRUCTION-WINDOW* ':line-out
                  (string-append  " " (format nil "~5,1F" sa-wt)
                                  "   " (format nil "~5,1F" total-1)
                                  "   " (format nil "~5,1F" total-2) "*") ) ) )    ;; End the loop

      (send *INSTRUCTION-WINDOW* ':set-label
            '( :string " - SPACE BAR TO CONTINUE --"
               :style (:swiss :bold :small) ) )
      (send *INSTRUCTION-WINDOW* ':clear-input)          ;; Clear extraneous prior input
      (sys:read-character)
      (send *INSTRUCTION-WINDOW* ':deactivate)  ) )


;*****************************************************************************
(defun SUM-OTHER-SCORES (factor factor-list)
;;; Calculate the total score for each COA for all measures except the one chosen for
;   sensitivity analysis
   (let ( (subtot-1 0) (subtot-2 0)
          (item-list (BUILD-OTHER-FACTOR-LIST factor factor-list) )  )

      (dotimes (i 2)                                     ;; Zero and 1
         (let ( (j (format nil "~s" (+ i 1) ) ) (total 0)   )
            (dolist (item item-list)
               (let ( (var (read-from-string
                             (string-append item "-" j "-wtd-score") ) ) )
                  (setq total (+ total (read-from-string
                                         (VALOF '*variable-data-alist* var) ) ) ) ) )
            (if (equal j "1")
               (setq subtot-1 total)
               (setq subtot-2 total) ) ) )

      (values subtot-1 subtot-2) ) )
```

```
;*****************************************************************************
(defun BUILD-OTHER-FACTOR-LIST (factor factor-list)
;;; Build a list of names of all measures from this set except the one
;   chosen for sensitivity analysis
   (let ( (other-factor-list nil)  )

      (dolist (factor-pair factor-list)
         (when (not (equalp (first factor-pair) factor) )
            (setq other-factor-list (append   other-factor-list
                                              (list (first factor-pair) ) ) ) ) )
      other-factor-list) )
```

APPENDIX J

INDEX OF FUNCTION, FLAVOR, AND GLOBAL VARIABLE DEFINITIONS

FUNCTIONS

FLAVORS

GLOBAL VARIABLES

APPENDIX K

PRE-ESTABLISHED DATA FOR EXERCISE MODE

>COAAT>EXER-CE-DATA-LIST.FILE - 3/3/89

```
((COMMENT-A11 "Seize limited obj's") (OBJECTIVE-A11 "Current LC ")
 (CE-TYPE-A11 "Fix En in Position  ") (COMMENT-B5 "Strongly defended")
 (OBJECTIVE-B5 "Objective WOLF") (CE-TYPE-B5 "Seize Objective     ")
 (COMMENT-B4 "North vic PL MUSTANG") (OBJECTIVE-B4 "by 18 MTR")
 (CE-TYPE-B4 "Defeat Enemy CATK   ") (COMMENT-B3 "Fordable; Opposed")
 (OBJECTIVE-B3 "Haune River") (CE-TYPE-B3 "Cross River         ")
 (COMMENT-B2 "Fordable; Opposed") (OBJECTIVE-B2 "Fulda River")
 (CE-TYPE-B2 "Cross River         ") (COMMENT-B1 "")
 (OBJECTIVE-B1 "PL Appaloosa")
 (CE-TYPE-B1 "Penetrate En 1st Ech") (COMMENT-B11 "Seize limited obj's")
 (OBJECTIVE-B11 "Current LC") (CE-TYPE-B11 "Fix En in Position  ")
 (COMMENT-A4 "Strongly defended") (OBJECTIVE-A4 "Objective FOX ")
 (CE-TYPE-A4 "Seize Objective     ") (COMMENT-A3 "Fordable; Opposed")
 (OBJECTIVE-A3 "Haune River") (CE-TYPE-A3 "Cross River         ")
 (COMMENT-A2 "Unfordable; Opposed") (OBJECTIVE-A2 "Fulda River")
 (CE-TYPE-A2 "Cross River         ") (COMMENT-A1 "")
 (OBJECTIVE-A1 "PL Appaloosa")
 (CE-TYPE-A1 "Penetrate En 1st Ech"))
```

>COAAT>EXER-CE-MENU-LIST-AB.FILE - 3/3/89

