10/3-26-90 gs (1)

SANDIA REPORT

SAND89—0926/2 • UC—~~545~~ 700

Unlimited Release

Printed December 1989

SAVI:
Systematic Analysis of Vulnerability to Intrusion
Volume 2 of 2

DO NOT MICROFILM COVER

Nuclear Security Systems Directorate
and
Science & Engineering Associates, Inc.

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550
for the United States Department of Energy
under Contract DE-AC04-76DP00789

SF 2900Q(8-81)

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
Office of Scientific and Technical Information
PO Box 62
Oak Ridge, TN 37831

Prices available from (615) 576-8401, FTS 626-8401

Available to the public from
National Technical Information Service
US Department of Commerce
5285 Port Royal Rd
Springfield, VA 22161

NTIS price codes
Printed copy: A21
Microfiche copy: A01

SAND--89-0926/2

DE90 008160

SAND89-0926/2
Unlimited Release
Printed December 1989

SAVI: SYSTEMATIC ANALYSIS OF VULNERABILITY TO INTRUSION
VOLUME 2 OF 2

Nuclear Security Systems Directorate
Sandia National Laboratories
Albuquerque, New Mexico 87185

Science & Engineering Associates
Albuquerque, New Mexico 87110

ABSTRACT

SAVI (Systematic Analysis of Vulnerability to Intrusion) is a PC-based software package for modeling and analyzing physical protection systems. This report is the complete documentation of the SAVI project.

MASTER

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

CONTENTS

VOLUME 1

Introduction
Section A -- SAVI Vulnerability Assessment Method
Section B -- SAVI Software Manual
Section C -- SAVI Component Lists
Section D -- SAVI Protection Element Diagrams
Section E -- SAVI Algorithm
Section F -- SAVI Element Scenario Equations
Section G -- SAVI Component Assignments and Component Performance Values for Protection Elements
Section H -- SAVI Data File Structure

VOLUME 2

Introduction
Section I -- SAVI Source Code

INTRODUCTION

SAVI (Systematic Analysis of Vulnerability to Intrusion) is a PC-based software package for modeling and analyzing physical protection systems. SAVI implements several features that make it a unique product. First, the user interface for site modeling and data entry is simple and flexible. Second, the SAVI model analyzes all adversary paths to the target location and, if selected, all exit paths from the target location. Third, a reference catalog and database are included that define the protection elements and safeguards components, and give detection and delay performance values for the components. Finally, SAVI's results are output in graphic form and include recommendations for upgrade.

This report is the complete documentation of the SAVI software package. Volume 1 of this report consists of Sections A through H, and Volume 2 consists of Section I. A brief description of all sections is provided below.

Section A, *SAVI Vulnerability Assessment Method*, describes the basic features of SAVI, defines its concepts and terminology, and gives an overview of the method used to define a site protection system. This section describes the type of threats that can be specified and presents concepts that the user should consider in selecting a threat. The section also discusses the selection of a Response Force Time and outlines the path algorithm that calculates the Probability of Interruption for each adversary path in the facility.

Section B, *SAVI Software Manual*, is the user's manual for SAVI and explains how to install and operate the SAVI software. The manual also includes a step-by-step tutorial session.

Section C, *SAVI Components Lists*, is a hardcopy of the components lists that are associated with each protection element. These lists appear on the computer screen as the component choice lists for individual protection elements. The hardcopy lists are helpful in gathering and documenting security information at the site for later input to the SAVI program.

Section D, *SAVI Protection Element Diagrams*, provides the graphic layouts for each protection element. The layouts show the locations of the detection and delay components that can be selected for a protection element. The Generic Adversary Sequence Diagram is also provided in this section.

Section E, *SAVI Algorithm*, describes the interaction of the detection, delay, and response parts of the security system involved in calculating the Probability of Interruption for each path. This section details the exhaustive tree traversal algorithm implemented in the Probability of Interruption calculation.

Section F, *SAVI Element Scenario Equations*, identifies the boolean equations that group the detection and delay components associated with each defeat scenario. Each equation describes a force or deceit scenario that the adversary may choose in traversing each element along a path.

Section G, *SAVI Component Assignments and Component Performance Values for Protection Elements*, specifies the detection and delay components that are associated with the protection elements and provides their locations. The performance values associated with each component are listed at the end of this section.

Section H, *SAVI Data File Structure*, specifies the format of the Physical Protection System (PPS) data files, which are used by the SAVI program.

Section I, *SAVI Source Code*, is the documented listing of the source code for the SAVI program.