```
(("CE-1A1" :VALUE "CE-1A1" :DOCUMENTATION
  " TYPE: Penetrate En 1st Ech   OBJECTIVE: PL Appaloosa   COMMENT: ")
 ("CE-1A2" :VALUE "CE-1A2" :DOCUMENTATION
  " TYPE: Cross River   OBJECTIVE: Fulda River   COMMENT: Unfordable; Opposed")
 ("CE-1A3" :VALUE "CE-1A3" :DOCUMENTATION
  " TYPE: Cross River   OBJECTIVE: Haune River   COMMENT: Fordable; Opposed")
 ("CE-1A4" :VALUE "CE-1A4" :DOCUMENTATION
  " TYPE: Seize Objective   OBJECTIVE: Objective FOX
    COMMENT: Strongly defended")
 ("CE-1B11" :VALUE "CE-1B11" :DOCUMENTATION
  " TYPE: Fix En in Position   OBJECTIVE: Current LC
    COMMENT: Seize limited obj's")
 ("CE-2B1" :VALUE "CE-2B1" :DOCUMENTATION
  " TYPE: Penetrate En 1st Ech   OBJECTIVE: PL Appaloosa   COMMENT: ")
 ("CE-2B2" :VALUE "CE-2B2" :DOCUMENTATION
  " TYPE: Cross River   OBJECTIVE: Fulda River   COMMENT: Fordable; Opposed")
 ("CE-2B3" :VALUE "CE-2B3" :DOCUMENTATION
  " TYPE: Cross River   OBJECTIVE: Haune River   COMMENT: Fordable; Opposed")
 ("CE-2B4" :VALUE "CE-2B4" :DOCUMENTATION
  " TYPE: Defeat Enemy CATK   OBJECTIVE: by 18 MTR
    COMMENT: North vic PL MUSTANG")
 ("CE-2B5" :VALUE "CE-2B5" :DOCUMENTATION
  " TYPE: Seize Objective   OBJECTIVE: Objective WOLF
    COMMENT: Strongly defended")
 ("CE-2A11" :VALUE "CE-2A11" :DOCUMENTATION
  " TYPE: Fix En in Position   OBJECTIVE: Current LC
    COMMENT: Seize limited obj's"))
```

```
>COAAT>EXER-VARIABLE-DATA-LIST-AB.FILE - 3/14/89

((TIME-RQD-2-SCALE "5") (FEBA-MVMT-2-SCALE "1") (RE-AMMO-2-SCALE "5")
 (RE-POL-2-SCALE "5") (EC-EQUIP-2-SCALE "1") (EC-PERS-2-SCALE "1")
 (FC-EQUIP-2-SCALE "5") (FC-PERS-2-SCALE "5") (TIME-RQD-2 "  0.0")
 (FEBA-MVMT-2 "   0 ") (RE-AMMO-2 "   0 ") (RE-POL-2 "   0 ")
 (EC-EQUIP-2 "   0 ") (EC-PERS-2 "   0 ") (FC-EQUIP-2 "   0 ")
 (FC-PERS-2 "   0 ") (TIME-RQD-22 "  0.0") (FEBA-MVMT-22 "   0 ")
 (RE-AMMO-22 "   0 ") (RE-POL-22 "   0 ") (EC-EQUIP-22 "   0 ")
 (EC-PERS-22 "   0 ") (FC-EQUIP-22 "   0 ") (FC-PERS-22 "   0 ")
 (TIME-RQD-21 "  0.0") (FEBA-MVMT-21 "   0 ") (RE-AMMO-21 "   0 ")
 (RE-POL-21 "   0 ") (EC-EQUIP-21 "   0 ") (EC-PERS-21 "   0 ")
 (FC-EQUIP-21 "   0 ") (FC-PERS-21 "   0 ")
 (TIME-RQD-1-SCALE "5") (FEBA-MVMT-1-SCALE "1") (RE-AMMO-1-SCALE "5")
 (RE-POL-1-SCALE "5") (EC-EQUIP-1-SCALE "1") (EC-PERS-1-SCALE "1")
 (FC-EQUIP-1-SCALE "5") (FC-PERS-1-SCALE "5") (TIME-RQD-1 "  0.0")
 (FEBA-MVMT-1 "   0 ") (RE-AMMO-1 "   0 ") (RE-POL-1 "   0 ")
 (EC-EQUIP-1 "   0 ") (EC-PERS-1 "   0 ") (FC-EQUIP-1 "   0 ")
 (FC-PERS-1 "   0 ") (TIME-RQD-12 "  0.0") (FEBA-MVMT-12 "   0 ")
 (RE-AMMO-12 "   0 ") (RE-POL-12 "   0 ") (EC-EQUIP-12 "   0 ")
 (EC-PERS-12 "   0 ") (FC-EQUIP-12 "   0 ") (FC-PERS-12 "   0 ")
 (TIME-RQD-11 "  0.0") (FEBA-MVMT-11 "   0 ") (RE-AMMO-11 "   0 ")
 (RE-POL-11 "   0 ") (EC-EQUIP-11 "   0 ") (EC-PERS-11 "   0 ")
 (FC-EQUIP-11 "   0 ") (FC-PERS-11 "   0 ")
 (MAX-COA 2) (MOD1-DATA-EXISTS T)
 (COMMENT-CE-2A11 "Seize limited obj's") (OBJECTIVE-CE-2A11 "Current LC")
 (CE-TYPE-CE-2A11 "Fix En in Position  ")
 (COA-2-AVENUE-2 "ALPHA")
 (COMMENT-CE-2B5 "Strongly defended") (OBJECTIVE-CE-2B5 "Objective WOLF")
 (CE-TYPE-CE-2B5 "Seize Objective      ")
 (COMMENT-CE-2B4 "North vic PL MUSTANG") (OBJECTIVE-CE-2B4 "by 18 MTR")
 (CE-TYPE-CE-2B4 "Defeat Enemy CATK    ")
 (COMMENT-CE-2B3 "Fordable; Opposed") (OBJECTIVE-CE-2B3 "Haune River")
 (CE-TYPE-CE-2B3 "Cross River          ")
 (COMMENT-CE-2B2 "Fordable; Opposed") (OBJECTIVE-CE-2B2 "Fulda River")
 (CE-TYPE-CE-2B2 "Cross River          ")
 (COMMENT-CE-2B1 " ") (OBJECTIVE-CE-2B1 "PL Appaloosa")
 (CE-TYPE-CE-2B1 "Penetrate En 1st Ech")
 (COA-2-AVENUE-1 "BRAVO") (MAIN-ATK-AXIS-COA-2 "B")
 (COMMENT-CE-1B11 "Seize limited obj's") (OBJECTIVE-CE-1B11 "Current LC")
 (CE-TYPE-CE-1B11 "Fix En in Position  ")
 (COA-1-AVENUE-2 "BRAVO")
 (COMMENT-CE-1A4 "Strongly defended") (OBJECTIVE-CE-1A4 "Objective FOX")
 (CE-TYPE-CE-1A4 "Seize Objective      ")
 (COMMENT-CE-1A3 "Fordable; Opposed") (OBJECTIVE-CE-1A3 "Haune River")
 (CE-TYPE-CE-1A3 "Cross River          ")
 (COMMENT-CE-1A2 "Unfordable; Opposed") (OBJECTIVE-CE-1A2 "Fulda River")
 (CE-TYPE-CE-1A2 "Cross River          ")
 (COMMENT-CE-1A1 "")  (OBJECTIVE-CE-1A1 "PL Appaloosa")
 (CE-TYPE-CE-1A1 "Penetrate En 1st Ech")
 (CE-NUMBER "CE-2A11") (COA-1-AVENUE-1 "ALPHA") (MAIN-ATK-AXIS-COA-1 "A")
 (*COA* "COA-2") (MOA-NUMBER 0) (COA-NUMBER 2) (*MOA-TYPE* AVENUE)
 (*MISSION* OFFENSE))
```