SECTION I

SAVI Source Code

```
/* ************************** SAVI.C source file ************************** */

/* ------------------------------------------------------------------------ */
/*                                                                          */
/*                                  SAVI                                    */
/*          (Systematic Analysis of Vulnerability to Intrusion)             */
/*                                                                          */
/*                              Version 2.2E                                */
/*                                                                          */
/*                               31 JULY 87                                 */
/*                                                                          */
/*               Developed by Sandia National Laboratories                  */
/*               and Science & Engineering Associates, Inc.                 */
/*             under contract to the U.S. Department of Energy              */
/*                                                                          */
/*             Project Leader          Richard McAniff (Sandia)             */
/*                                                                          */
/*             Software Developers     Brad Key (SEA)                       */
/*                                     Scott Walker (SEA)                   */
/*                                     Malcolm Moore (SEA)                  */
/*                                                                          */
/*             Design Assistance       Bret Simpkins (SEA)                  */
/*                                                                          */
/*             Model Design            Al Winblad (Sandia)                  */
/*                                     John Darby (SEA)                     */
/*                                                                          */
/*             Software Testing        Bill Paulus (Sandia)                 */
/*                                                                          */
/* SAVI is a powerful, easy-to-use software package for evaluating the      */
/* effectiveness of physical protection systems against outsider threats.   */
/* The software utilizes the Adversary Sequence Diagram (ASD) to represent  */
/* physical protection systems and model the most vulnerable paths through  */
/* the system. A highly interactive user interface supports a broad range   */
/* of users, allowing the accurate modeling of complex facilities including */
/* display of vulnerability results and improvement recommendations.        */
/*                                                                          */
/* Refer to the SAVI manual for details on software operation.              */
/*                                                                          */
/* Note: This software utilizes portions of Windows for Data from Vermont   */
/* Creative Software and contains licensed program materials of Metagraphics */
/* Software Corporation Copyright (C) 1985, 1986, Metagraphics Software     */
/* Corporation, Scotts Valley, CA 95066                                     */
/*                                                                          */
/* ------------------------------------------------------------------------ */

#include <wfd.h>
#include <wfd_glob.h>
#include <savidefs.h>

/* ------------------------------------------------------------------------ */

main()
```

```
{
   savi_init();
   savi_menu();
   savi_stop();
}

/* ************************ end SAVI.C source file ************************** */

/* ************************ SAVIANAL.C source file ************************** */

#include <wfd.h>
#include <savidefs.h>

DFORMPTR analyze_menu;
DFORMPTR analyze_results_sub_menu;
DFORMPTR analyze_print_sub_menu;

/* current_rft_num contains the number of the current rft in the */
/* rft table                                                     */
int current_rft_num;

/* current_path is used to maintain the current path settings between uses */
/* of the analyze functions (vulnerability and recommendations)            */
int current_path;

/* current_ASD_saved is used to indicate whether the current ASD has been */
/* saved since the last changes were made                                 */
int current_ASD_saved;

/* prev_input_segment_num and prev_output_segment_num hold the input and */
/* output segment numbers from the previous call to insert_link          */
int prev_input_segment_num, prev_output_segment_num;

#define TURBO_ESCAPED (-2)

/* ------------------------------------------------------------------------ */

savi_analyze()

{
   int error_check, start_area, end_area;

   /* calculate path vulnerabilities if the current data is not */
   /* correct for the current ASD                               */
   if (get_analysis_state() == FALSE)
   {
      /* ensure that all components have been set */
      if (!all_comp_choices_made())
      {
         popup_analysis_error_window("All Components Are NOT Defined");
         user_acknowledge_wait();
         remove_analysis_error_window();
         return(0);
      }
```

```
      /* ensure that all PEs are reachable */
      if (unreachable_PEs_present(&start_area,&end_area))
      {
         popup_stranded_PE_window(start_area,end_area);
         user_acknowledge_wait();
         remove_stranded_PE_window();
         return(0);
      }

      popup_turbo_progress_window();

      error_check = go_turbo();

      remove_turbo_progress_window();

      if (error_check == OUT_OF_MEMORY)
      {
         popup_analysis_error_window("Insufficient Memory");
         user_acknowledge_wait();
         remove_analysis_error_window();
         return(0);
      }
      else if (error_check == TURBO_ESCAPED)
      {
         return(0);
      }

      set_vulnerability_current(TRUE);

      /* set the current path to the most vulnerable */
      set_current_path(0);

      /* set the default rft num to the maximum rft */
      set_current_rft_num(get_num_rfts()-1);

      /* signal user that analysis is complete */
      complete_tone();
   }

   /* dump the keyboard buffer before  */
   /* beginning menu handling function */
   dump_key_buffer();

   rd_menu(analyze_menu);

   dump_key_buffer();

   return(0);
}

/* ------------------------------------------------------------------------ */

/* go_turbo sets up the data for the call to turbo_SEApath and calls turbo   */
```

```
/* with segment level values. Note: All component choices must be made       */
/* before this call.                                                         */
/* If a memory allocation error has occurred, go_turbo returns OUT_OF_MEMORY */
/* If turbo was escaped by the user, go_turbo returns TURBO_ESCAPED          */
/* Otherwise it returns ZERO (0)                                             */

go_turbo()

{
   PROT_ELEM *PE_ptr;
   int layer, PE_num, target_node_num, error_check;
   float escape_check;

   target_node_num = get_area_node_number(MAX_AREAS_PER_ASD-1);

   /* free the old turbo tree space before reinitializing */
   free_tree();

   /* initialize turbo */
   error_check = turbo_init(get_num_rfts(), TOP_VULNERABLE_PATHS,
                            target_node_num, 0, target_node_num,
                            (get_current_direction() == EXIT));

   if (error_check == OUT_OF_MEMORY) return(error_check);

   /* insert a segment for each present path segment in the current ASD */
   for (layer = 0; layer < MAX_LAYERS_PER_ASD; layer++)
   {
      if (get_ASD_layer_bypass(layer))
      {
         error_check = insert_bypass(layer);
         if (error_check == OUT_OF_MEMORY) return(error_check);
      }
      else
      {
         for (PE_num = 0; PE_num < MAX_PE_PER_LAYER; PE_num++)
         {
            PE_ptr = get_ASD_PE_ptr(layer,PE_num);

            if (PE_ptr->present)
            {
               error_check = insert_segments(layer, PE_num, UNAUTH);
               if (error_check == OUT_OF_MEMORY) return(error_check);

               /* insert AUTHORIZED segments if    */
               /* authorized passage is possible   */
               /* and MIXED strategy is selected   */
               if ( get_PE_cat_spec_ptr(PE_ptr->type)->authorized &&
                    get_current_strategy() == MIXED )
               {
                  error_check = insert_segments(layer, PE_num, AUTH);

                  if (error_check == OUT_OF_MEMORY) return(error_check);
               }
```

```
            }
         }
      }
   }

   layer = MAX_LAYERS_PER_ASD;
   PE_num = 0;

   /* insert a segment for each of the task PE's path segments */
   /* Note: the task PE can ONLY be traversed in an unauthorized way */
   error_check = insert_segments(layer, PE_num, UNAUTH);
   if (error_check == OUT_OF_MEMORY) return(error_check);

   /********/
   /* saviwalk(get_ASD_rft_table_ptr(), get_num_rfts()); */
   /********/

   /* note: turbo_SEApath returns the number of paths analyzed upon */
   /* completion.  This value is negative if the user escapes prior */
   /* to completion of analysis                                     */
   escape_check = turbo_SEApath(get_ASD_rft_table_ptr(), get_num_rfts());

   if (escape_check < 0.0) return(TURBO_ESCAPED);

   return(0);
}

/* ------------------------------------------------------------------------ */

/* insert_segments inserts both input and output segment-links into the      */
/* turbo tree for the given PE for the given authorized/unauthorized state   */
/* If the maximum non-detection entry values on either segment is 0 (ie: the */
/* minimum detection probability is 100% on entry) and this is an authorized */
/* request, the segments are NOT inserted into the turbo tree since this     */
/* situation indicates that the intruder cannot pass this PE in an           */
/* authorized way                                                            */
/* Note: auth_unauth must be AUTH or UNAUTH.                                 */
/* Note: insert_segment returns OUT_OF_MEMORY if a memory allocation         */
/* error occurs in the call to insert_link, otherwise ZERO (0) is returned   */

insert_segments(layer, PE_num, auth_unauth)
int layer, PE_num, auth_unauth;

{
   PROT_ELEM *PE_ptr;
   int link_return, eqn, input_segment_num, output_segment_num;
   float input_entry_delay, input_exit_delay, input_exit_dep_delay,
         input_entry_nondet, input_exit_nondet, input_exit_dep_nondet,
         output_entry_delay, output_exit_delay, output_exit_dep_delay,
         output_entry_nondet, output_exit_nondet, output_exit_dep_nondet,
         temp_transit;

   PE_ptr = get_ASD_PE_ptr(layer, PE_num);
```

```
/* calculate the six segment values for the input segment */
min_delay_traversal(layer, PE_num, auth_unauth, INPUT_SEG,
                    ENTRY, &input_entry_delay, &eqn);
min_delay_traversal(layer, PE_num, auth_unauth, INPUT_SEG,
                    EXIT, &input_exit_delay, &eqn);
min_delay_traversal(layer, PE_num, auth_unauth, INPUT_SEG,
                    EXIT_DEP, &input_exit_dep_delay, &eqn);
max_nondet_traversal(layer, PE_num, auth_unauth, INPUT_SEG,
                     ENTRY, &input_entry_nondet, &eqn);
max_nondet_traversal(layer, PE_num, auth_unauth, INPUT_SEG,
                     EXIT, &input_exit_nondet, &eqn);
max_nondet_traversal(layer, PE_num, auth_unauth, INPUT_SEG,
                     EXIT_DEP, &input_exit_dep_nondet, &eqn);

/* calculate the six segment values for the output segment */
min_delay_traversal(layer, PE_num, auth_unauth, OUTPUT_SEG,
                    ENTRY, &output_entry_delay, &eqn);
min_delay_traversal(layer, PE_num, auth_unauth, OUTPUT_SEG,
                    EXIT, &output_exit_delay, &eqn);
min_delay_traversal(layer, PE_num, auth_unauth, OUTPUT_SEG,
                    EXIT_DEP, &output_exit_dep_delay, &eqn);
max_nondet_traversal(layer, PE_num, auth_unauth, OUTPUT_SEG,
                     ENTRY, &output_entry_nondet, &eqn);
max_nondet_traversal(layer, PE_num, auth_unauth, OUTPUT_SEG,
                     EXIT, &output_exit_nondet, &eqn);
max_nondet_traversal(layer, PE_num, auth_unauth, OUTPUT_SEG,
                     EXIT_DEP, &output_exit_dep_nondet, &eqn);

/* if the current traversal is Authorized and the */
/* maximum non-detection entry values on either   */
/* segment is 0 then do NOT insert these segments */
if ( (auth_unauth == AUTH) &&
     ( (input_entry_nondet == 0.0) || (output_entry_nondet == 0.0) ) )
   return(0);

/* add in transit times to input segment delay values */
input_entry_delay += combined_area_transit_time(PE_ptr->input_area,
                                                INPUT_SEG);
input_exit_delay += element_transit_time(layer,PE_num);
input_exit_dep_delay += element_transit_time(layer,PE_num);

/* insert input segment link */

/* if this PE is authorized, then set the segment numbers to the */
/* values returned by the previous call to insert_link for the   */
/* unauthorized clone of this PE                                 */
/* if this PE is unauthorized, then set the segment number to -1 */
if (auth_unauth == AUTH)
{
   input_segment_num = prev_input_segment_num;
   output_segment_num = prev_output_segment_num;
}
else
{
```

```
      input_segment_num = -1;
      output_segment_num = -1;
   }

   link_return = insert_link(get_area_node_number(PE_ptr->input_area),
                             get_PE_node_number(layer, PE_num, auth_unauth),
                             (int)input_entry_delay, input_entry_nondet,
                             (int)input_exit_delay, input_exit_nondet,
                             (int)input_exit_dep_delay, input_exit_dep_nondet,
                             input_segment_num);
   if (link_return == OUT_OF_MEMORY)
      return(link_return);
   else
      prev_input_segment_num = link_return;

   /* add in transit times to output segment delay values */
   temp_transit = combined_area_transit_time(PE_ptr->output_area,
                                             OUTPUT_SEG);
   output_entry_delay += element_transit_time(layer,PE_num);
   output_exit_delay += temp_transit;
   output_exit_dep_delay += temp_transit;

   /* insert output segment link */

   /* note that the clone node number is always -1 for an area */
   link_return = insert_link(get_PE_node_number(layer, PE_num, auth_unauth),
                             get_area_node_number(PE_ptr->output_area),
                             (int)output_entry_delay, output_entry_nondet,
                             (int)output_exit_delay, output_exit_nondet,
                             (int)output_exit_dep_delay,
                             output_exit_dep_nondet, output_segment_num);
   if (link_return == OUT_OF_MEMORY)
      return(link_return);
   else
   {
      prev_output_segment_num = link_return;
      return(0);
   }
}

/* ------------------------------------------------------------------------ */

/* insert_bypass inserts a PE node with NULL input and output segment links  */
/* between the areas which bracket the given layer to indicate a bypass in   */
/* the turbo tree                                                            */

insert_bypass(layer)
int layer;

{
   int error_check, PE_node_num;

   PE_node_num = get_PE_node_number(layer, 0, UNAUTH);
```

```
   error_check = insert_link(get_area_node_number(layer),
                             PE_node_num,
                             0, 1.0, 0, 1.0, 0, 1.0, -1);
   if (error_check == OUT_OF_MEMORY) return(error_check);

   error_check = insert_link(PE_node_num,
                             get_area_node_number(layer+1),
                             0, 1.0, 0, 1.0, 0, 1.0, -1);
   if (error_check == OUT_OF_MEMORY) return(error_check);

   return(0);
}

/* ------------------------------------------------------------------------ */

analyze_menu_init()

{
   cl_att = LNORMAL;

   /* Define the top level menu form */
   analyze_menu = def_frm(0, 0, 1, 80, LFIELDI, BDR_0P);

   /* Define the analyze submenu forms */
   analyze_results_sub_menu = def_frm(0, 0, 1, 80, LFIELDI, BDR_0P);
   analyze_print_sub_menu = def_frm(0, 0, 1, 80, LFIELDI, BDR_0P);

   /* Define the functions for the analyze menu */
   def_mfld(0, 1, "Results", "Vulnerability      Recommendations      \
Graphs      Quit",
            analyze_results_sub_menu, NULLFP, analyze_menu);
   def_mfld(0, 14, "Print", "Vulnerability      Recommendations      Quit",
            analyze_print_sub_menu, NULLFP, analyze_menu);
   def_mfld(0, 25, "Quit", "Return to Main Menu",
            NULLP, savi_analyze_quit, analyze_menu);

   /* Define the functions for the analyze results sub-menu */
   def_mfld(0, 1, "Vulnerability", "Display the Most Vulnerable Paths \
for any RFT in Range",
            NULLP, savi_analyze_results_vulnerability,
            analyze_results_sub_menu);
   def_mfld(0, 20, "Recommendations","Recommend Improvements to the \
Most Vulnerable Paths for any RFT in Range",
            NULLP, savi_analyze_results_recommendations,
            analyze_results_sub_menu);
   def_mfld(0, 41, "Graphs", "Type      Response      Zoom      Scale\
      Print      Quit",
            NULLP, savi_analyze_results_graphs, analyze_results_sub_menu);
   def_mfld(0, 53, "Quit", "Return to Analyze Menu",
            NULLP, savi_analyze_results_quit, analyze_results_sub_menu);

   /* Define the functions for the analyze print sub-menu */
   def_mfld(0, 1, "Vulnerability", "Print the Most Vulnerable Paths",
            NULLP, savi_analyze_print_vulnerability, analyze_print_sub_menu);
```

```
   def_mfld(0, 20, "Recommendations", "Print the Recommended Improvements \
to the Most Vulnerable Paths",
            NULLP, savi_analyze_print_recommendations, analyze_print_sub_menu);
   def_mfld(0, 41, "Quit", "Return to Analyze Menu",
            NULLP, savi_analyze_print_quit, analyze_print_sub_menu);

     return;
}

/* ------------------------------------------------------------------------ */

/* select_current_rft allows the user to select the current RFT to be used  */
/* in the display of analysis results.                                      */
/* Note: TRUE is returned if the user selects an RFT, FALSE is returned if  */
/* the user presses escape to exit the window                               */

select_current_rft()

{
   int key, rft_num, start_rft_num, num_rfts,
       done, last_key_was_1, selected;

   selected = FALSE;

   popup_current_rft_window();

   num_rfts = get_num_rfts();

   rft_num = get_current_rft_num();
   start_rft_num = rft_num;

   last_key_was_1 = FALSE;
   done = FALSE;

   /* display the current rft */
   display_current_rft(rft_num);

   /* process key press */
   while (!done)
   {
      /* dump the keyboard input buffer */
      dump_key_buffer();

      /* wait for a key press */
      key = ki();

      /* process key */
      /* Note: ki() returns the negative of the key scan code  */
      /* if an extended code key is pressed (ie: UP, DOWN etc) */
      /* and returns the ASCII code directly otherwise         */
      switch (-key)
      {
         /* LEFT and UP arrows decrement the RFT number */
         /* not including wrap around                   */
```

```
case K_LEFT    : ;
case K_UP      : if (rft_num > 0)
                 {
                    rft_num--;
                    display_current_rft(rft_num);
                 }

                 break;

/* RIGHT and DOWN arrows increment the RFT number */
/* not including wrap around                      */
case K_RIGHT   : ;
case K_DN      : if (rft_num < num_rfts-1)
                 {
                    rft_num++;
                    display_current_rft(rft_num);
                 }

                 break;

/* HOME and PAGE UP keys */
/* set the rft number to 0 */
case K_HOME    : ;
case K_PGUP    : rft_num = 0;

                 display_current_rft(rft_num);

                 break;

/* END and PAGE DOWN keys */
/* set the rft number to num_rfts-1 */
case K_END     : ;
case K_PGDN    : rft_num = num_rfts-1;

                 display_current_rft(rft_num);

                 break;

/* ENTER (Return) key */
/* user has selected an rft */
case -K_ENTER : selected = TRUE;
                 done = TRUE;
                 break;

/* ESCAPE key */
/* abort rft selection */
case -K_ESC    : done = TRUE;
                 break;

/* ZERO key (only valid after a 1) */
case (-'0')    : if ( (last_key_was_1) && (num_rfts == 10) )
                 {
                    rft_num = 9;
                    display_current_rft(rft_num);
```

```
                        }
                        else
                           bell();

                        last_key_was_1 = FALSE;

                        break;

            /* any non-zero single digit key */
            case ('1')   : ;
            case ('2')   : ;
            case ('3')   : ;
            case ('4')   : ;
            case ('5')   : ;
            case ('6')   : ;
            case ('7')   : ;
            case ('8')   : ;
            case ('9')   : if (key == '1')
                              last_key_was_1 = TRUE;
                           else
                              last_key_was_1 = FALSE;

                           if ((key - '0') <= num_rfts)
                              rft_num = (key - '0') - 1;
                           else
                              bell();

                           display_current_rft(rft_num);

                           break;

            default      : bell();
                           break;
         }
      }

      /* set the new current RFT */
      set_current_rft_num(rft_num);

      /* if the current rft number was changed then  */
      /* set the current path to the most vulnerable */
      if (rft_num != start_rft_num)
         set_current_path(0);

      remove_current_rft_window();

      dump_key_buffer();

      return(selected);
   }

   /* ------------------------------------------------------------------------ */

   /* savi_analyze_results_vulnerability allows the user to scroll through the  */
```

```
/* most vulnerable paths by highlighting each path on the current ASD and    */
/* display the measures of vulnerability as well as any diagnostics          */

savi_analyze_results_vulnerability()

{
   int key, path_num, old_path_num, last_path_num,
       rft_num, old_rft_num, num_rfts,
       done, windows_state, changed;

   done = FALSE;

   /* start with both index and diagnostics windows on screen */
   windows_state = 2;

   path_num = current_path;
   old_path_num = path_num;
   last_path_num = get_number_of_critical_paths() - 1;

   rft_num = get_current_rft_num();
   old_rft_num = rft_num;
   num_rfts = get_num_rfts();

   /* highlight the most vulnerable path */
   highlight_path(rft_num, path_num);

   /* highlight the TDP if possible */
   highlight_tdp(rft_num, path_num);

   /* display the index and diagnostics data windows */
   popup_results_index_window();
   display_results_index(rft_num,path_num);
   popup_diagnostics_window();
   display_diagnostics_vals(rft_num, path_num);

   /* process key press */
   while (!done)
   {
      /* dump the keyboard input buffer */
      dump_key_buffer();

      /* wait for a key press */
      key = ki();

      /* process key */
      /* Note: ki() returns the negative of the key scan code  */
      /* if an extended code key is pressed (ie: UP, DOWN etc) */
      /* and returns the ASCII code directly otherwise         */
      switch (-key)
      {
         /* LEFT arrow decrements the path */
         /* without wrap around            */
         case K_LEFT   : if (path_num > 0)
                         {
```

```
                    changed = TRUE;
                    path_num--;
                 }
                 else
                    changed = FALSE;
                 break;

/* RIGHT arrow increments the path */
/* without wrap around             */
case K_RIGHT   : if (path_num < last_path_num)
                 {
                    changed = TRUE;
                    path_num++;
                 }
                 else
                    changed = FALSE;
                 break;

/* HOME key displays the most vulnerable path */
case K_HOME    : if (path_num > 0)
                 {
                    changed = TRUE;
                    path_num = 0;
                 }
                 else
                    changed = FALSE;
                 break;

/* END key displays the least vulnerable path */
case K_END     : if (path_num < last_path_num)
                 {
                    changed = TRUE;
                    path_num = last_path_num;
                 }
                 else
                    changed = FALSE;
                 break;

/* UP arrow decrements the RFT number */
/* without wrap around                */
case K_UP      : if (rft_num > 0)
                 {
                    changed = TRUE;
                    rft_num--;
                    set_current_rft_num(rft_num);
                 }
                 else
                    changed = FALSE;
                 break;

/* DOWN arrow increments the RFT number */
/* without wrap around                  */
case K_DN      : if (rft_num < num_rfts-1)
                 {
```

```
                    changed = TRUE;
                    rft_num++;
                    set_current_rft_num(rft_num);
                 }
                 else
                    changed = FALSE;
                 break;

/* PAGE UP key sets the rft number to 0 */
case K_PGUP    : if (rft_num > 0)
                 {
                    changed = TRUE;
                    rft_num = 0;
                    set_current_rft_num(rft_num);
                 }
                 else
                    changed = FALSE;
                 break;

/* PAGE DOWN key sets to the largest rft number */
case K_PGDN    : if (rft_num < num_rfts-1)
                 {
                    changed = TRUE;
                    rft_num = num_rfts-1;
                    set_current_rft_num(rft_num);
                 }
                 else
                    changed = FALSE;
                 break;

/* ESCAPE key */
/* leave this function */
case -K_ESC    : done = TRUE;
                 changed = FALSE;
                 break;

/* SPACE bar */
/* toggle the windows display */
case -K_SPACE : switch (windows_state)
                 {
                    /* no windows are on screen */
                    /* popup both windows       */
                    case 0 : windows_state = 2;
                             popup_results_index_window();
                             display_results_index(rft_num,
                                                   path_num);
                             popup_diagnostics_window();
                             display_diagnostics_vals(rft_num,
                                                      path_num);
                             break;

                    /* index window is on screen */
                    /* remove index window       */
                    case 1 : windows_state = 0;
```

```
                                    remove_results_index_window();
                                    break;

                              /* both windows are on screen */
                              /* remove diagnostics window  */
                              case 2 : windows_state = 1;
                                    remove_diagnostics_window();
                                    break;
                           }

                           changed = FALSE;

                           break;

         default        : bell();
                          changed = FALSE;
                          break;
      }

      if (changed)
      {
         if (windows_state != 0)
         {
            popup_results_index_window();
            display_results_index(rft_num,path_num);
         }

         remove_diagnostics_window();
         unhighlight_path(old_rft_num, old_path_num);

         dump_key_buffer();

         /* highlight the new path */
         highlight_path(rft_num, path_num);

         /* highlight the TDP if possible */
         highlight_tdp(rft_num, path_num);

         dump_key_buffer();

         if (windows_state == 2)
         {
            popup_diagnostics_window();
            display_diagnostics_vals(rft_num, path_num);
         }

         old_path_num = path_num;
         old_rft_num = rft_num;
      }
   }

   /* remove the windows */
   remove_diagnostics_window();
   remove_results_index_window();
```

```
   unhighlight_path(old_rft_num, old_path_num);

   /* set the current path to the last path accessed in this function */
   set_current_path(old_path_num);

   dump_key_buffer();

   return(0);
}

/* ------------------------------------------------------------------------ */

/* savi_analyze_results_recommendations allows the user to display the      */
/* recommended changes to the most vulnerable paths                         */

savi_analyze_results_recommendations()

{
   int key, path_num, old_path_num, last_path_num,
       rft_num, old_rft_num, num_rfts,
       done, windows_state, changed;

   done = FALSE;

   /* start with both index and recommendations windows on screen */
   windows_state = 2;

   path_num = current_path;
   old_path_num = path_num;
   last_path_num = get_number_of_critical_paths() - 1;

   rft_num = get_current_rft_num();
   old_rft_num = rft_num;
   num_rfts = get_num_rfts();

   /* highlight the most vulnerable path */
   highlight_path(rft_num, path_num);

   /* highlight the TDP if possible */
   highlight_tdp(rft_num, path_num);

   /* display the index and recommendations data windows */
   popup_results_index_window();
   display_results_index(rft_num,path_num);
   popup_recommendations_window();
   display_recommendations(rft_num, path_num);

   /* process key press */
   while (!done)
   {
      /* dump the keyboard input buffer */
      dump_key_buffer();

      /* wait for a key press */
```

```
key = ki();

/* process key */
/* Note: ki() returns the negative of the key scan code  */
/* if an extended code key is pressed (ie: UP, DOWN etc) */
/* and returns the ASCII code directly otherwise         */
switch (-key)
{
   /* LEFT arrow decrements the path */
   /* without wrap around            */
   case K_LEFT   : if (path_num > 0)
                   {
                      changed = TRUE;
                      path_num--;
                   }
                   else
                      changed = FALSE;
                   break;

   /* RIGHT arrow increments the path */
   /* without wrap around             */
   case K_RIGHT  : if (path_num < last_path_num)
                   {
                      changed = TRUE;
                      path_num++;
                   }
                   else
                      changed = FALSE;
                   break;

   /* HOME key displays the most vulnerable path */
   case K_HOME   : if (path_num > 0)
                   {
                      changed = TRUE;
                      path_num = 0;
                   }
                   else
                      changed = FALSE;
                   break;

   /* END key displays the least vulnerable path */
   case K_END    : if (path_num < last_path_num)
                   {
                      changed = TRUE;
                      path_num = last_path_num;
                   }
                   else
                      changed = FALSE;
                   break;

   /* UP arrow decrements the RFT number */
   /* without wrap around                */
   case K_UP     : if (rft_num > 0)
                   {
```

```
                    changed = TRUE;
                    rft_num--;
                    set_current_rft_num(rft_num);
                 }
                 else
                    changed = FALSE;
                 break;

/* DOWN arrow increments the RFT number */
/* without wrap around                  */
case K_DN      : if (rft_num < num_rfts-1)
                 {
                    changed = TRUE;
                    rft_num++;
                    set_current_rft_num(rft_num);
                 }
                 else
                    changed = FALSE;
                 break;

/* PAGE UP key sets the rft number to 0 */
case K_PGUP    : if (rft_num > 0)
                 {
                    changed = TRUE;
                    rft_num = 0;
                    set_current_rft_num(rft_num);
                 }
                 else
                    changed = FALSE;
                 break;

/* PAGE DOWN key sets to the largest rft number */
case K_PGDN    : if (rft_num < num_rfts-1)
                 {
                    changed = TRUE;
                    rft_num = num_rfts-1;
                    set_current_rft_num(rft_num);
                 }
                 else
                    changed = FALSE;
                 break;

/* ESCAPE key */
/* leave this function */
case -K_ESC    : done = TRUE;
                 changed = FALSE;
                 break;

/* SPACE bar */
/* toggle the windows display */
case -K_SPACE : switch (windows_state)
                 {
                    /* no windows are on screen */
                    /* popup both windows       */
```

```
                        case 0 : windows_state = 2;
                                 popup_results_index_window();
                                 display_results_index(rft_num,
                                                       path_num);
                                 popup_recommendations_window();
                                 display_recommendations(rft_num,
                                                         path_num);
                                 break;

                        /* index window is on screen */
                        /* remove index window       */
                        case 1 : windows_state = 0;
                                 remove_results_index_window();
                                 break;

                        /* both windows are on screen     */
                        /* remove recommendations window  */
                        case 2 : windows_state = 1;
                                 remove_recommendations_window();
                                 break;
                    }

                    changed = FALSE;

                    break;

    default       : bell();
                    changed = FALSE;
                    break;
}

if (changed)
{
    if (windows_state != 0)
    {
       popup_results_index_window();
       display_results_index(rft_num,path_num);
    }

    remove_recommendations_window();
    unhighlight_path(old_rft_num, old_path_num);

    dump_key_buffer();

    /* highlight the new path */
    highlight_path(rft_num, path_num);

    /* highlight the TDP if possible */
    highlight_tdp(rft_num, path_num);

    dump_key_buffer();

    if (windows_state == 2)
    {
```

```
            popup_recommendations_window();
            display_recommendations(rft_num, path_num);
         }

         old_path_num = path_num;
         old_rft_num = rft_num;
      }

   }

   /* remove the windows */
   remove_recommendations_window();
   remove_results_index_window();
   unhighlight_path(old_rft_num, old_path_num);

   /* set the current path to the last path accessed in this function */
   set_current_path(old_path_num);

   dump_key_buffer();

   return(0);
}

/* ------------------------------------------------------------------------ */

/* highlight_path highlights all protection elements on path path_num       */
/* within the current ASD for the given rft                                 */

highlight_path(rft_num, path_num)
int rft_num, path_num;

{
    int layer, PE_num, direction, auth_unauth, count, done, check;

    count = 0;
    done = FALSE;

    while (!done)
    {
       check = get_PE_on_path(rft_num, path_num, count,
                              &layer, &PE_num,
                              &direction, &auth_unauth);

       if (check == NOT_FOUND)
          done = TRUE;
       else
       {
          /* only mark task PE or present PE */
          if (layer == MAX_LAYERS_PER_ASD ||
              get_ASD_PE_ptr(layer,PE_num)->present)
          {
             /* mark the PE direction and authorized/unauthorized traversal */
             mark_PE_traversal(layer, PE_num, direction, auth_unauth);
```

```
            /* highlight PE on given path */
            highlight_PE(layer, PE_num);
         }

         count++;
      }
   }

   return;
}

/* ------------------------------------------------------------------------ */

/* unhighlight_path unhighlights all protection elements on path path_num    */
/* within the current ASD for the given rft                                  */

unhighlight_path(rft_num, path_num)
int rft_num, path_num;

{
   int layer, PE_num, direction, auth_unauth, count, done, check;

   count = 0;
   done = FALSE;

   while (!done)
   {
      check = get_PE_on_path(rft_num, path_num, count,
                             &layer, &PE_num,
                             &direction, &auth_unauth);

      if (check == NOT_FOUND)
         done = TRUE;
      else
      {
         /* only mark task PE or present PE */
         if (layer == MAX_LAYERS_PER_ASD ||
             get_ASD_PE_ptr(layer,PE_num)->present)
         {
            /* unhighlight PE on given path */
            unhighlight_PE(layer, PE_num);
         }

         count++;
      }
   }

   return;
}

/* ------------------------------------------------------------------------ */

/* highlight_TDP highlights the Timely Detection Point on path path_num for  */
/* the given rft if possible                                                 */
```

```
highlight_TDP(rft_num, path_num)
int rft_num, path_num;

{
    int timely_layer, timely_PE_num, segment, direction, test;

    /* determine the Timely Detection Point */
    test = timely_detection_point(rft_num, path_num,
                                  &timely_layer, &timely_PE_num,
                                  &segment, &direction);

    /* Do not display the Timely Detection Point if the cumulative */
    /* path delay is less than the RFT or TDP is OFFSITE           */
    if ((test == PATH_DELAY_DEFICIENCY) || (test == TDP_OFFSITE)) return;

    /* highlight the Timely Detection Point */
    highlight_PE_segment(timely_layer,timely_PE_num,segment,direction);

    return;
}

/* ------------------------------------------------------------------------ */

/* timely_detection_point loads layer and PE_num to specify the PE on the    */
/* given path for the given rft at which timely detection must take place.   */
/* segment is loaded with PATH_IN or PATH_OUT depending on which segment the */
/* timely detection point is on for the particular PE.                       */
/* direction is set to the direction of traversal of the given PE (ENTRY or  */
/* EXIT) when the tdp occurs.                                                */
/*                                                                           */
/* Note: if the Response Force Time is longer than the cumulative delay      */
/* along the given path, then timely detection is not possible and this      */
/* function returns PATH_DELAY_DEFICIENCY. In this case the timely_layer,    */
/* timely_PE_num and segment are not modified.                               */