```
>COAAT>EXER-MOD2-SUMMARY-AB.SCRN - 10/10/89

((CENTER "WARGAMING SUMMARY SHEET") (NEWLINE 1) (FFIELD 14 " ")
 (FFIELD 21 "   CE Type") (FFIELD 20 "   Objective")
 (TFIELD 14 "   Friendly%  Casualties% Pers  Equip")
 (TFIELD 14 "    Enemy  %  Casualties% Pers  Equip")
 (TFIELD 12 "  Percent%  Expended%  POL  Ammo")
 (TFIELD 8 "  FEBA%  Mvmt%  (km)")
 (TFIELD 8 "  Time%  Rqd%  (hrs)") (NEWLINE 1)
 (MSETS OUTER-ONE
        ""
        (LEFT "COA-1  Main Attack Axis = A")
        (MSETS INNER-ONE
               ""
               (LEFT "  AVENUE-ALPHA")
               (MROW ONE
                     ""
                     (FFIELD 6 " ") (FFIELD 8 "CE-1
                     (FFIELD 21 "Penetrate En 1st Echn")
                     (FFIELD 20 "PL Appaloosa")
                     (DFIELD 6 "   0  " FC-PERS-CE-1A1) (FFIELD 1 " ")
                     (DFIELD 6 "   0  " FC-EQUIP-CE-1A1) (FFIELD 1 " ")
                     (DFIELD 6 "   0  " EC-PERS-CE-1A1) (FFIELD 1 " ")
                     (DFIELD 6 "   0  " EC-EQUIP-CE-1A1) (FFIELD 1 " ")
                     (DFIELD 6 "   0  " RE-POL-CE-1A1)
                     (DFIELD 6 "   0  " RE-AMMO-CE-1A1) (FFIELD 1 " ")
                     (DFIELD 6 "   0  " FEBA-MVMT-CE-1A1) (FFIELD 2 " ")
                     (DFIELD 6 "  0.0" TIME-RQD-CE-1A1)
                     (NEWLINE 1))
               (MROW TWO
                     ""
                     (FFIELD 6 " ") (FFIELD 8 "CE-1A2")
                     (FFIELD 21 "Cross River         ")
                     (FFIELD 20 "Fulda River")
                     (DFIELD 6 "   0  " FC-PERS-CE-1A2) (FFIELD 1 " ")
                     (DFIELD 6 "   0  " FC-EQUIP-CE-1A2) (FFIELD 1 " ")
                     (DFIELD 6 "   0  " EC-PERS-CE-1A2) (FFIELD 1 " ")
                     (DFIELD 6 "   0  " EC-EQUIP-CE-1A2) (FFIELD 1 " ")
                     (DFIELD 6 "   0  " RE-POL-CE-1A2)
                     (DFIELD 6 "   0  " RE-AMMO-CE-1A2) (FFIELD 1 " ")
                     (DFIELD 6 "   0  " FEBA-MVMT-CE-1A2) (FFIELD 2 " ")
                     (DFIELD 6 "  0.0" TIME-RQD-CE-1A2)
                     (NEWLINE 1))
               (MROW THREE
                     ""
                     (FFIELD 6 " ") (FFIELD 8 "CE-1A3")
                     (FFIELD 21 "Cross River         ")
                     (FFIELD 20 "Hauna River")
                     (DFIELD 6 "   0  " FC-PERS-CE-1A3) (FFIELD 1 " ")
                     (DFIELD 6 "   0  " FC-EQUIP-CE-1A3) (FFIELD 1 " ")
                     (DFIELD 6 "   0  " EC-PERS-CE-1A3) (FFIELD 1 " ")
                     (DFIELD 6 "   0  " EC-EQUIP-CE-1A3) (FFIELD 1 " ")
                     (DFIELD 6 "   0  " RE-POL-CE-1A3)
                     (DFIELD 6 "   0  " RE-AMMO-CE-1A3) (FFIELD 1 " ")
                     (DFIELD 6 "   0  " FEBA-MVMT-CE-1A3) (FFIELD 2 " ")
                     (DFIELD 6 "  0.0" TIME-RQD-CE-1A3)
```