/*                                                                           */
/* If no detection exists before the timely detection point on               */
/* the given path (ie: all detectors prior to the point are set to a nondet  */
/* value of 1) then this function returns NO_TIMELY_DETECTION, and the       */
/* timely_layer, timely_PE_num and segment reflect the Timely Detection      */
/* Point anyway.                                                             */
/*                                                                           */
/* If the timely detection point is found to be prior to the first path      */
/* segment leading from OFFSITE, this function returns TDP_OFFSITE and       */
/* timely_layer is set to -1 and timely_PE_num is set to 0                   */
/*                                                                           */
/* Otherwise, if the timely detection point is found and detection exists    */
/* prior to this point, this function returns TIMELY_DETECTION_OK            */

timely_detection_point(rft_num, path_num,
                       timely_layer_ptr, timely_PE_num_ptr,
                       segment_ptr, direction_ptr)
int rft_num, path_num, *timely_layer_ptr, *timely_PE_num_ptr,
                       *segment_ptr, *direction_ptr;
```

```
{
   PATH_INFO *path_ptr;
   int bp_node_num, bp_node_val, tdp_traversal;

   path_ptr = get_path_ptr(rft_num, path_num);

   /* ensure that the break point is on a correct node relative to bypasses */
   adjust_break_point(path_ptr);

   bp_node_num = path_ptr->break_point;

   /* check if the rft was ever exceeded on the given path */
   if (bp_node_num == THRESH_NEVER_PASSED)
      return(PATH_DELAY_DEFICIENCY);

   /* check if the break point falls on the offsite node (0) */
   if (bp_node_num == 0)
   {
      *timely_layer_ptr = -1;
      *timely_PE_num_ptr = 0;
      return(TDP_OFFSITE);
   }

   /* tdp exists */

   tdp_traversal = get_node_traversal(path_ptr, bp_node_num);

   *direction_ptr = tdp_traversal;

   bp_node_val = path_ptr->nodes[bp_node_num];

   if (node_is_area(bp_node_val))
   /* break point is on an area */
   {
      /* set tdp to PE before break point area */
      *timely_layer_ptr = get_PE_layer(path_ptr->nodes[bp_node_num - 1]);
      *timely_PE_num_ptr = get_PE_num(path_ptr->nodes[bp_node_num - 1]);

      if (tdp_traversal == ENTRY)
      /* tdp is on output path segment of PE before break point area */
      {
         *segment_ptr = PATH_OUT;
      }
      else
      /* tdp is on input path segment of PE before break point area */
      {
         *segment_ptr = PATH_IN;
      }
   }
   else
   /* break point is on a PE */
   {
      /* set tdp to break point PE */
```

```
      *timely_layer_ptr = get_PE_layer(bp_node_val);
      *timely_PE_num_ptr = get_PE_num(bp_node_val);

      if (tdp_traversal == ENTRY)
      /* tdp is on input path segment of break point PE */
      {
         *segment_ptr = PATH_IN;
      }
      else
      /* tdp is on output path segment of break point PE */
      {
         *segment_ptr = PATH_OUT;
      }
   }

   if (path_ptr->cum_prod == 1)
   /* tdp exists without prior detection probability */
   {
      return(NO_TIMELY_DETECTION);
   }
   else
   /* tdp exists with prior detection probability */
   {
      return(TIMELY_DETECTION_OK);
   }
}

/* ------------------------------------------------------------------------ */

/* adjust_break_point adjusts the break point on the given path if necessary */
/* to allow for bypassed layers.  This ensures that the break point is on a  */
/* present PE or points to an area which is preceded by a present PE         */

adjust_break_point(path_ptr)
PATH_INFO *path_ptr;

{
   int done, bp_node_num, prev_layer;

   bp_node_num = path_ptr->break_point;

   /* if the break point is negative (indicating an error condition) */
   /* then it cannot be adjusted                                     */
   if (bp_node_num < 0) return;

   /* if the break point is on a PE then it does not need adjusting */
   /* since break point cannot fall on a NULL (bypassed) PE         */
   if (node_is_PE(path_ptr->nodes[bp_node_num])) return;

   done = FALSE;

   while (!done)
   {
      /* if the break point is on the origin node then return */
```

```
      if (bp_node_num == 0)
         done = TRUE;
      else
      {
         /* if layer before break point area is bypassed then */
         /* decrement break point to next area node           */
         prev_layer = get_PE_layer(path_ptr->nodes[bp_node_num - 1]);

         if (get_ASD_layer_bypass(prev_layer))
            bp_node_num -= 2;
         else
            done = TRUE;
      }
   }

   path_ptr->break_point = bp_node_num;

   return;
}

/* ------------------------------------------------------------------------ */

/* savi_analyze_results_graphs allows the user to specify which graph to    */
/* produce for display of vulnerability analysis results                    */

savi_analyze_results_graphs()

{
   int current_rft_num, start_rft_num;

   /* save the current text screen within the  */
   /* default full screen window by popping up */
   /* the full screen window over the current  */
   /* screen                                   */
   sw_popup(ON,&wn0);
   set_wn(&wn0);

   start_rft_num = get_current_rft_num();
   current_rft_num = start_rft_num;

   graph_service(get_ASD_rft_table_ptr(),get_num_rfts(),&current_rft_num);

   set_current_rft_num(current_rft_num);

   /* if the current rft number was changed then  */
   /* set the current path to the most vulnerable */
   if (current_rft_num != start_rft_num)
      set_current_path(0);

   /* hide the screen cursor */
   csr_hide();

   /* recreate the screen by removing the full */
   /* screen popup window                      */
```

```
   unset_wn(&wn0);

   dump_key_buffer();

   return(0);
}

/* ------------------------------------------------------------------------ */

savi_analyze_results_quit()

{
   dump_key_buffer();

   /* return a 1 to jump up one menu level */
   return(1);
}

/* ------------------------------------------------------------------------ */

savi_analyze_print_quit()

{
   dump_key_buffer();

   /* return a 1 to jump up one menu level */
   return(1);
}

/* ------------------------------------------------------------------------ */

savi_analyze_quit()

{
   dump_key_buffer();

   /* return a -2 to quit the analyze menu */
   return(-2);
}

/* ------------------------------------------------------------------------ */

/* set_vulnerability_current sets the current vulnerability to the given    */
/* state.  TRUE indicates that the current ASD vulnerability has been       */
/* determined and is current.                                               */

set_vulnerability_current(state)
int state;

{
   /* each time the results state is changed, some data in the current ASD */
   /* has been changed, therefore, the current ASD state including results */
   /* has not been saved at this point                                     */
   set_current_ASD_saved(FALSE);
```

```
   /* set and display the current analysis state */
   set_analysis_state(state);
   display_analysis_state(state);

   return;
}

/* ------------------------------------------------------------------------ */

/* set_current_ASD_saved sets the state of the variable which indicates     */
/* whether the current ASD and its results have been saved since the last   */
/* changes were made                                                        */

set_current_ASD_saved(state)
int state;

{
   current_ASD_saved = state;

   return;
}

/* ------------------------------------------------------------------------ */

/* get_current_ASD_saved returns the state of the variable which indicates  */
/* whether the current ASD and its results have been saved since the last   */
/* changes were made                                                        */

get_current_ASD_saved()

{
   return(current_ASD_saved);
}

/* ------------------------------------------------------------------------ */

/* set_current_path sets the current path for use by the display results    */
/* functions under the Analyze menu                                         */

set_current_path(path_num)
int path_num;

{
   current_path = path_num;

   return;
}

/* ------------------------------------------------------------------------ */

/* set_current_rft_num sets the current rft to the given rft number. The    */
/* rft_num corresponds to the index of the desired rft in the rft table     */
```

```
set_current_rft_num(rft_num)
int rft_num;

{
   current_rft_num = rft_num;

   return;
}

/* ------------------------------------------------------------------------ */

/* get_current_rft_num returns the current rft number. The rft number       */
/* corresponds to the index of the desired rft in the rft table             */

get_current_rft_num()

{
   return(current_rft_num);
}

/* ********************** end SAVIANAL.C source file ********************** */

/* ************************ SAVIAREA.C source file ************************** */

#include <wfd.h>
#include <savidefs.h>

/* ------------------------------------------------------------------------ */

/* savi_modify_area_labels allows the user to modify the area labels for the */
/* current ASD                                                               */

savi_modify_area_labels()

{
   int area_num, key, done, window_on;

   area_num = 1;
   done = FALSE;
   window_on = TRUE;

   /* highlight the first modifiable area */
   highlight_area_label(area_num);

   /* display the current label window */
   popup_area_label_window();
   update_area_label_window(area_num);

   /* process key press */
   while (!done)
   {
      /* dump the keyboard input buffer */
      dump_key_buffer();
```

```
/* wait for a key press */
key = ki();

/* process key */
/* Note: ki() returns the negative of the key scan code  */
/* if an extended code key is pressed (ie: UP, DOWN etc) */
/* and returns the ASCII code directly otherwise         */
switch (-key)
{
   /* LEFT and UP arrows */
   /* highlight the area label above current area */
   /* do not wrap around top to bottom */
   case K_LEFT  : ;
   case K_UP    : if (area_num > 1)
                  {
                     unhighlight_area_label(area_num);
                     area_num--;

                     if (window_on)
                     {
                        update_area_label_window(area_num);
                     }

                     highlight_area_label(area_num);
                  }
                  else
                     bell();
                  break;

   /* RIGHT and DOWN arrows */
   /* highlight the area label below current area */
   /* do not wrap around bottom to top */
   case K_RIGHT : ;
   case K_DN    : if (area_num < MAX_MODIFIABLE_AREAS)
                  {
                     unhighlight_area_label(area_num);
                     area_num++;

                     if (window_on)
                     {
                        update_area_label_window(area_num);
                     }

                     highlight_area_label(area_num);
                  }
                  else
                     bell();
                  break;

   /* HOME and PAGE UP keys */
   /* highlight the first modifiable area label */
   case K_HOME  : ;
   case K_PGUP  : if (area_num != 1)
                  {
```

```
                    unhighlight_area_label(area_num);
                    area_num = 1;

                    if (window_on)
                    {
                       update_area_label_window(area_num);
                    }

                    highlight_area_label(area_num);
                 }
                 else
                    bell();
                 break;

/* END and PAGE DOWN keys */
/* highlight the last modifiable area label */
case K_END   : ;
case K_PGDN  : if (area_num != MAX_MODIFIABLE_AREAS)
                 {
                    unhighlight_area_label(area_num);
                    area_num = MAX_MODIFIABLE_AREAS;

                    if (window_on)
                    {
                       update_area_label_window(area_num);
                    }

                    highlight_area_label(area_num);
                 }
                 else
                    bell();
                 break;

/* ENTER (Return) key */
/* allow user to edit the highlighted area label */
case -K_ENTER : if (!window_on)
                 {
                    popup_area_label_window();
                    update_area_label_window(area_num);
                 }

                 edit_area_label(area_num);

                 update_area_label_window(area_num);

                 if (!window_on)
                    remove_area_label_window();

                 /* update the area label display */
                 unhighlight_area_label(area_num);
                 highlight_area_label(area_num);

                 break;
```

```
         /* SPACE bar */
         /* toggle the window display */
         case -K_SPACE : if (popup_window_on_screen())
                         {
                            window_on = FALSE;
                            remove_area_label_window();
                         }
                         else
                         {
                            window_on = TRUE;
                            popup_area_label_window();
                            update_area_label_window(area_num);
                         }

                         break;

         /* ESCAPE key */
         case -K_ESC   : done = TRUE;
                         break;

         default       : bell();
                         break;

      }
   }

   remove_area_label_window();

   /* unhighlight the current area label */
   unhighlight_area_label(area_num);

   dump_key_buffer();

   return(0);
}

/* ------------------------------------------------------------------------ */

/* edit_area_label allows the user to edit the first and second lines of the */
/* given area's label                                                        */

edit_area_label(area_num)
int area_num;

{
   DFIELDPTR field_ptr;
   char area_label_first_line[MAX_HALF_AREA_LABEL_LEN],
        area_label_second_line[MAX_HALF_AREA_LABEL_LEN];
   char temp_string[30];

   /* copy the current area label lines into the      */
   /* input strings so that the def_fld function      */
   /* can display the current label lines for editing */
   strcpy(area_label_first_line,get_ASD_area_label_first_line(area_num));
```

```
   strcpy(area_label_second_line,get_ASD_area_label_second_line(area_num));

   /* set the input form size */
   mod_wn(12,40,
          2,MAX_HALF_AREA_LABEL_LEN+1,
          STDWNP);

   /* set the input form to have no border */
   sw_border(BDR_OP,STDWNP);

   /* define the input form fields */
   field_ptr = def_fld(0,0,"",def_pic('X',MAX_HALF_AREA_LABEL_LEN-1),
                       F_STRING,area_label_first_line);
   sf_location(0,0,field_ptr);
   field_ptr = def_fld(0,0,"",def_pic('X',MAX_HALF_AREA_LABEL_LEN-1),
                       F_STRING,area_label_second_line);
   sf_location(1,0,field_ptr);

   /* accept input from user */
   if (rd_flds() != ABORT_FORM)
      set_current_ASD_saved(FALSE);

   /* set the new area label lines in the current ASD */
   set_ASD_area_label_first_line(area_num,area_label_first_line);
   set_ASD_area_label_second_line(area_num,area_label_second_line);

   return;
}

/* ------------------------------------------------------------------------ */

/* savi_modify_area_traversal allows the user to select the current area     */
/* traversal (INTERIOR or EXTERIOR) for any area in the current ASD          */

savi_modify_area_traversal()

{
   int area_num, last_area, old_traversal, new_traversal,
       key, done, window_on;

   area_num = 1;

   done = FALSE;
   window_on = TRUE;

   /* highlight the first modifiable area */
   highlight_area_label(area_num);

   /* display the traversal setting window for the given area */
   popup_area_traversal_window(area_num);

   /* mark the current traversal */
   mark_traversal(get_ASD_area_type(area_num));
```

```
/* process key press */
while (!done)
{
   /* dump the keyboard input buffer */
   dump_key_buffer();

   /* wait for a key press */
   key = ki();

   /* process key */
   /* Note: ki() returns the negative of the key scan code  */
   /* if an extended code key is pressed (ie: UP, DOWN etc) */
   /* and returns the ASCII code directly otherwise         */
   switch (-key)
   {
      /* LEFT and UP arrows */
      /* highlight the area label above current area */
      /* do not wrap around top to bottom */
      case K_LEFT  : ;
      case K_UP    : if (area_num > 1)
                     {
                        unhighlight_area_label(area_num);
                        last_area = area_num;
                        area_num--;

                        if (window_on)
                        {
                           popup_area_traversal_window(area_num);
                           unmark_traversal(get_ASD_area_type(last_area));
                           mark_traversal(get_ASD_area_type(area_num));
                        }

                        highlight_area_label(area_num);
                     }
                     else
                        bell();

                     break;

      /* RIGHT and DOWN arrows */
      /* highlight the area label below current area */
      /* do not wrap around bottom to top */
      case K_RIGHT : ;
      case K_DN    : if (area_num < MAX_MODIFIABLE_AREAS)
                     {
                        unhighlight_area_label(area_num);
                        last_area = area_num;
                        area_num++;

                        if (window_on)
                        {
                           popup_area_traversal_window(area_num);
                           unmark_traversal(get_ASD_area_type(last_area));
                           mark_traversal(get_ASD_area_type(area_num));
```

```
                    }

                    highlight_area_label(area_num);
                }
                else
                    bell();

                break;

/* HOME and PAGE UP keys */
/* highlight the first modifiable area label */
case K_HOME  : ;
case K_PGUP  : if (area_num != 1)
                {
                    unhighlight_area_label(area_num);
                    last_area = area_num;
                    area_num = 1;

                    if (window_on)
                    {
                        popup_area_traversal_window(area_num);
                        unmark_traversal(get_ASD_area_type(last_area));
                        mark_traversal(get_ASD_area_type(area_num));
                    }

                    highlight_area_label(area_num);
                }
                else
                    bell();

                break;

/* END and PAGE DOWN keys */
/* highlight the last modifiable area label */
case K_END   : ;
case K_PGDN  : if (area_num != MAX_MODIFIABLE_AREAS)
                {
                    unhighlight_area_label(area_num);
                    last_area = area_num;
                    area_num = MAX_MODIFIABLE_AREAS;

                    if (window_on)
                    {
                        popup_area_traversal_window(area_num);
                        unmark_traversal(get_ASD_area_type(last_area));
                        mark_traversal(get_ASD_area_type(area_num));
                    }

                    highlight_area_label(area_num);
                }
                else
                    bell();

                break;