```
                (NEWLINE 1))
          (MROW FOUR
                ""
                (FFIELD 6 " ") (FFIELD 8 "CE-1A4")
                (FFIELD 21 "Seize Objective     ")
                (FFIELD 20 "Objective FOX")
                (DFIELD 6 "   0  " FC-PERS-CE-1A4) (FFIELD 1 " ")
                (DFIELD 6 "   0  " FC-EQUIP-CE-1A4) (FFIELD 1 " ")
                (DFIELD 6 "   0  " EC-PERS-CE-1A4) (FFIELD 1 " ")
                (DFIELD 6 "   0  " EC-EQUIP-CE-1A4) (FFIELD 1 " ")
                (DFIELD 6 "   0  " RE-POL-CE-1A4)
                (DFIELD 6 "   0  " RE-AMMO-CE-1A4) (FFIELD 1 " ")
                (DFIELD 6 "   0  " FEBA-MVMT-CE-1A4) (FFIELD 2 " ")
                (DFIELD 6 "  0.0" TIME-RQD-CE-1A4)
                (NEWLINE 1))
          (MROW FIVE
                ""
                (FFIELD 38 " ") (FFIELD 15 "AVENUE TOTAL") (FFIELD 2 " ")
                (DEMON 6 FC-PERS-11 INT-TOTAL FC-PERS-CE-1A1
                         FC-PERS-CE-1A2 FC-PERS-CE-1A3 FC-PERS-CE-1A4)
                (FFIELD 1 " ")
                (DEMON 6 FC-EQUIP-11 INT-TOTAL FC-EQUIP-CE-1A1
                         FC-EQUIP-CE-1A2 FC-EQUIP-CE-1A3 FC-EQUIP-CE-1A4)
                (FFIELD 1 " ")
                (DEMON 6 EC-PERS-11 INT-TOTAL EC-PERS-CE-1A1
                         EC-PERS-CE-1A2 EC-PERS-CE-1A3 EC-PERS-CE-1A4)
                (FFIELD 1 " ")
                (DEMON 6 EC-EQUIP-11 INT-TOTAL EC-EQUIP-CE-1A1
                         EC-EQUIP-CE-1A2 EC-EQUIP-CE-1A3 EC-EQUIP-CE-1A4)
                (FFIELD 1 " ")
                (DEMON 6 RE-POL-11 INT-TOTAL RE-POL-CE-1A1
                         RE-POL-CE-1A2 RE-POL-CE-1A3 RE-POL-CE-1A4)
                (DEMON 6 RE-AMMO-11 INT-TOTAL RE-AMMO-CE-1A1
                         RE-AMMO-CE-1A2 RE-AMMO-CE-1A3 RE-AMMO-CE-1A4)
                (FFIELD 1 " ")
                (DEMON 6 FEBA-MVMT-11 INT-TOTAL FEBA-MVMT-CE-1A1
                        FEBA-MVMT-CE-1A2 FEBA-MVMT-CE-1A3 FEBA-MVMT-CE-1A4)
                (FFIELD 2 " ")
                (DEMON 6 TIME-RQD-11 TOTAL TIME-RQD-CE-1A1
                         TIME-RQD-CE-1A2 TIME-RQD-CE-1A3 TIME-RQD-CE-1A4)
                (NEWLINE 1)))
    (MSETS INNER-TWO
          ""
          (LEFT "  AVENUE-BRAVO")
          (MROW ONE
                ""
                (FFIELD 6 " ") (FFIELD 8 "CE-1B11")
                (FFIELD 21 "Fix En in Position  ")
                (FFIELD 20 "Current LC")
                (DFIELD 6 "   0  " FC-PERS-CE-1B11) (FFIELD 1 " ")
                (DFIELD 6 "   0  " FC-EQUIP-CE-1B11) (FFIELD 1 " ")
                (DFIELD 6 "   0  " EC-PERS-CE-1B11) (FFIELD 1 " ")
                (DFIELD 6 "   0  " EC-EQUIP-CE-1B11) (FFIELD 1 " ")
                (DFIELD 6 "   0  " RE-POL-CE-1B11)
```