```

```
        /* ENTER (Return) key */
        /* allow user to set transit value on current area */
        case -K_ENTER : if (!window_on)
                        {
                           popup_area_traversal_window(area_num);
                           mark_traversal(get_ASD_area_type(area_num));
                        }

                        old_traversal = get_ASD_area_type(area_num);

                        select_area_traversal_setting(area_num);

                        new_traversal = get_ASD_area_type(area_num);

                        if (!window_on)
                           remove_area_traversal_window();

                        /* update the area label traversal setting display */
                        /* if the traversal setting was modified           */
                        if (new_traversal != old_traversal)
                        {
                           unhighlight_area_label(area_num);
                           highlight_area_label(area_num);
                        }

                        break;

        /* SPACE bar */
        /* toggle the window display */
        case -K_SPACE : if (popup_window_on_screen())
                        {
                           window_on = FALSE;
                           remove_area_traversal_window();
                        }
                        else
                        {
                           window_on = TRUE;
                           popup_area_traversal_window(area_num);
                           mark_traversal(get_ASD_area_type(area_num));
                        }

                        break;

        /* ESCAPE key */
        case -K_ESC   : done = TRUE;
                        break;

        default       : bell();
                        break;
    }
}

remove_area_traversal_window();
```

```
   /* unhighlight the current area label */
   unhighlight_area_label(area_num);

   dump_key_buffer();

   return(0);
}

/* ------------------------------------------------------------------------ */

/* select_area_traversal_setting allows the user to select the current area  */
/* traversal setting (INTERIOR or EXTERIOR) for the given area               */

select_area_traversal_setting(area_num)
int area_num;

{
   int key, current_type, start_type, marked_type, done;

   current_type = get_ASD_area_type(area_num);
   start_type = current_type;
   marked_type = start_type;

   /* highlight the current setting within the traversal setting window */
   highlight_traversal_setting(current_type);

   done = FALSE;

   /* process key press */
   while (!done)
   {
      /* dump the keyboard input buffer */
      dump_key_buffer();

      /* wait for a key press */
      key = ki();

      /* process key */
      /* Note: ki() returns the negative of the key scan code  */
      /* if an extended code key is pressed (ie: UP, DOWN etc) */
      /* and returns the ASCII code directly otherwise         */
      switch (-key)
      {
         /* LEFT, UP, Home and PgUp highlight traversal (INTERIOR) */
         case K_LEFT   : ;
         case K_UP     : ;
         case K_HOME   : ;
         case K_PGUP   : if (current_type != INTERIOR)
                         {
                            unhighlight_traversal_setting(current_type);
                            current_type = INTERIOR;
                            highlight_traversal_setting(current_type);
                         }
```

```
                break;

/* RIGHT, DOWN, End and PgDn highlight traversal (EXTERIOR) */
case K_RIGHT  : ;
case K_DN     : ;
case K_END    : ;
case K_PGDN   : if (current_type != EXTERIOR)
                {
                   unhighlight_traversal_setting(current_type);
                   current_type = EXTERIOR;
                   highlight_traversal_setting(current_type);
                }
                break;

/* ESCAPE key exits with current type */
case -K_ESC    : done = TRUE;
                 break;

/* ENTER key sets the current type and exits */
case -K_ENTER  : if (current_type != marked_type)
                 {
                    unmark_traversal(marked_type);
                    set_ASD_area_type(area_num, current_type);
                    mark_traversal(current_type);
                    marked_type = current_type;
                    highlight_traversal_setting(current_type);
                 }
                 done = TRUE;
                 break;

/* I and i keys */
/* set traversal to (INTERIOR) and exit */
case -'I'      : ;
case -'i'      : if (current_type != INTERIOR)
                 {
                    unhighlight_traversal_setting(current_type);
                    current_type = INTERIOR;
                 }
                 if (current_type != marked_type)
                 {
                    unmark_traversal(marked_type);
                    set_ASD_area_type(area_num, current_type);
                    mark_traversal(current_type);
                    marked_type = current_type;
                 }
                 highlight_traversal_setting(current_type);
                 done = TRUE;
                 break;

/* E and e keys */
/* set traversal to (EXTERIOR) and exit */
case -'E'      : ;
case -'e'      : if (current_type != EXTERIOR)
                 {
```

```
                        unhighlight_traversal_setting(current_type);
                        current_type = EXTERIOR;
                     }
                     if (current_type != marked_type)
                     {
                        unmark_traversal(marked_type);
                        set_ASD_area_type(area_num, current_type);
                        mark_traversal(current_type);
                        marked_type = current_type;
                     }
                     highlight_traversal_setting(current_type);
                     done = TRUE;
                     break;

            default        : bell();
                             break;
         }
      }

      unhighlight_traversal_setting(current_type);

      /* if the type was changed then set the vulnerability false */
      if (marked_type != start_type)
         set_vulnerability_current(FALSE);

      return(0);
}

/* ------------------------------------------------------------------------ */

savi_modify_area_quit()

{
   /* return a 1 to jump up one menu level */
   return(1);
}

/* ********************** end SAVIAREA.C source file ********************** */

/* ************************ SAVI_ASD.C source file ************************ */

#include <savidefs.h>
#include <malloc.h>
#include <stdio.h>

/* ------------------------------------------------------------------------ */

char error_message[MAX_ERROR_STRING_LEN];

ASD_STRUCT *current_ASD;

/* full_area_label is temporary storage for the     */
/* two-line area labels merged into a single string */
char full_area_label[MAX_AREA_LABEL_LEN];
```

```
/* ------------------------------------------------------------------------ */

/* allocate_ASD allocates enough memory for the current ASD and sets the    */
/* current ASD pointer                                                      */

allocate_ASD()

{
   alloc_ASD(&current_ASD);

   return;
}

/* ------------------------------------------------------------------------ */

/* alloc_ASD allocates enough memory for a single ASD and sets the given ASD */
/* pointer.  If not enough memory is available, an error message is output   */
/* to the screen and the code is exited.                                     */

alloc_ASD(ASD_ptr_ptr)
ASD_STRUCT * (*ASD_ptr_ptr);

{
   char *alloc;

   alloc = calloc(1,sizeof(ASD_STRUCT));
   if (alloc == 0l)
   {
      /* create the error message */
      sprintf(error_message,"Insufficient Memory for ASD");
      cls();
      popup_fatal_error_window(error_message);
      user_acknowledge_wait();
      remove_fatal_error_window();
      display_quit();
      exit(0);
   }
   else
      *ASD_ptr_ptr = (ASD_STRUCT *)alloc;

   return;

}

/* ------------------------------------------------------------------------ */

/* get_ASD_PE_ptr returns a pointer to the PE within the current_ASD defined */
/* by layer and PE_num. Note that the task PE is returned if layer is passed */
/* in as MAX_LAYERS_PER_ASD                                                  */

PROT_ELEM *get_ASD_PE_ptr(layer,PE_num)
int layer, PE_num;
```

```
{
   if (layer == MAX_LAYERS_PER_ASD)
      return(&(current_ASD->task_PE));
   else
      return(&(current_ASD->model[layer][PE_num]));
}

/* ------------------------------------------------------------------------ */

/* get_ASD_task_PE_ptr returns a pointer to the task PE within the current  */
/* ASD                                                                      */

PROT_ELEM *get_ASD_task_PE_ptr()

{
   return(&(current_ASD->task_PE));
}

/* ------------------------------------------------------------------------ */

/* get_ASD_label returns the current_ASD label                              */

char *get_ASD_label()

{
   return(current_ASD->label);
}

/* ------------------------------------------------------------------------ */

/* set_ASD_label copies the given new_ASD_label into the current ASD's label */

set_ASD_label(new_ASD_label)
char *new_ASD_label;

{
   strcpy(current_ASD->label,new_ASD_label);

   return;
}

/* ------------------------------------------------------------------------ */

/* get_ASD_area_dist returns the distance in meters across the given area in */
/* the current ASD                                                          */
/* Note: area_num must be in range from 1 to MAX_MODIFIABLE_AREAS           */

get_ASD_area_dist(area_num)
int area_num;

{
   return(current_ASD->area_dist[area_num-1]);
}
```

```
/* ------------------------------------------------------------------------ */

/* set_ASD_area_dist sets the distance in meters across the given area in   */
/* the current ASD                                                          */
/* Note: area_num must be in range from 1 to MAX_MODIFIABLE_AREAS           */

set_ASD_area_dist(area_num,dist)
int area_num, dist;

{
   current_ASD->area_dist[area_num-1] = dist;
}

/* ------------------------------------------------------------------------ */

/* get_ASD_area_type returns the type (EXTERIOR or INTERIOR) of the given   */
/* area in the current ASD                                                  */
/* Note: area_num must be in range from 1 to MAX_MODIFIABLE_AREAS           */

get_ASD_area_type(area_num)
int area_num;

{
   return(current_ASD->area_type[area_num-1]);
}

/* ------------------------------------------------------------------------ */

/* set_ASD_area_type sets the type (EXTERIOR or INTERIOR) of the given      */
/* area in the current ASD                                                  */
/* Note: area_num must be in range from 1 to MAX_MODIFIABLE_AREAS           */

set_ASD_area_type(area_num,type)
int area_num, type;

{
   current_ASD->area_type[area_num-1] = type;
}

/* ------------------------------------------------------------------------ */

/* get_ASD_layer_bypass returns TRUE if the given layer has no present PEs  */
/* on the current ASD and FALSE otherwise                                   */

get_ASD_layer_bypass(layer)
int layer;

{
   /* note: the target task layer cannot be bypassed */
   if (layer == MAX_LAYERS_PER_ASD)
      return(FALSE);
   else
      return(current_ASD->bypasses[layer]);
}
```

```
/* ------------------------------------------------------------------------ */

/* set_ASD_layer_bypass sets the given layer's bypass value to val in the    */
/* current ASD (val must be TRUE or FALSE)                                   */

set_ASD_layer_bypass(layer,val)
int val;

{
   current_ASD->bypasses[layer] = val;

   return;
}

/* ------------------------------------------------------------------------ */

/* get_num_bypasses returns the number of bypassed layers in the current ASD */

get_num_bypasses()

{
   int bypass_count, layer;

   bypass_count = 0;

   for (layer = 0; layer < MAX_LAYERS_PER_ASD; layer++)
   {
      if (get_ASD_layer_bypass(layer))
         bypass_count++;
   }

   return(bypass_count);
}

/* ------------------------------------------------------------------------ */

/* get_ASD_rft returns the current ASD's given response force time           */

get_ASD_rft(rft_num)
int rft_num;

{
   return(current_ASD->rft_table[rft_num]);
}

/* ------------------------------------------------------------------------ */

/* set_ASD_rft sets the current ASD's given response force time to rft       */

set_ASD_rft(rft_num, rft)
int rft_num, rft;

{
```

```
   current_ASD->rft_table[rft_num] = rft;

   return;
}

/* ------------------------------------------------------------------------ */

/* get_num_rfts returns the number of RFTs in the current ASD's RFT table    */

get_num_rfts()

{
   return(current_ASD->num_rfts);
}

/* ------------------------------------------------------------------------ */

/* set_num_rfts sets the number of RFTs in the current ASD's RFT table       */
/* Note: num_rfts must be between 1 and MAX_RFTS                             */

set_num_rfts(num_rfts)
int num_rfts;

{
   current_ASD->num_rfts = num_rfts;

   return;
}

/* ------------------------------------------------------------------------ */

/* get_ASD_rft_table_ptr returns a pointer to the current ASD rft_table      */

int *get_ASD_rft_table_ptr()

{
   return(current_ASD->rft_table);
}

/* ------------------------------------------------------------------------ */

/* get_analysis_state returns the analysis state for the current ASD         */

get_analysis_state()

{
   return(current_ASD->analysis_state);
}

/* ------------------------------------------------------------------------ */

/* set_analysis_state sets the analysis state for the current ASD            */
/* analysis_state should be set TRUE if the current ASD has been analyzed    */
```

```
set_analysis_state(state)
int state;

{
   current_ASD->analysis_state = state;

   return;
}

/* ------------------------------------------------------------------------ */

/* get_ASD_threshold_pi returns the current ASD's threshold probability of  */
/* interruption                                                             */

float get_ASD_threshold_pi()

{
   return(current_ASD->threshold_pi);
}

/* ------------------------------------------------------------------------ */

/* set_ASD_threshold_pi sets the current ASD's threshold probability of     */
/* interruption to pi                                                       */

set_ASD_threshold_pi(pi)
float pi;

{
   current_ASD->threshold_pi = pi;

   return;
}

/* ------------------------------------------------------------------------ */

/* get_current_threat returns the current threat number for the current ASD */

get_current_threat()

{
   return(current_ASD->current_threat);
}

/* ------------------------------------------------------------------------ */

/* set_current_threat sets the current threat to the given threat number.   */
/* The threat number corresponds to the index of the desired threat in the  */
/* threat table                                                             */

set_current_threat(threat_num)
int threat_num;

{
```

```
   current_ASD->current_threat = threat_num;

   return;
}

/* ------------------------------------------------------------------------ */

/* get_current_direction returns the current analysis direction (ENTRY/EXIT) */
/* for the current ASD                                                       */

get_current_direction()

{
   return(current_ASD->current_direction);
}

/* ------------------------------------------------------------------------ */

/* set_current_direction sets the current analysis direction (ENTRY or EXIT) */
/* for the current ASD                                                       */

set_current_direction(direction)
int direction;

{
   current_ASD->current_direction = direction;

   return;
}

/* ------------------------------------------------------------------------ */

/* get_current_strategy returns the current ASD strategy setting.  This     */
/* indicates whether the user desires FORCE or OPTIMAL strategy analysis     */

get_current_strategy()

{
   return(current_ASD->current_strategy);
}

/* ------------------------------------------------------------------------ */

/* set_current_strategy sets the current ASD strategy setting.  This         */
/* indicates whether the user desires FORCE or OPTIMAL strategy analysis     */

set_current_strategy(strategy)
int strategy;

{
   current_ASD->current_strategy = strategy;

   return;
}
```

```
/* ------------------------------------------------------------------------ */

/* get_comp_choice_num returns the user selected choice number for the      */
/* given component (comp_num) in the list of components of the given type   */
/* (comp_type = DELAY or NONDET) for the PE at the given layer and PE_num in */
/* the current ASD.  This value is either CLEAR, DIRECT or indicates the    */
/* choice number within the given component's choices in the associated     */
/* component catalog.                                                       */

get_comp_choice_num(comp_type, comp_num, layer, PE_num)
int comp_type, comp_num, layer, PE_num;

{
   if (comp_type == DELAY)
      return(get_ASD_PE_ptr(layer,PE_num)->
             del_comp_settings[comp_num].choice);
   else
      return(get_ASD_PE_ptr(layer,PE_num)->
             nondet_comp_settings[comp_num].choice);
}

/* ------------------------------------------------------------------------ */

/* set_comp_choice_num sets the user selected choice number for the         */
/* given component (comp_num) in the list of components of the given type   */
/* (comp_type = DELAY or NONDET) for the PE at the given layer and PE_num in */
/* the current ASD.  This value is either CLEAR, DIRECT or indicates the    */
/* choice number within the given component's choices in the associated     */
/* component catalog.                                                       */

set_comp_choice_num(comp_type, comp_num, layer, PE_num, choice_setting)
int comp_type, comp_num, layer, PE_num, choice_setting;

{
   if (comp_type == DELAY)
      get_ASD_PE_ptr(layer,PE_num)->
        del_comp_settings[comp_num].choice = choice_setting;
   else
      get_ASD_PE_ptr(layer,PE_num)->
        nondet_comp_settings[comp_num].choice = choice_setting;
}

/* ------------------------------------------------------------------------ */

/* all_comp_choices_made returns TRUE if all component choices (both delay  */
/* and detection) have been made (ie: are not CLEAR) for every present PE in */
/* the current ASD.  Returns FALSE otherwise.                               */

all_comp_choices_made()

{
   int layer, PE_num;
```

```
   for (layer = 0; layer < MAX_LAYERS_PER_ASD; layer++)
   {
      if (!get_ASD_layer_bypass(layer))
      {
         for (PE_num = 0; PE_num < MAX_PE_PER_LAYER; PE_num++)
         {
            if (get_ASD_PE_ptr(layer, PE_num)->present)
            {
               if (any_PE_comp_type_choices_are_val(layer,PE_num,
                                                    DELAY,CLEAR) ||
                   any_PE_comp_type_choices_are_val(layer,PE_num,
                                                    NONDET,CLEAR))
                  return(FALSE);
            }
         }
      }
   }

   /* check if any choices have yet to be made in the task PE */
   layer = MAX_LAYERS_PER_ASD;
   PE_num = 0;
   if (any_PE_comp_type_choices_are_val(layer,PE_num,DELAY,CLEAR) ||
       any_PE_comp_type_choices_are_val(layer,PE_num,NONDET,CLEAR))
      return(FALSE);

   return(TRUE);
}