```
                    (DFIELD 6 "    0  " RE-AMMO-CE-1B11) (FFIELD 1 " ")
                    (DFIELD 6 "    0  " FEBA-MVMT-CE-1B11) (FFIELD 2 " ")
                    (DFIELD 6 "  0.0" TIME-RQD-CE-1B11)
                    (NEWLINE 1))
              (MROW TWO
                    ""
                    (FFIELD 38 " ") (FFIELD 15 "AVENUE TOTAL") (FFIELD 2 " ")
                    (DEMON 6 FC-PERS-12 INT-TOTAL FC-PERS-CE-1B11)
       (FFIELD 1 " ")
                    (DEMON 6 FC-EQUIP-12 INT-TOTAL FC-EQUIP-CE-1B11)
                    (FFIELD 1 " ")
                    (DEMON 6 EC-PERS-12 INT-TOTAL EC-PERS-CE-1B11)
                    (FFIELD 1 " ")
                    (DEMON 6 EC-EQUIP-12 INT-TOTAL EC-EQUIP-CE-1B11)
                    (FFIELD 1 " ")
                    (DEMON 6 RE-POL-12 INT-TOTAL RE-POL-CE-1B11)
                    (DEMON 6 RE-AMMO-12 INT-TOTAL RE-AMMO-CE-1B11)
                    (FFIELD 1 " ")
                    (DEMON 6 FEBA-MVMT-12 INT-TOTAL) (FFIELD 2 " ")
                    (DEMON 6 TIME-RQD-12 TOTAL)
                    (NEWLINE 1)))
        (MSETS INNER-THREE
               ""
               (FFIELD 38 " ") (FFIELD 15 "COA TOTAL ") (FFIELD 2 " ")
               (DEMON 6 FC-PERS-1 INT-TOTAL FC-PERS-CE-1A1 FC-PERS-CE-1A2
                        FC-PERS-CE-1A3 FC-PERS-CE-1A4 FC-PERS-CE-1B11)
               (FFIELD 1 " ")
               (DEMON 6 FC-EQUIP-1 INT-TOTAL FC-EQUIP-CE-1A1 FC-EQUIP-CE-1A2
                        FC-EQUIP-CE-1A3 FC-EQUIP-CE-1A4 FC-EQUIP-CE-1B11)
               (FFIELD 1 " ")
               (DEMON 6 EC-PERS-1 INT-TOTAL EC-PERS-CE-1A1 EC-PERS-CE-1A2
                        EC-PERS-CE-1A3 EC-PERS-CE-1A4 EC-PERS-CE-1B11)
               (FFIELD 1 " ")
               (DEMON 6 EC-EQUIP-1 INT-TOTAL EC-EQUIP-CE-1A1 EC-EQUIP-CE-1A2
                        EC-EQUIP-CE-1A3 EC-EQUIP-CE-1A4 EC-EQUIP-CE-1B11)
               (FFIELD 1 " ")
               (DEMON 6 RE-POL-1 INT-TOTAL RE-POL-CE-1A1 RE-POL-CE-1A2
                        RE-POL-CE-1A3 RE-POL-CE-1A4 RE-POL-CE-1B11)
               (DEMON 6 RE-AMMO-1 INT-TOTAL RE-AMMO-CE-1A1 RE-AMMO-CE-1A2
                        RE-AMMO-CE-1A3 RE-AMMO-CE-1A4 RE-AMMO-CE-1B11)
               (FFIELD 1 " ")
               (DEMON 6 FEBA-MVMT-1 INT-TOTAL FEBA-MVMT-CE-1A1 FEBA-MVMT-CE-1A2
                        FEBA-MVMT-CE-1A3 FEBA-MVMT-CE-1A4)
               (FFIELD 2 " ")
               (DEMON 6 TIME-RQD-1 TOTAL TIME-RQD-CE-1A1 TIME-RQD-CE-1A2
                        TIME-RQD-CE-1A3 TIME-RQD-CE-1A4)
               (NEWLINE 1)
               (FFIELD 38 " ") (FFIELD 15 "SCALED VALUE") (FFIELD 4 " ")
               (DEMON 2 FC-PERS-1-SCALE CALC-FC-PERS-SCALE-VAL FC-PERS-1)
               (FFIELD 5 " ")
               (DEMON 2 FC-EQUIP-1-SCALE CALC-FC-EQUIP-SCALE-VAL FC-EQUIP-1)
               (FFIELD 5 " ")
               (DEMON 2 EC-PERS-1-SCALE CALC-EC-PERS-SCALE-VAL EC-PERS-1)
               (FFIELD 5 " ")
```

```
                (DEMON 2 EC-EQUIP-1-SCALE CALC-EC-EQUIP-SCALE-VAL EC-EQUIP-1)
                (FFIELD 5 " ")
                (DEMON 2 RE-POL-1-SCALE CALC-RE-POL-SCALE-VAL RE-POL-1)
                (FFIELD 4 " ")
                (DEMON 2 RE-AMMO-1-SCALE CALC-RE-AMMO-SCALE-VAL RE-AMMO-1)
                (FFIELD 5 " ")
                (DEMON 2 FEBA-MVMT-1-SCALE CALC-FEBA-MVMT-SCALE-VAL FEBA-MVMT-1)
                (FFIELD 7 " ")
                (DEMON 2 TIME-RQD-1-SCALE CALC-TIME-RQD-SCALE-VAL TIME-RQD-1)
                (NEWLINE 1)))
 (MSETS OUTER-TWO
        ""
        (LEFT "COA-2  Main Attack Axis = B")
        (MSETS INNER-ONE
                ""
                (LEFT "  AVENUE-BRAVO")
                (MROW ONE
                      ""
                      (FFIELD 6 " ") (FFIELD 8 "CE-2B1")
                      (FFIELD 21 "Penetrate En 1st Ech")
                      (FFIELD 20 "FL Appaloosa")
                      (DFIELD 6 "   0  " FC-PERS-CE-2B1) (FFIELD 1 " ")
                      (DFIELD 6 "   0  " FC-EQUIP-CE-2B1) (FFIELD 1 " ")
                      (DFIELD 6 "   0  " EC-PERS-CE-2B1) (FFIELD 1 " ")
                      (DFIELD 6 "   0  " EC-EQUIP-CE-2B1) (FFIELD 1 " ")
                      (DFIELD 6 "   0  " RE-POL-CE-2B1)
                      (DFIELD 6 "   0  " RE-AMMO-CE-2B1) (FFIELD 1 " ")
                      (DFIELD 6 "   0  " FEBA-MVMT-CE-2B1) (FFIELD 2 " ")
                      (DFIELD 6 "  0.0" TIME-RQD-CE-2B1)
                      (NEWLINE 1))
                (MROW TWO
                      ""
                      (FFIELD 6 " ") (FFIELD 8 "CE-2B2")
                      (FFIELD 21 "Cross River          ")
                      (FFIELD 20 "Fulda River")
                      (DFIELD 6 "   0  " FC-PERS-CE-2B2) (FFIELD 1 " ")
                      (DFIELD 6 "   0  " FC-EQUIP-CE-2B2) (FFIELD 1 " ")
                      (DFIELD 6 "   0  " EC-PERS-CE-2B2) (FFIELD 1 " ")
                      (DFIELD 6 "   0  " EC-EQUIP-CE-2B2) (FFIELD 1 " ")
                      (DFIELD 6 "   0  " RE-POL-CE-2B2)
                      (DFIELD 6 "   0  " RE-AMMO-CE-2B2) (FFIELD 1 " ")
                      (DFIELD 6 "   0  " FEBA-MVMT-CE-2B2) (FFIELD 2 " ")
                      (DFIELD 6 "  0.0" TIME-RQD-CE-2B2)
                      (NEWLINE 1))
                (MROW THREE
                      ""
                      (FFIELD 6 " ") (FFIELD 8 "CE-2B3")
                      (FFIELD 21 "Cross River          ")
                      (FFIELD 20 "Haune River")
                      (DFIELD 6 "   0  " FC-PERS-CE-2B3) (FFIELD 1 " ")
                      (DFIELD 6 "   0  " FC-EQUIP-CE-2B3) (FFIELD 1 " ")
                      (DFIELD 6 "   0  " EC-PERS-CE-2B3) (FFIELD 1 " ")
                      (DFIELD 6 "   0  " EC-EQUIP-CE-2B3) (FFIELD 1 " ")
                      (DFIELD 6 "   0  " RE-POL-CE-2B3)
```