/* ------------------------------------------------------------------------ */

/* any_PE_comp_type_choices_are_val returns TRUE if any component choices of */
/* the given type (DELAY or NONDET) are set to the given value (val) within  */
/* the PE specified by the given layer and PE_num in the current ASD         */

any_PE_comp_type_choices_are_val(layer,PE_num,comp_type,val)
int layer, PE_num, comp_type, val;

{
   int num_comps, comp_num, found_val;

   num_comps = get_number_of_comps(comp_type,
                                   get_ASD_PE_ptr(layer,PE_num)->type);

   found_val = FALSE;
   comp_num = 0;

   while (!found_val && comp_num < num_comps)
   {
      if (get_comp_choice_num(comp_type,comp_num,layer,PE_num) == val)
         found_val = TRUE;

      comp_num++;
   }
```

```
   return(found_val);

}

/* ------------------------------------------------------------------------ */

/* number_of_PEs_present returns the number of present PEs on the given     */
/* layer within the current ASD                                             */

number_of_PEs_present(layer)
int layer;

{
   PROT_ELEM *PE_ptr;
   int PE_num, PE_count;

   /* if the layer is bypassed, no PEs are present */
   if (get_ASD_layer_bypass(layer)) return(0);

   /* if layer is the task layer, the task PE is the only PE */
   if (layer == MAX_LAYERS_PER_ASD) return(1);

   PE_count = 0;

   for (PE_num = 0; PE_num < MAX_PE_PER_LAYER; PE_num++)
   {
      PE_ptr = get_ASD_PE_ptr(layer,PE_num);
      if (PE_ptr->present) PE_count++;
   }

   return(PE_count);
}

/* ------------------------------------------------------------------------ */

/* unreachable_PEs_present returns TRUE if any PEs in the current ASD cannot */
/* be reached.  If TRUE is returned, the variables which start_area_ptr     */
/* and end_area_ptr point to are loaded with the area numbers which bracket  */
/* the unreachable PEs found.                                               */

unreachable_PEs_present(start_area_ptr, end_area_ptr)
int *start_area_ptr, *end_area_ptr;

{
   int layer, PE_num;

   for (layer = 0; layer < MAX_MODIFIABLE_AREAS-1; layer++)
   {
      if (!get_ASD_layer_bypass(layer) && !layer_directly_connected(layer))
      {
         for (PE_num = 0; PE_num < MAX_PE_PER_LAYER; PE_num++)
         {
            if (PE_strands_PEs(layer, PE_num))
            {
```

```
                *start_area_ptr = layer+1;
                *end_area_ptr = get_ASD_PE_ptr(layer,PE_num)->output_area;
                return(TRUE);
             }
          }
       }
    }

    return(FALSE);
}

/* ------------------------------------------------------------------------ */

/* layer_directly_connected returns TRUE if the given layer has any PEs      */
/* which directly connect the bracketing areas.  FALSE is returned otherwise */

layer_directly_connected(layer)
int layer;

{
    PROT_ELEM *PE_ptr;
    int PE_num;

    for (PE_num = 0; PE_num < MAX_PE_PER_LAYER; PE_num++)
    {
       PE_ptr = get_ASD_PE_ptr(layer,PE_num);
       if (PE_ptr->present && (PE_ptr->output_area == layer+1))
          return(TRUE);
    }

    return(FALSE);
}

/* ------------------------------------------------------------------------ */

/* PE_strands_PEs returns TRUE if the given PE jumps over non-bypassed      */
/* layers.  FALSE is returned otherwise                                     */

PE_strands_PEs(layer,PE_num)
int layer, PE_num;

{
    PROT_ELEM *PE_ptr;
    int layer_num;

    PE_ptr = get_ASD_PE_ptr(layer,PE_num);

    if (PE_ptr->present && (PE_ptr->output_area != layer+1))
    {
       for (layer_num = layer+1; layer_num < PE_ptr->output_area; layer_num++)
          if (!get_ASD_layer_bypass(layer_num))
             return(TRUE);
    }
```

```
   return(FALSE);
}

/* ------------------------------------------------------------------------ */

/* copy_PE_settings copys all PE settings from PE 1 to PE 2 specified by     */
/* the given layers and PE_nums.                                             */

copy_PE_settings(layer1,PE_num1,layer2,PE_num2)
int layer1, PE_num1, layer2, PE_num2;

{
   int comp_num;
   PROT_ELEM *PE_ptr1;
   PROT_ELEM *PE_ptr2;

   PE_ptr1 = get_ASD_PE_ptr(layer1,PE_num1);
   PE_ptr2 = get_ASD_PE_ptr(layer2,PE_num2);

   /* copy the PE type */
   PE_ptr2->type = PE_ptr1->type;

   /* copy the delay and non-detection component settings */
   for (comp_num = 0; comp_num < MAX_COMP_PER_PE; comp_num++)
   {
      PE_ptr2->del_comp_settings[comp_num].choice =
         PE_ptr1->del_comp_settings[comp_num].choice;
      PE_ptr2->del_comp_settings[comp_num].direct_setting =
         PE_ptr1->del_comp_settings[comp_num].direct_setting;
      PE_ptr2->del_comp_settings[comp_num].panic_bar =
         PE_ptr1->del_comp_settings[comp_num].panic_bar;

      PE_ptr2->nondet_comp_settings[comp_num].choice =
         PE_ptr1->nondet_comp_settings[comp_num].choice;
      PE_ptr2->nondet_comp_settings[comp_num].direct_setting =
         PE_ptr1->nondet_comp_settings[comp_num].direct_setting;
      PE_ptr2->nondet_comp_settings[comp_num].panic_bar =
         PE_ptr1->nondet_comp_settings[comp_num].panic_bar;
   }

   /* copy the transit distance */
   PE_ptr2->transit_dist = PE_ptr1->transit_dist;

   return;
}

/* ------------------------------------------------------------------------ */

/* clear_PE_settings clears the PE settings for the PE specified by the      */
/* given layer and PE_num.  All delay values are set to 0 and all non-       */
/* detection probabilities are set to 1 (ie: open PE with no protection)     */
/* All delay and non-detection component type/choice pairs are set to        */
/* CLEAR to indicate that no components have been selected for this PE       */
```

```
clear_PE_settings(layer,PE_num)
int layer, PE_num;

{
   int comp_num;
   PROT_ELEM *PE_ptr;

   PE_ptr = get_ASD_PE_ptr(layer,PE_num);

   /* set all delay values to zero */
   PE_ptr->user_delay_in  = 0.0;
   PE_ptr->user_delay_out = 0.0;

   /* set all non-detection values to one */
   PE_ptr->user_nondet_in  = 1.0;
   PE_ptr->user_nondet_out = 1.0;

   /* set the delay and non-detection components to CLEAR */
   for (comp_num = 0; comp_num < MAX_COMP_PER_PE; comp_num++)
   {
      PE_ptr->del_comp_settings[comp_num].choice = CLEAR;
      PE_ptr->del_comp_settings[comp_num].direct_setting = 0.0;
      PE_ptr->del_comp_settings[comp_num].panic_bar = FALSE;

      PE_ptr->nondet_comp_settings[comp_num].choice = CLEAR;
      PE_ptr->nondet_comp_settings[comp_num].direct_setting = 1.0;
      PE_ptr->nondet_comp_settings[comp_num].panic_bar = FALSE;
   }

   /* set the transit distance to zero */
   PE_ptr->transit_dist = 0;

   return;
}

/* ------------------------------------------------------------------------ */

/* clear_all_PE_choices sets all component choices to CLEAR for each PE in  */
/* the current ASD                                                           */

clear_all_PE_choices()

{
   int layer, PE_num;

   for (layer = 0; layer < MAX_LAYERS_PER_ASD; layer++)
   {
      for (PE_num = 0; PE_num < MAX_PE_PER_LAYER; PE_num++)
      {
         clear_PE_delay_choices(layer,PE_num);
         clear_PE_nondet_choices(layer,PE_num);
      }
   }
```

```
   /* clear the task PE component choices */
   clear_PE_delay_choices(MAX_LAYERS_PER_ASD,0);
   clear_PE_nondet_choices(MAX_LAYERS_PER_ASD,0);

   return;
}

/* ------------------------------------------------------------------------ */

/* clear_PE_delay_choices sets all delay component choices to CLEAR for the  */
/* given PE in the current ASD                                                */

clear_PE_delay_choices(layer,PE_num)
int layer,PE_num;

{
   int comp_num, num_comps, PE_type;

   PE_type = get_ASD_PE_ptr(layer,PE_num)->type;

   num_comps = get_number_of_comps(DELAY,PE_type);

   for (comp_num = 0; comp_num < num_comps; comp_num++)
   {
      set_comp_choice_num(DELAY, comp_num, layer, PE_num, CLEAR);
   }

   return;
}

/* ------------------------------------------------------------------------ */

/* clear_PE_nondet_choices sets all detection component choices to CLEAR for */
/* the given PE in the current ASD                                            */

clear_PE_nondet_choices(layer,PE_num)
int layer,PE_num;

{
   int comp_num, num_comps, PE_type;

   PE_type = get_ASD_PE_ptr(layer,PE_num)->type;

   num_comps = get_number_of_comps(NONDET,PE_type);

   for (comp_num = 0; comp_num < num_comps; comp_num++)
   {
      set_comp_choice_num(NONDET, comp_num, layer, PE_num, CLEAR);
   }

   return;
}

/* ------------------------------------------------------------------------ */
```

```
/* get_full_ASD_area_label returns the full ASD area label for the area      */
/* area_num. area_num must be in the range (0 to (MAX_AREAS_PER_ASD-1))      */

char *get_full_ASD_area_label(area_num)
int area_num;

{
   static char *fixed_area_catalog[2] =
      {"OFFSITE",
       "TARGET"
      };

   switch (area_num)
   {
      case 0 : return(fixed_area_catalog[0]);

      case MAX_AREAS_PER_ASD-1 : return(fixed_area_catalog[1]);

      /* create a full area label by concatenating the first     */
      /* and second lines of the area label with a space between */
      default : strcpy(full_area_label,
                       get_ASD_area_label_first_line(area_num));
                strcat(full_area_label," ");
                strcat(full_area_label,
                       get_ASD_area_label_second_line(area_num));
                return(full_area_label);
   }

}

/* ------------------------------------------------------------------------ */

/* get_ASD_area_label_first_line returns the first line of the ASD area      */
/* label for the given area.                                                 */
/* area_num must be in the range (1 to (MAX_AREAS_PER_ASD-2))                */

char *get_ASD_area_label_first_line(area_num)
int area_num;

{
   switch (area_num)
   {
      case 0 : break;

      case MAX_AREAS_PER_ASD-1 : break;

      default : return(current_ASD->area_labels[area_num-1].first_line);
   }

   return;

}
```

```
/* ------------------------------------------------------------------------ */

/* get_ASD_area_label_second_line returns the second line of the ASD area   */
/* label for the given area.                                                */
/* area_num must be in the range (1 to (MAX_AREAS_PER_ASD-2))               */

char *get_ASD_area_label_second_line(area_num)
int area_num;

{
   switch (area_num)
   {
      case 0 : break;

      case MAX_AREAS_PER_ASD-1 : break;

      default : return(current_ASD->area_labels[area_num-1].second_line);
   }

   return;

}

/* ------------------------------------------------------------------------ */

/* set_ASD_area_label_first_line sets the first line of the ASD area label  */
/* for the given area to the given area string. area_num must be in the     */
/* range (1 to (MAX_AREAS_PER_ASD-2)) and the string must be no longer than */
/* MAX_HALF_AREA_LABEL_LEN                                                  */
/* Note: any attempt to set the first or last area labels has no effect     */

set_ASD_area_label_first_line(area_num,area_string)
int area_num;
char *area_string;

{
   switch (area_num)
   {
      case 0 : break;

      case MAX_AREAS_PER_ASD-1 : break;

      default : strcpy(current_ASD->area_labels[area_num-1].first_line,
                       area_string);
   }

   return;

}

/* ------------------------------------------------------------------------ */

/* set_ASD_area_label_second_line sets the second line of the ASD area label */
/* for the given area to the given area string. area_num must be in the     */
```

```
/* range (1 to (MAX_AREAS_PER_ASD-2)) and the string must be no longer than  */
/* MAX_HALF_AREA_LABEL_LEN                                                   */
/* Note: any attempt to set the first or last area labels has no effect      */

set_ASD_area_label_second_line(area_num,area_string)
int area_num;
char *area_string;

{
   switch (area_num)
   {
      case 0 : break;

      case MAX_AREAS_PER_ASD-1 : break;

      default : strcpy(current_ASD->area_labels[area_num-1].second_line,
                       area_string);
   }

   return;

}

/* ------------------------------------------------------------------------- */

/* save_current_ASD saves the current ASD to disk file file_name.            */

save_current_ASD(file_name)
char *file_name;

{
   save_ASD(file_name,current_ASD);
   set_current_ASD_saved(TRUE);

   return;
}

/* ------------------------------------------------------------------------- */

/* save_ASD saves the ASD pointed to by ASD_ptr to disk file file_name.      */

save_ASD(file_name,ASD_ptr)
char *file_name;
ASD_STRUCT *ASD_ptr;

{
   FILE *stream;

   stream = fopen(file_name,"w+b");
   write_ASD(stream,ASD_ptr);
   /* if analysis results are current then save them */
   /* at the end of the ASD settings in the ASD file */
   if (get_analysis_state())
      save_results(stream);
```

```
   fclose(stream);

   return;

}

/* ------------------------------------------------------------------------ */

write_ASD(stream,ASD_ptr)
FILE *stream;
ASD_STRUCT *ASD_ptr;

{
   int float_size, int_size;
   int layer, PE_num, label_num, area_num;
   float version;

   float_size = sizeof(float);
   int_size = sizeof(int);

   version = SAVI_VERSION;

   /* write the current SAVI Version Number */
   fwrite((char *)&version,float_size,1,stream);

   /* write ASD label */
   fwrite(ASD_ptr->label,1,MAX_ASD_LABEL_LEN,stream);

   /* write the area transit distances */
   for (area_num = 0; area_num < MAX_MODIFIABLE_AREAS; area_num++)
   {
      fwrite((char *)&(ASD_ptr->area_dist[area_num]),int_size,1,stream);
   }

   /* write the area types */
   for (area_num = 0; area_num < MAX_MODIFIABLE_AREAS; area_num++)
   {
      fwrite((char *)&(ASD_ptr->area_type[area_num]),int_size,1,stream);
   }

   /* write the modifiable area labels */
   for (label_num = 0; label_num < MAX_MODIFIABLE_AREAS; label_num++)
   {
      fwrite(ASD_ptr->area_labels[label_num].first_line,1,
             MAX_HALF_AREA_LABEL_LEN,stream);
      fwrite(ASD_ptr->area_labels[label_num].second_line,1,
             MAX_HALF_AREA_LABEL_LEN,stream);
   }

   /* write each PE */
   for (layer = 0; layer < MAX_LAYERS_PER_ASD; layer++)
   {
      for (PE_num = 0; PE_num < MAX_PE_PER_LAYER; PE_num++)
      {
```

```
            write_PE(stream,&(ASD_ptr->model[layer][PE_num]));
         }
      }

      /* write the task PE */
      write_PE(stream,&(ASD_ptr->task_PE));

      /* write the bypasses array */
      for (layer = 0; layer < MAX_LAYERS_PER_ASD; layer++)
      {
         fwrite((char *)&(ASD_ptr->bypasses[layer]),int_size,1,stream);
      }

      /* write the analysis state */
      fwrite((char *)&(ASD_ptr->analysis_state),int_size,1,stream);

      /* write the Response Force Time table */
      fwrite((char *)ASD_ptr->rft_table,int_size,MAX_RFTS,stream);

      /* write the number of RFTS in table */
      fwrite((char *)&(ASD_ptr->num_rfts),int_size,1,stream);

      /* write the threshold Probability of Interruption */
      fwrite((char *)&(ASD_ptr->threshold_pi),float_size,1,stream);

      /* write the current_threat number */
      fwrite((char *)&(ASD_ptr->current_threat),int_size,1,stream);

      /* write the current_direction */
      fwrite((char *)&(ASD_ptr->current_direction),int_size,1,stream);

      /* write the current_strategy */
      fwrite((char *)&(ASD_ptr->current_strategy),int_size,1,stream);

      return;
   }

   /* ------------------------------------------------------------------------ */

   write_PE(stream,PE_ptr)

   FILE *stream;
   PROT_ELEM *PE_ptr;

   {
      int int_size, float_size, comp_num;

      int_size = sizeof(int);
      float_size = sizeof(float);

      fwrite((char *)&(PE_ptr->present),int_size,1,stream);
      fwrite((char *)&(PE_ptr->type),int_size,1,stream);

      fwrite((char *)&(PE_ptr->user_delay_in),float_size,1,stream);
```

```
   fwrite((char *)&(PE_ptr->user_nondet_in),float_size,1,stream);
   fwrite((char *)&(PE_ptr->user_delay_out),float_size,1,stream);
   fwrite((char *)&(PE_ptr->user_nondet_out),float_size,1,stream);

   for (comp_num = 0; comp_num < MAX_COMP_PER_PE; comp_num++)
   {
      fwrite((char *)&(PE_ptr->del_comp_settings[comp_num].choice),
             int_size,1,stream);
      fwrite((char *)&(PE_ptr->del_comp_settings[comp_num].direct_setting),
             float_size,1,stream);
      fwrite((char *)&(PE_ptr->del_comp_settings[comp_num].panic_bar),
             int_size,1,stream);
      fwrite((char *)&(PE_ptr->nondet_comp_settings[comp_num].choice),
             int_size,1,stream);
      fwrite((char *)&(PE_ptr->nondet_comp_settings[comp_num].direct_setting),
             float_size,1,stream);
      fwrite((char *)&(PE_ptr->nondet_comp_settings[comp_num].panic_bar),
             int_size,1,stream);
   }

   fwrite((char *)&(PE_ptr->input_area),int_size,1,stream);
   fwrite((char *)&(PE_ptr->output_area),int_size,1,stream);
   fwrite((char *)&(PE_ptr->transit_dist),int_size,1,stream);

   return;

}

/* ------------------------------------------------------------------------ */

/* retrieve_current_ASD reads the current ASD from disk file file_name.     */

retrieve_current_ASD(file_name)
char *file_name;

{
   retrieve_ASD(file_name,current_ASD);
   set_current_ASD_saved(TRUE);

   return;
}

/* ------------------------------------------------------------------------ */

/* retrieve_ASD retrieves the ASD contained in disk file file_name and      */
/* stores the data in the ASD pointed to by ASD_ptr.                        */

retrieve_ASD(file_name,ASD_ptr)

char *file_name;
ASD_STRUCT *ASD_ptr;

{
   FILE *stream;
```

```
   int test;

   stream = fopen(file_name,"r+b");
   read_ASD(stream,ASD_ptr);
   /* check if analysis results follow on ASD file */
   if (get_analysis_state())
   {
      test = load_results(stream);
      if (test == OUT_OF_MEMORY)
      {
         /* insufficient room for analysis results */
         set_analysis_state(FALSE);
      }
      else if (test == READ_FAILURE)
      {
         /* analysis state flag was incorrect */
         /* no analysis results were found    */
         set_analysis_state(FALSE);
      }
      else
      {
         /* set the default path number and rft number */
         /* for the analysis display results functions */

         /* set the current path to the most vulnerable */
         set_current_path(0);

         /* set the default rft num to the maximum rft */
         set_current_rft_num(get_num_rfts()-1);
      }
   }
   fclose(stream);

   return;
}

/* ------------------------------------------------------------------------ */

read_ASD(stream,ASD_ptr)
FILE *stream;
ASD_STRUCT *ASD_ptr;

{
   int float_size, int_size;
   int layer, PE_num, label_num, area_num;
   float version;

   float_size = sizeof(float);
   int_size = sizeof(int);

   /* read the current SAVI Version Number */
   fread((char *)&version,float_size,1,stream);

   /* read ASD label */
```

```
fread(ASD_ptr->label,1,MAX_ASD_LABEL_LEN,stream);

/* read the area transit distances */
for (area_num = 0; area_num < MAX_MODIFIABLE_AREAS; area_num++)
{
   fread((char *)&(ASD_ptr->area_dist[area_num]),int_size,1,stream);
}

/* read the area types */
for (area_num = 0; area_num < MAX_MODIFIABLE_AREAS; area_num++)
{
   fread((char *)&(ASD_ptr->area_type[area_num]),int_size,1,stream);
}

/* read the modifiable area labels */
for (label_num = 0; label_num < MAX_MODIFIABLE_AREAS; label_num++)
{
   fread(ASD_ptr->area_labels[label_num].first_line,1,
         MAX_HALF_AREA_LABEL_LEN,stream);
   fread(ASD_ptr->area_labels[label_num].second_line,1,