```
        (DFIELD 6 "    0  " RE-AMMO-CE-2B3) (FFIELD 1 " ")
        (DFIELD 6 "    0  " FEBA-MVMT-CE-2B3) (FFIELD 2 " ")
        (DFIELD 6 "  0.0" TIME-RQD-CE-2B3)
        (NEWLINE 1))
(MROW FOUR
        ""
        (FFIELD 6 " ") (FFIELD 8 "CE-2B4")
        (FFIELD 21 "Defeat Enemy CATK    ")
        (FFIELD 20 "by 18 MTR")
        (DFIELD 6 "    0  " FC-PERS-CE-2B4) (FFIELD 1 " ")
        (DFIELD 6 "    0  " FC-EQUIP-CE-2B4) (FFIELD 1 " ")
        (DFIELD 6 "    0  " EC-PERS-CE-2B4) (FFIELD 1 " ")
        (DFIELD 6 "    0  " EC-EQUIP-CE-2B4) (FFIELD 1 " ")
        (DFIELD 6 "    0  " RE-POL-CE-2B4)
        (DFIELD 6 "    0  " RE-AMMO-CE-2B4) (FFIELD 1 " ")
        (DFIELD 6 "    0  " FEBA-MVMT-CE-2B4) (FFIELD 2 " ")
        (DFIELD 6 "  0.0" TIME-RQD-CE-2B4)
        (NEWLINE 1))
(MROW FIVE
        ""
        (FFIELD 6 " ") (FFIELD 8 "CE-2B5")
        (FFIELD 21 "Seize Objective      ")
        (FFIELD 20 "objective WOLF")
        (DFIELD 6 "    0  " FC-PERS-CE-2B5) (FFIELD 1 " ")
        (DFIELD 6 "    0  " FC-EQUIP-CE-2B5) (FFIELD 1 " ")
        (DFIELD 6 "    0  " EC-PERS-CE-2B5) (FFIELD 1 " ")
        (DFIELD 6 "    0  " EC-EQUIP-CE-2B5) (FFIELD 1 " ")
        (DFIELD 6 "    0  " RE-POL-CE-2B5)
        (DFIELD 6 "    0  " RE-AMMO-CE-2B5) (FFIELD 1 " ")
        (DFIELD 6 "    0  " FEBA-MVMT-CE-2B5) (FFIELD 2 " ")
        (DFIELD 6 "  0.0" TIME-RQD-CE-2B5)
        (NEWLINE 1))
(MROW SIX
        ""
        (FFIELD 38 " ") (FFIELD 15 "AVENUE TOTAL") (FFIELD 2 " ")
        (DEMON 6 FC-PERS-21 INT-TOTAL FC-PERS-CE-2B1 FC-PERS-CE-2B2
                 FC-PERS-CE-2B3 FC-PERS-CE-2B4 FC-PERS-CE-2B5)
        (FFIELD 1 " ")
        (DEMON 6 FC-EQUIP-21 INT-TOTAL FC-EQUIP-CE-2B1
                 FC-EQUIP-CE-2B2 FC-EQUIP-CE-2B3 FC-EQUIP-CE-2B4
                 FC-EQUIP-CE-2B5)  (FFIELD 1 " ")
        (DEMON 6 EC-PERS-21 INT-TOTAL EC-PERS-CE-2B1 EC-PERS-CE-2B2
                 EC-PERS-CE-2B3 EC-PERS-CE-2B4 EC-PERS-CE-2B5)
        (FFIELD 1 " ")
        (DEMON 6 EC-EQUIP-21 INT-TOTAL EC-EQUIP-CE-2B1
                 EC-EQUIP-CE-2B2 EC-EQUIP-CE-2B3 EC-EQUIP-CE-2B4
                 EC-EQUIP-CE-2B5)  (FFIELD 1 " ")
        (DEMON 6 RE-POL-21 INT-TOTAL RE-POL-CE-2B1 RE-POL-CE-2B2
                 RE-POL-CE-2B3 RE-POL-CE-2B4 RE-POL-CE-2B5)
        (DEMON 6 RE-AMMO-21 INT-TOTAL RE-AMMO-CE-2B1 RE-AMMO-CE-2B2
                 RE-AMMO-CE-2B3 RE-AMMO-CE-2B4 RE-AMMO-CE-2B5)
        (FFIELD 1 " ")
        (DEMON 6 FEBA-MVMT-21 INT-TOTAL FEBA-MVMT-CE-2B1
                 FEBA-MVMT-CE-2B2FEBA-MVMT-CE-2B3FEBA-MVMT-CE-2B4
```

```
                        FEBA-MVMT-CE-2B5)  (FFIELD 2 " ")
                (DEMON 6 TIME-RQD-21 TOTAL TIME-RQD-CE-2B1 TIME-RQD-CE-2B2
                        TIME-RQD-CE-2B3 TIME-RQD-CE-2B4 TIME-RQD-CE-2B5)
                (NEWLINE 1)))
        (MSETS INNER-TWO
                ""
                (LEFT "  AVENUE-ALPHA")
                (MROW ONE
                        ""
                        (FFIELD 6 " ") (FFIELD 8 "CE-2A11")
                        (FFIELD 21 "Fix En in Position  ")
                        (FFIELD 20 "Current LC")
                        (DFIELD 6 "    0  " FC-PERS-CE-2A11) (FFIELD 1 " ")
                        (DFIELD 6 "    0  " FC-EQUIP-CE-2A11) (FFIELD 1 " ")
                        (DFIELD 6 "    0  " EC-PERS-CE-2A11) (FFIELD 1 " ")
                        (DFIELD 6 "    0  " EC-EQUIP-CE-2A11) (FFIELD 1 " ")
                        (DFIELD 6 "    0  " RE-POL-CE-2A11)
                        (DFIELD 6 "    0  " RE-AMMO-CE-2A11) (FFIELD 1 " ")
                        (DFIELD 6 "    0  " FEBA-MVMT-CE-2A11) (FFIELD 2 " ")
                        (DFIELD 6 "  0.0" TIME-RQD-CE-2A11)
                        (NEWLINE 1))
                (MROW TWO
                        ""
                        (FFIELD 38 " ") (FFIELD 15 "AVENUE TOTAL") (FFIELD 2 " ")
                        (DEMON 6 FC-PERS-22 INT-TOTAL FC-PERS-CE-2A11)
                        (FFIELD 1 " ")
                        (DEMON 6 FC-EQUIP-22 INT-TOTAL FC-EQUIP-CE-2A11)
                        (FFIELD 1 " ")
                        (DEMON 6 EC-PERS-22 INT-TOTAL EC-PERS-CE-2A11)
                        (FFIELD 1 " ")
                        (DEMON 6 EC-EQUIP-22 INT-TOTAL EC-EQUIP-CE-2A11)
                        (FFIELD 1 " ")
                        (DEMON 6 RE-POL-22 INT-TOTAL RE-POL-CE-2A11)
                        (DEMON 6 RE-AMMO-22 INT-TOTAL RE-AMMO-CE-2A11)
                        (FFIELD 1 " ")
                        (DEMON 6 FEBA-MVMT-22 INT-TOTAL) (FFIELD 2 " ")
                        (DEMON 6 TIME-RQD-22 TOTAL)
                        (NEWLINE 1)))
        (MSETS INNER-THREE
                ""
                (FFIELD 38 " ") (FFIELD 15 "COA TOTAL ") (FFIELD 2 " ")
                (DEMON 6 FC-PERS-2 INT-TOTAL FC-PERS-CE-2B1 FC-PERS-CE-2B2
                        FC-PERS-CE-2B3 FC-PERS-CE-2B4 FC-PERS-CE-2B5
                        FC-PERS-CE-2A11)  (FFIELD 1 " ")
                (DEMON 6 FC-EQUIP-2 INT-TOTAL FC-EQUIP-CE-2B1 FC-EQUIP-CE-2B2
                        FC-EQUIP-CE-2B3 FC-EQUIP-CE-2B4 FC-EQUIP-CE-2B5
                        FC-EQUIP-CE-2A11)  (FFIELD 1 " ")
                (DEMON 6 EC-PERS-2 INT-TOTAL EC-PERS-CE-2B1 EC-PERS-CE-2B2
                        EC-PERS-CE-2B3 EC-PERS-CE-2B4 EC-PERS-CE-2B5
                        EC-PERS-CE-2A11)  (FFIELD 1 " ")
                (DEMON 6 EC-EQUIP-2 INT-TOTAL EC-EQUIP-CE-2B1 EC-EQUIP-CE-2B2
                        EC-EQUIP-CE-2B3 EC-EQUIP-CE-2B4 EC-EQUIP-CE-2B5
                        EC-EQUIP-CE-2A11)  (FFIELD 1 " ")
                (DEMON 6 RE-POL-2 INT-TOTAL RE-POL-CE-2B1 RE-POL-CE-2B2
```

```
          RE-POL-CE-2B3 RE-POL-CE-2B4 RE-POL-CE-2B5
          RE-POL-CE-2A11)
(DEMON 6 RE-AMMO-2 INT-TOTAL RE-AMMO-CE-2B1 RE-AMMO-CE-2B2
          RE-AMMO-CE-2B3 RE-AMMO-CE-2B4 RE-AMMO-CE-2B5
          RE-AMMO-CE-2A11)  (FFIELD 1 " ")
(DEMON 6 FEBA-MVMT-2 INT-TOTAL FEBA-MVMT-CE-2B1
          FEBA-MVMT-CE-2B2 FEBA-MVMT-CE-2B3 FEBA-MVMT-CE-2B4
          FEBA-MVMT-CE-2B5)  (FFIELD 2 " ")
(DEMON 6 TIME-RQD-2 TOTAL TIME-RQD-CE-2B1 TIME-RQD-CE-2B2
          TIME-RQD-CE-2B3 TIME-RQD-CE-2B4 TIME-RQD-CE-2B5)
(NEWLINE 1)
(FFIELD 38 " ") (FFIELD 15 "SCALED VALUE") (FFIELD 4 " ")
(DEMON 2 FC-PERS-2-SCALE CALC-FC-PERS-SCALE-VAL FC-PERS-2)
(FFIELD 5 " ")
(DEMON 2 FC-EQUIP-2-SCALE CALC-FC-EQUIP-SCALE-VAL FC-EQUIP-2)
(FFIELD 5 " ")
(DEMON 2 EC-PERS-2-SCALE CALC-EC-PERS-SCALE-VAL EC-PERS-2)
(FFIELD 5 " ")
(DEMON 2 EC-EQUIP-2-SCALE CALC-EC-EQUIP-SCALE-VAL EC-EQUIP-2)
(FFIELD 5 " ")
(DEMON 2 RE-POL-2-SCALE CALC-RE-POL-SCALE-VAL RE-POL-2)
(FFIELD 4 " ")
(DEMON 2 RE-AMMO-2-SCALE CALC-RE-AMMO-SCALE-VAL RE-AMMO-2)
(FFIELD 5 " ")
(DEMON 2 FEBA-MVMT-2-SCALE CALC-FEBA-MVMT-SCALE-VAL FEBA-MVMT-2)
(FFIELD 7 " ")
(DEMON 2 TIME-RQD-2-SCALE CALC-TIME-RQD-SCALE-VAL TIME-RQD-2)
(NEWLINE 1))))
```