         MAX_HALF_AREA_LABEL_LEN,stream);
}

/* read each PE */
for (layer = 0; layer < MAX_LAYERS_PER_ASD; layer++)
{
   for (PE_num = 0; PE_num < MAX_PE_PER_LAYER; PE_num++)
   {
      read_PE(stream,&(ASD_ptr->model[layer][PE_num]));
   }
}

/* read the task PE */
read_PE(stream,&(ASD_ptr->task_PE));

/* read the bypasses array */
for (layer = 0; layer < MAX_LAYERS_PER_ASD; layer++)
{
   fread((char *)&(ASD_ptr->bypasses[layer]),int_size,1,stream);
}

/* read the analysis state */
fread((char *)&(ASD_ptr->analysis_state),int_size,1,stream);

/* read the Response Force Time table */
fread((char *)ASD_ptr->rft_table,int_size,MAX_RFTS,stream);

/* read the number of RFTS in table */
fread((char *)&(ASD_ptr->num_rfts),int_size,1,stream);

/* read the threshold Probability of Interruption */
fread((char *)&(ASD_ptr->threshold_pi),float_size,1,stream);

/* read the current_threat number */
```

```
   fread((char *)&(ASD_ptr->current_threat),int_size,1,stream);

   /* read the current_direction */
   fread((char *)&(ASD_ptr->current_direction),int_size,1,stream);

   /* read the current_strategy */
   fread((char *)&(ASD_ptr->current_strategy),int_size,1,stream);

   return;
}

/* ------------------------------------------------------------------------ */

read_PE(stream,PE_ptr)

FILE *stream;
PROT_ELEM *PE_ptr;

{
   int int_size, float_size, comp_num;

   int_size = sizeof(int);
   float_size = sizeof(float);

   fread((char *)&(PE_ptr->present),int_size,1,stream);
   fread((char *)&(PE_ptr->type),int_size,1,stream);

   fread((char *)&(PE_ptr->user_delay_in),float_size,1,stream);
   fread((char *)&(PE_ptr->user_nondet_in),float_size,1,stream);
   fread((char *)&(PE_ptr->user_delay_out),float_size,1,stream);
   fread((char *)&(PE_ptr->user_nondet_out),float_size,1,stream);

   for (comp_num = 0; comp_num < MAX_COMP_PER_PE; comp_num++)
   {
      fread((char *)&(PE_ptr->del_comp_settings[comp_num].choice),
            int_size,1,stream);
      fread((char *)&(PE_ptr->del_comp_settings[comp_num].direct_setting),
            float_size,1,stream);
      fread((char *)&(PE_ptr->del_comp_settings[comp_num].panic_bar),
            int_size,1,stream);
      fread((char *)&(PE_ptr->nondet_comp_settings[comp_num].choice),
            int_size,1,stream);
      fread((char *)&(PE_ptr->nondet_comp_settings[comp_num].direct_setting),
            float_size,1,stream);
      fread((char *)&(PE_ptr->nondet_comp_settings[comp_num].panic_bar),
            int_size,1,stream);
   }

   fread((char *)&(PE_ptr->input_area),int_size,1,stream);
   fread((char *)&(PE_ptr->output_area),int_size,1,stream);
   fread((char *)&(PE_ptr->transit_dist),int_size,1,stream);

   return;
```

```
}

/* ********************** end SAVI_ASD.C source file ********************** */

/* ************************ SAVICALC.C source file ************************ */

#include <savidefs.h>

/* AREA_NODE_NUM_STEP is the step size between area node numbers */
#define AREA_NODE_NUM_STEP 60

/* AUTH_OFFSET is the offset between the unauthorized PE node numbers */
/* and the authorized PE node numbers for PEs on a given layer        */
#define AUTH_OFFSET 25

/* ------------------------------------------------------------------------ */

/* find_node_in_range returns the number of the node with the given node     */
/* value between the start and end nodes (inclusive) on the given path.      */

find_node_in_range(path_ptr, start_node_num, end_node_num, node_value)
PATH_INFO *path_ptr;
int start_node_num, end_node_num, node_value;

{
   int done, node_num;

   node_num = start_node_num;
   done = FALSE;

   while (!done)
   {
      if (path_ptr->nodes[node_num] == node_value)
         done = TRUE;
      else
      {
         node_num++;
         if (node_num > end_node_num)
         {
            node_num = NOT_FOUND;
            done = TRUE;
         }
      }
   }

   return(node_num);
}

/* ------------------------------------------------------------------------ */

/* min_delay_traversal sets the minimum cumulative delay across the input or */
/* output path segment of the given PE into the variable which               */
/* min_delay_value_ptr points to.  The traversal equation number which       */
/* corresponds to this delay is set into the variable which                  */
```

```
/* min_delay_eqn_num_ptr points to.  auth_unauth must be set to AUTH or      */
/* UNAUTH to indicate whether the authorized or unauthorized paths are to be */
/* considered.  Note: at least one traversal equation must exist for the     */
/* given auth_unauth setting. location must be set to INPUT_SEG or           */
/* OUTPUT_SEG to indicate which path segment is to be considered.            */
/* direction_state must be set to ENTRY, EXIT or EXIT_DEP to indicate        */
/* whether the intruder is traversing toward the target, away from the       */
/* target or away from the target along a path segment which was previously  */
/* traversed toward the target, respectively                                 */
/* If the direction_state is EXIT_DEP (exit dependent) then the variable     */
/* which min_delay_eqn_num_ptr points to must contain the number of the      */
/* minimum EXIT equation from a call to this function                        */

min_delay_traversal(layer, PE_num, auth_unauth, location, direction_state,
                    min_delay_value_ptr, min_delay_eqn_num_ptr)
int layer, PE_num, auth_unauth, location, direction_state;
float *min_delay_value_ptr;
int *min_delay_eqn_num_ptr;

{
   SEGMENT_EQN *eqn_ptr;
   PROT_ELEM *PE_ptr;
   int PE_type, min_eqn_num, eqn_num, comp_num, type_num,
       found_last_comp, found_last_eqn, panic_set;
   float min_delay_sum, current_delay_sum, indep_delay_val;

   PE_ptr = get_ASD_PE_ptr(layer, PE_num);
   PE_type = PE_ptr->type;

   if (auth_unauth == AUTH)
   /* authorized traversal */
   {
      if (location == INPUT_SEG)
      /* input path segment traversal */
      {
         switch(direction_state)
         {
            case ENTRY    : eqn_ptr = get_PE_cat_spec_ptr(PE_type)->
                                      authorized_del_eqns.
                                      entry_input_eqns;
                            break;

            case EXIT     : ;
            case EXIT_DEP : eqn_ptr = get_PE_cat_spec_ptr(PE_type)->
                                      authorized_del_eqns.
                                      exit_input_eqns;
                            break;
         }
      }
      /* output path segment traversal */
      else
      {
         switch(direction_state)
         {
```

```
         case ENTRY    : eqn_ptr = get_PE_cat_spec_ptr(PE_type)->
                                   authorized_del_eqns.
                                   entry_output_eqns;
                         break;

         case EXIT     : ;
         case EXIT_DEP : eqn_ptr = get_PE_cat_spec_ptr(PE_type)->
                                   authorized_del_eqns.
                                   exit_output_eqns;
                         break;
      }
   }
}
/* unauthorized traversal */
else
{
   if (location == INPUT_SEG)
   /* input path segment traversal */
   {
      switch(direction_state)
      {
         case ENTRY    : eqn_ptr = get_PE_cat_spec_ptr(PE_type)->
                                   unauthorized_del_eqns.
                                   entry_input_eqns;
                         break;

         case EXIT     : ;
         case EXIT_DEP : eqn_ptr = get_PE_cat_spec_ptr(PE_type)->
                                   unauthorized_del_eqns.
                                   exit_input_eqns;
                         break;
      }
   }
   /* output path segment traversal */
   else
   {
      switch(direction_state)
      {
         case ENTRY    : eqn_ptr = get_PE_cat_spec_ptr(PE_type)->
                                   unauthorized_del_eqns.
                                   entry_output_eqns;
                         break;

         case EXIT     : ;
         case EXIT_DEP : eqn_ptr = get_PE_cat_spec_ptr(PE_type)->
                                   unauthorized_del_eqns.
                                   exit_output_eqns;
                         break;
      }
   }
}

/* use the given minimum delay eqn only to calculate exit dependent sum */
if (direction_state == EXIT_DEP)
```

```
   eqn_num = *min_delay_eqn_num_ptr;
else
   eqn_num = 0;

min_eqn_num = eqn_num;

found_last_eqn = FALSE;

while (!found_last_eqn)
{
   comp_num = 0;
   current_delay_sum = 0;
   found_last_comp = FALSE;

   while (!found_last_comp)
   {
      if (comp_num == MAX_COMP_PER_EQN)
         found_last_comp = TRUE;
      else
      {
         type_num = eqn_ptr[eqn_num].comptype[comp_num];
         if (type_num == EMPTY)
         {
            found_last_comp = TRUE;
            if (comp_num == 0)
            {
               found_last_eqn = TRUE;
            }
         }
         else
         {
            indep_delay_val = find_delay_comp_choice_val(layer, PE_num,
                                                         type_num, location,
                                                         &panic_set,
                                                         get_current_threat());
            /* note that the state of the panic bar setting is valid only */
            /* if a panic bar is possible on the given delay component    */
            current_delay_sum += convert_indep_delay_val(indep_delay_val,
                                    (get_delay_comp_spec_ptr(type_num)->
                                       panic_bar && panic_set),
                                    auth_unauth,
                                    direction_state);
            comp_num++;
         }
      }
   }

   /* completed sum for current equation number */
   if ((eqn_num == 0) || (direction_state == EXIT_DEP) ||
       ((current_delay_sum < min_delay_sum) && !found_last_eqn))
   {
      min_delay_sum = current_delay_sum;
      min_eqn_num = eqn_num;
   }
```

```
      eqn_num++;
      if ((eqn_num == MAX_EQNS_PER_SEGMENT) ||
          (direction_state == EXIT_DEP))
         found_last_eqn = TRUE;
   }

   *min_delay_value_ptr = min_delay_sum;
   *min_delay_eqn_num_ptr = min_eqn_num;

   return;
}

/* ------------------------------------------------------------------------ */

/* find_delay_comp_choice_val returns the delay value (secs) for the given   */
/* component type (type_num) at the given location (INPUT_SEG or OUTPUT_SEG) */
/* within the given PE based on the given threat                             */
/* If the current setting for this component is DIRECT, the direct_setting   */
/* is returned, otherwise the threat-specific independent delay value is     */
/* returned from the delay catalog. Note: If the current component setting   */
/* is CLEAR, then (float)CLEAR is returned                                   */
/* The variable that panic_bar_ptr points to is loaded with the state of the */
/* panic bar setting for the given component at the given location within    */
/* the given PE                                                              */

float find_delay_comp_choice_val(layer, PE_num, type_num, location,
                                 panic_bar_ptr, threat_num)
int layer, PE_num, type_num, location, *panic_bar_ptr, threat_num;

{
   PE_SPECS *PE_spec_ptr;
   PROT_ELEM *PE_ptr;
   int PE_type, done, comp_num, comp_choice, temp_location;

   PE_ptr = get_ASD_PE_ptr(layer, PE_num);
   PE_type = PE_ptr->type;
   PE_spec_ptr = get_PE_cat_spec_ptr(PE_type);

   done = FALSE;
   comp_num = 0;

   while (!done)
   {
      /* find the component type at the given location within the */
      /* corresponding PE catalog delay component list            */
      if (PE_spec_ptr->del_comp_list[comp_num].comptype == type_num)
      {
         temp_location = PE_spec_ptr->del_comp_list[comp_num].location;
         if ( (temp_location == location) || (temp_location == EITHER_SEG) )
         {
            /* found the component in the given PE delay choice list */
            comp_choice = PE_ptr->del_comp_settings[comp_num].choice;
            done = TRUE;
```

```
         }
      }
      if (!done) comp_num++;
   }

   *panic_bar_ptr = PE_ptr->del_comp_settings[comp_num].panic_bar;

   if (comp_choice == CLEAR)
      return((float)CLEAR);
   else if (comp_choice == DIRECT)
      return(PE_ptr->del_comp_settings[comp_num].direct_setting);
   else
      return(get_delay_comp_spec_ptr(type_num)->
                threat_values_table[comp_choice][threat_num]);
}

/* ------------------------------------------------------------------------ */

/* convert_indep_delay_val converts the given independent delay value to the */
/* appropriate value given the panic bar setting. auth_unauth is set to      */
/* indicate whether the traversal is authorized or unauthorized              */
/* (AUTH or UNAUTH) and direction_state is set to indicate the traversal     */
/* direction (ENTRY, EXIT or EXIT_DEP)                                       */
/* panic_bar indicates the state of the panic_bar setting for the component  */
/* which has the given independent delay value.                              */

float convert_indep_delay_val(indep_delay_val, panic_bar,
                              auth_unauth, direction_state)
float indep_delay_val;
int panic_bar, auth_unauth, direction_state;

{
   float convert_val;

   switch (direction_state)
   {
      case ENTRY    : if (indep_delay_val == INFINITY)
                         convert_val = INFINITE_VAL;
                      else
                         convert_val = indep_delay_val;
                      break;
      case EXIT     : if (indep_delay_val == INFINITY)
                         convert_val = INFINITE_VAL;
                      else if (panic_bar)
                         convert_val = 0;
                      else
                         convert_val = indep_delay_val;
                      break;
      case EXIT_DEP : if (indep_delay_val == INFINITY)
                         convert_val = INFINITE_VAL;
                      else if (auth_unauth == UNAUTH)
                         convert_val = 0;
                      else if (panic_bar)
                         convert_val = 0;
```

```
                    else
                       convert_val = indep_delay_val;
                    break;
   }

   return(convert_val);
}

/* ------------------------------------------------------------------------ */

/* max_nondet_traversal sets the maximum cumulative nondetection probability */
/* across the input or output path segment of the given PE into the variable */
/* which max_nondet_value_ptr points to.  The traversal equation number      */
/* which corresponds to this detection is set into the variable which        */
/* max_nondet_eqn_num_ptr points to.  auth_unauth must be set to AUTH or     */
/* UNAUTH to indicate whether the authorized or unauthorized paths are to be */
/* considered.  Note: at least one traversal equation must exist for the     */
/* given auth_unauth setting. location must be set to INPUT_SEG or           */
/* OUTPUT_SEG to indicate which path segment is to be considered.            */
/* direction_state must be set to ENTRY, EXIT or EXIT_DEP to indicate        */
/* whether the intruder is traversing toward the target, away from the       */
/* target or away from the target along a path segment which was previously  */
/* traversed toward the target, respectively                                 */
/* If the direction_state is EXIT_DEP (exit dependent) then the variable     */
/* which max_nondet_eqn_num_ptr points to must contain the number of the     */
/* maximum EXIT equation from a call to this function                        */

max_nondet_traversal(layer, PE_num, auth_unauth, location, direction_state,
                     max_nondet_value_ptr, max_nondet_eqn_num_ptr)
int layer, PE_num, auth_unauth, location, direction_state;
float *max_nondet_value_ptr;
int *max_nondet_eqn_num_ptr;

{
   SEGMENT_EQN *eqn_ptr;
   PROT_ELEM *PE_ptr;
   int PE_type, max_eqn_num, eqn_num, comp_num, type_num,
       found_last_comp, found_last_eqn;
   float max_nondet_prod, current_nondet_prod, indep_nondet_val;

   PE_ptr = get_ASD_PE_ptr(layer, PE_num);
   PE_type = PE_ptr->type;

   if (auth_unauth == AUTH)
   /* authorized traversal */
   {
      if (location == INPUT_SEG)
      /* input path segment traversal */
      {
         switch(direction_state)
         {
            case ENTRY    : eqn_ptr = get_PE_cat_spec_ptr(PE_type)->
                                      authorized_nondet_eqns.
                                      entry_input_eqns;
```

```
                        break;

            case EXIT     : ;
            case EXIT_DEP : eqn_ptr = get_PE_cat_spec_ptr(PE_type)->
                                      authorized_nondet_eqns.
                                      exit_input_eqns;
                        break;
         }
      }
      /* output path segment traversal */
      else
      {
         switch(direction_state)
         {
            case ENTRY    : eqn_ptr = get_PE_cat_spec_ptr(PE_type)->
                                      authorized_nondet_eqns.
                                      entry_output_eqns;
                        break;

            case EXIT     : ;
            case EXIT_DEP : eqn_ptr = get_PE_cat_spec_ptr(PE_type)->
                                      authorized_nondet_eqns.
                                      exit_output_eqns;
                        break;
         }
      }
   }
   /* unauthorized traversal */
   else
   {
      if (location == INPUT_SEG)
      /* input path segment traversal */
      {
         switch(direction_state)
         {
            case ENTRY    : eqn_ptr = get_PE_cat_spec_ptr(PE_type)->
                                      unauthorized_nondet_eqns.
                                      entry_input_eqns;
                        break;

            case EXIT     : ;
            case EXIT_DEP : eqn_ptr = get_PE_cat_spec_ptr(PE_type)->
                                      unauthorized_nondet_eqns.
                                      exit_input_eqns;
                        break;
         }
      }
      /* output path segment traversal */
      else
      {
         switch(direction_state)
         {
            case ENTRY    : eqn_ptr = get_PE_cat_spec_ptr(PE_type)->
                                      unauthorized_nondet_eqns.
```

```
                                    entry_output_eqns;
                        break;

            case EXIT     : ;
            case EXIT_DEP : eqn_ptr = get_PE_cat_spec_ptr(PE_type)->
                                    unauthorized_nondet_eqns.
                                    exit_output_eqns;
                        break;
         }
      }
   }

   /* use the given maximum detection eqn only */
   /* to calculate exit dependent product      */
   if (direction_state == EXIT_DEP)
      eqn_num = *max_nondet_eqn_num_ptr;
   else
      eqn_num = 0;

   max_eqn_num = eqn_num;

   found_last_eqn = FALSE;

   while (!found_last_eqn)
   {
      current_nondet_prod = 1;
      comp_num = 0;
      found_last_comp = FALSE;

      while (!found_last_comp)
      {
         if (comp_num == MAX_COMP_PER_EQN)
            found_last_comp = TRUE;
         else
         {
            type_num = eqn_ptr[eqn_num].comptype[comp_num];
            if (type_num == EMPTY)
            {
               found_last_comp = TRUE;
               if (comp_num == 0)
                  found_last_eqn = TRUE;
            }
            else
            {
               indep_nondet_val = find_nondet_comp_choice_val(layer, PE_num,
                                                   type_num, location,
                                                   get_current_threat());
               current_nondet_prod *= convert_indep_nondet_val(indep_nondet_val,
                                                      type_num,
                                                      direction_state);
               comp_num++;
            }
         }
      }
```

```
      /* completed product for current equation number */
      if ((eqn_num == 0) || (direction_state == EXIT_DEP) ||
         ((current_nondet_prod > max_nondet_prod) && !found_last_eqn))
      {
         max_nondet_prod = current_nondet_prod;
         max_eqn_num = eqn_num;
      }

      eqn_num++;
      if ((eqn_num == MAX_EQNS_PER_SEGMENT) ||
          (direction_state == EXIT_DEP))
         found_last_eqn = TRUE;
   }

   *max_nondet_value_ptr = max_nondet_prod;
   *max_nondet_eqn_num_ptr = max_eqn_num;

   return;
}

/* ------------------------------------------------------------------------ */

/* find_nondet_comp_choice_val returns the detection probability for the     */
/* component type (type_num) at the given location (INPUT_SEG or OUTPUT_SEG) */
/* within the given PE based on the given threat                             */
/* If the current setting for this component is DIRECT, the direct_setting   */
/* is returned, otherwise the threat-specific independent detection prob. is */
/* returned from the detection catalog. Note: If the current component       */
/* setting is CLEAR, then (float)CLEAR is returned                           */

float find_nondet_comp_choice_val(layer, PE_num, type_num,
                                  location, threat_num)
int layer, PE_num, type_num, location, threat_num;

{
   PE_SPECS *PE_spec_ptr;
   PROT_ELEM *PE_ptr;
   int PE_type, done, comp_num, comp_choice, temp_location;

   PE_ptr = get_ASD_PE_ptr(layer, PE_num);
   PE_type = PE_ptr->type;
   PE_spec_ptr = get_PE_cat_spec_ptr(PE_type);

   done = FALSE;
   comp_num = 0;

   while (!done)
   {
      /* find the component type at the given location within the */
      /* corresponding PE catalog detection component list        */
      if (PE_spec_ptr->nondet_comp_list[comp_num].comptype == type_num)
      {
         temp_location = PE_spec_ptr->nondet_comp_list[comp_num].location;
```

```
         if ( (temp_location == location) || (temp_location == EITHER_SEG) )
         {
            /* found the component in the given PE detection choice list */
            comp_choice = PE_ptr->nondet_comp_settings[comp_num].choice;
            done = TRUE;
         }
      }
      if (!done) comp_num++;
   }

   if (comp_choice == CLEAR)
      return((float)CLEAR);
   else if (comp_choice == DIRECT)
      return(PE_ptr->nondet_comp_settings[comp_num].direct_setting);
   else
      return(get_nondet_comp_spec_ptr(type_num)->
                threat_values_table[comp_choice][threat_num]);

}

/* ------------------------------------------------------------------------ */

/* convert_indep_nondet_val converts the given independent nondetection value*/
/* to the appropriate value given the detection component type number.       */
/* direction_state is set to indicate the traversal direction (ENTRY, EXIT   */
/* or EXIT_DEP)                                                              */

float convert_indep_nondet_val(indep_nondet_val, type_num, direction_state)
float indep_nondet_val;
int type_num, direction_state;

{
   float convert_val;
   int snm;

   snm = get_nondet_comp_spec_ptr(type_num)->snm;

   switch (direction_state)
   {
      case ENTRY    : if (snm)
                         convert_val = 1;
                      else
                         convert_val = indep_nondet_val;
                      break;
      case EXIT     : convert_val = indep_nondet_val;
                      break;
      case EXIT_DEP : if (indep_nondet_val == 0)
                         convert_val = 0;
                      else if (!snm)
                         convert_val = 1;
                      else
                         convert_val = indep_nondet_val;
                      break;
   }
```

```
   return(convert_val);
}

/* ------------------------------------------------------------------------ */

/* get_number_of_critical_paths returns the number of critical paths which   */
/* were determined by the last call to turbo_SEApath.  This number will be   */
/* less than or equal to TOP_VULNERABLE_PATHS                                */

get_number_of_critical_paths()

{
   return(get_displayable_paths(get_current_rft_num()));
}

/* ------------------------------------------------------------------------ */

/* number_of_PEs_on_path returns the number of PEs on the given path for the */
/* given rft in the rft table for the current ASD                            */

number_of_PEs_on_path(rft_num, path_num)
int rft_num, path_num;

{
   return((get_path_ptr(rft_num, path_num)->number_of_nodes-1)/2);
}

/* ------------------------------------------------------------------------ */

/* get_first_present_PE_on_path loads the layer and PE_num for the first PE  */
/* which is present on the given path for the given rft num.  The direction  */
/* of traversal for this PE as well as traversal mode (AUTH or UNAUTH) is    */
/* also set.  If no present PEs are found on the given path, NOT_FOUND is    */
/* returned.  Otherwise, ZERO (0) is returned.  Note: NOT_FOUND should not   */
/* ever be returned during normal calls since the target task PE must be     */
/* present on all paths through an ASD                                       */

get_first_present_PE_on_path(rft_num, path_num, layer_ptr, PE_num_ptr,
                             direction_ptr, auth_unauth_ptr)
int rft_num, path_num;
int *layer_ptr, *PE_num_ptr, *direction_ptr, *auth_unauth_ptr;

{
   int node_num, last_node_num, node_val, done;
   PATH_INFO *path_ptr;

   path_ptr = get_path_ptr(rft_num, path_num);

   last_node_num = path_ptr->number_of_nodes - 1;

   node_num = 1;
   done = FALSE;
```

```
   while (!done)
   {
      node_val = path_ptr->nodes[node_num];

      *layer_ptr = get_PE_layer(node_val);
      *PE_num_ptr = get_PE_num(node_val);

      if (get_ASD_PE_ptr(*layer_ptr, *PE_num_ptr)->present)
      {
         if (node_is_unauth(node_val))
            *auth_unauth_ptr = UNAUTH;
         else
            *auth_unauth_ptr = AUTH;

         *direction_ptr = get_node_traversal(path_ptr, node_num);

         done = TRUE;
      }
      else
      {
         node_num += 2;
         if (node_num > last_node_num)
            return(NOT_FOUND);
      }
   }

   return(0);
}

/* ------------------------------------------------------------------------ */

/* get_PE_on_path loads the layer and PE_num variables for the count'th PE   */
/* on the given path for the given rft.  The direction in which this PE is   */
/* traversed is set in the variable that direction_ptr points to.  The mode  */
/* of traversal (AUTH or UNAUTH) is set in the variable that auth_unauth_ptr */
/* points to.  This function returns NOT_FOUND if the count'th PE does not   */
/* exist on the given path and returns ZERO (0) otherwise                    */
/* Note: the count variable should start at ZERO (0)                         */

get_PE_on_path(rft_num, path_num, count,
               layer_ptr, PE_num_ptr,
               direction_ptr, auth_unauth_ptr)
int rft_num, path_num, count;
int *layer_ptr, *PE_num_ptr, *direction_ptr, *auth_unauth_ptr;

{
   int node_num, last_node_num, node_val;
   PATH_INFO *path_ptr;

   path_ptr = get_path_ptr(rft_num, path_num);

   node_num = count * 2 + 1;
   last_node_num = path_ptr->number_of_nodes - 1;
```

```
   if (node_num > last_node_num)
      return(NOT_FOUND);
   else
   {
      node_val = path_ptr->nodes[node_num];

      *layer_ptr = get_PE_layer(node_val);
      *PE_num_ptr = get_PE_num(node_val);

      if (node_is_unauth(node_val))
         *auth_unauth_ptr = UNAUTH;
      else
         *auth_unauth_ptr = AUTH;

      *direction_ptr = get_node_traversal(path_ptr, node_num);

      return(0);
   }

}

/* ------------------------------------------------------------------------ */

/* get_time_remaining returns the time remaining after interruption along   */
/* given path for the given rft                                             */

get_time_remaining(rft_num, path_num)
int rft_num, path_num;

{
   return(get_path_ptr(rft_num, path_num)->cum_sum);
}

/* ------------------------------------------------------------------------ */

/* get_prob_interruption returns the probability of interruption along the  */
/* given path for the given rft                                             */
/* Note: the number of significant digits after the decimal point is        */
/* truncated to 4                                                           */

float get_prob_interruption(rft_num, path_num)
int rft_num, path_num;

{
   return(trunc_float(invert_det_nondet(get_path_ptr(rft_num, path_num)
                                        ->cum_prod),4));
}

/* ------------------------------------------------------------------------ */

/* the following routines handle conversions between standard SAVI areas &  */
/* layer, PE numbers and node numbers which are used by Turbo_SEApath.  The */
/* node numbers are assigned monotonically increasing from zero which is the */
/* node number of the first area (OFFSITE).  The other areas are numbered in */
```

```
/* steps of AREA_NODE_NUM_STEP. The PEs on each layer are given node numbers */
/* which begin at 1 more than the preceding area node number and increase by */
/* one across the layer.  The authorized clone PEs are assigned the node     */
/* number of the unauthorized node plus AUTH_OFFSET thus grouping all        */
/* unauthorized node numbers together and all authorized node numbers        */
/* together between the bracketing area node numbers.                        */
/* example:                                                                  */
/*                                                                           */
/*              PE 0          PE 1          PE 0          PE 1               */
/*  area 0  unauthorized  unauthorized  authorized  authorized    area 1     */
/*                                                                           */
/*    0          1             2             26            27          60     */
/*                                                                           */
/*              PE 0          PE 1          PE 0          PE 1               */
/*  area 1  unauthorized  unauthorized  authorized  authorized    area 2     */
/*                                                                           */
/*   60         61            62            76            77          120    */

/* ------------------------------------------------------------------------- */

/* get_area_node_number returns the Turbo_SEApath node number for the given  */
/* area in the current ASD                                                   */

get_area_node_number(area_num)
int area_num;

{
   return(area_num * AREA_NODE_NUM_STEP);
}

/* ------------------------------------------------------------------------- */

/* get_PE_node_number returns the Turbo_SEApath node number for the given    */
/* PE in the current ASD.  auth_unauth must be set to AUTH or UNAUTH and     */
/* specifies whether the function returns the unauthorized PE node number or */
/* the authorized clone node number.                                         */

get_PE_node_number(layer, PE_num, auth_unauth)
int layer, PE_num, auth_unauth;

{
   int node_num;

   node_num = (layer * AREA_NODE_NUM_STEP) + PE_num + 1;

   if (auth_unauth == AUTH)
      node_num += AUTH_OFFSET;

   return(node_num);
}

/* ------------------------------------------------------------------------- */

/* get_auth_node_num returns the corresponding authorized node number for    */
```

```
/* the given unauthorized node number                                    */

get_auth_node_num(unauth_node_num)
int unauth_node_num;

{
   return(unauth_node_num + AUTH_OFFSET);
}

/* ------------------------------------------------------------------- */

/* get_area_num returns the area number for the area which node_num refers  */
/* to.  Note that the given node_num must refer to an area and not a PE     */

get_area_num(node_num)
int node_num;

{
   return(node_num / AREA_NODE_NUM_STEP);
}

/* ------------------------------------------------------------------- */

/* get_PE_layer returns the layer associated with the PE which node_num     */
/* refers to.  Note that the given node_num must refer to a PE and not an   */
/* area                                                                     */

get_PE_layer(node_num)
int node_num;

{
   return(node_num / AREA_NODE_NUM_STEP);
}

/* ------------------------------------------------------------------- */

/* get_PE_num returns the PE number for the PE which node_num refers to     */

get_PE_num(node_num)
int node_num;

{
   int PE_num;

   PE_num = get_base_PE_num(node_num);

   /* remove the authorized offset if this is an authorized clone */
   if (PE_num >= AUTH_OFFSET) PE_num -= AUTH_OFFSET;

   return(PE_num);
}

/* ------------------------------------------------------------------- */
```

```
/* get_base_PE_num returns the base PE number for the PE which node_num      */
/* refers to including the AUTH_OFFSET if the node refers to an authorized   */
/* clone.                                                                    */

get_base_PE_num(node_num)
int node_num;

{
   return((node_num - 1) -
          ((node_num / AREA_NODE_NUM_STEP) * AREA_NODE_NUM_STEP));
}

/* ------------------------------------------------------------------------- */

/* node_is_area returns TRUE if the given node_num represents an area rather */
/* than a PE in the current ASD                                              */

node_is_area(node_num)
int node_num;

{
   return( (node_num % AREA_NODE_NUM_STEP) == 0);
}

/* ------------------------------------------------------------------------- */

/* node_is_PE returns TRUE if the given node_num represents a PE rather than */
/* an area in the current ASD                                                */

node_is_PE(node_num)
int node_num;

{
   return(!node_is_area(node_num));
}

/* ------------------------------------------------------------------------- */

/* node_is_unauth returns TRUE if the given node_num represents unauthorized */
/* passage of the corresponding PE                                           */
/* Note: node_num must refer to a PE and not an area                         */

node_is_unauth(node_num)
int node_num;

{
   return(get_base_PE_num(node_num) < AUTH_OFFSET);
}

/* ------------------------------------------------------------------------- */

/* get_node_traversal returns the direction relative to the target that the  */
/* path traversal is in for the given node on the given path                 */
```

```
get_node_traversal(path_ptr, node_num)
PATH_INFO *path_ptr;
int node_num;

{
   int last_node_num, target_node_num;

   last_node_num = path_ptr->number_of_nodes - 1;

   target_node_num = find_node_in_range(path_ptr, 0, last_node_num,
                                get_area_node_number(MAX_AREAS_PER_ASD - 1));

   if (node_num <= target_node_num)
      return(ENTRY);
   else
      return(EXIT);
}

/* ********************** end SAVICALC.C source file ********************** */

/* ************************ SAVICATS.C source file ************************ */

#include <savidefs.h>
#include <malloc.h>
#include <stdio.h>

/* ------------------------------------------------------------------------ */

char error_message[MAX_ERROR_STRING_LEN];

PE_SPECS *current_PE_catalog_1;
PE_SPECS *current_PE_catalog_2;

COMP_SPECS *current_delay_catalog_1;
COMP_SPECS *current_delay_catalog_2;

COMP_SPECS *current_nondet_catalog_1;
COMP_SPECS *current_nondet_catalog_2;

THREAT_CATALOG *current_threat_catalog;

/* ------------------------------------------------------------------------ */

/* allocate_PE_cat allocates enough memory for the PE catalogs (1 and 2) and */
/* sets the appropriate PE catalog pointers                                  */

allocate_PE_cat()

{
   alloc_PE_cat(&current_PE_catalog_1);
   alloc_PE_cat(&current_PE_catalog_2);

   return;
}
```

```
/* ------------------------------------------------------------------------ */

/* alloc_PE_cat allocates enough memory for a single PE catalog and sets the */
/* given PE catalog pointer.  If not enough memory is available, an error    */
/* message is output to the screen and the code is exited.                   */


alloc_PE_cat(PE_cat_ptr_ptr)
PE_SPECS * (*PE_cat_ptr_ptr);

{
   char *alloc;

   alloc = calloc(MAX_PE_CATALOG_ENTRIES,sizeof(PE_SPECS));
   if (alloc == 0l)
   {
      /* create the error message */
      sprintf(error_message,"Insufficient Memory for %s",
              PE_CATALOG_FILE_NAME);
      cls();
      popup_fatal_error_window(error_message);
      user_acknowledge_wait();
      remove_fatal_error_window();
      display_quit();
      exit(0);
   }
   else
      *PE_cat_ptr_ptr = (PE_SPECS *)alloc;

   return;

}

/* ------------------------------------------------------------------------ */

/* allocate_delay_cat allocates enough memory for the delay catalogs (1 & 2) */
/* and sets the appropriate delay catalog pointers                           */

allocate_delay_cat()

{
   alloc_delay_cat(&current_delay_catalog_1);
   alloc_delay_cat(&current_delay_catalog_2);

   return;
}

/* ------------------------------------------------------------------------ */

/* alloc_delay_cat allocates enough memory for a single delay catalog and    */
/* sets the given delay catalog pointer.  If not enough memory is available, */
/* an error message is output to the screen and the code is exited.          */
```

```
alloc_delay_cat(delay_cat_ptr_ptr)
COMP_SPECS * (*delay_cat_ptr_ptr);

{
   char *alloc;

   alloc = calloc(MAX_COMP_CATALOG_ENTRIES,sizeof(COMP_SPECS));
   if (alloc == 0l)
   {
      /* create the error message */
      sprintf(error_message,"Insufficient Memory for %s",
              DELAY_CATALOG_FILE_NAME);
      cls();
      popup_fatal_error_window(error_message);
      user_acknowledge_wait();
      remove_fatal_error_window();
      display_quit();
      exit(0);
   }
   else
      *delay_cat_ptr_ptr = (COMP_SPECS *)alloc;

   return;

}

/* ------------------------------------------------------------------------ */

/* allocate_nondet_cat allocates enough memory for the detection catalogs   */
/* (1 and 2) and sets the appropriate nondet catalog pointers               */

allocate_nondet_cat()

{
   alloc_nondet_cat(&current_nondet_catalog_1);
   alloc_nondet_cat(&current_nondet_catalog_2);

   return;
}

/* ------------------------------------------------------------------------ */

/* alloc_nondet_cat allocates enough memory for a single detection catalog  */
/* and sets the given nondet catalog pointer.  If not enough memory is      */
/* available, an error message is output to the screen and the code is      */
/* exited.                                                                  */

alloc_nondet_cat(nondet_cat_ptr_ptr)
COMP_SPECS * (*nondet_cat_ptr_ptr);

{
   char *alloc;

   alloc = calloc(MAX_COMP_CATALOG_ENTRIES,sizeof(COMP_SPECS));
```

```
   if (alloc == 0l)
   {
      /* create the error message */
      sprintf(error_message,"Insufficient Memory for %s",
              NONDET_CATALOG_FILE_NAME);
      cls();
      popup_fatal_error_window(error_message);
      user_acknowledge_wait();
      remove_fatal_error_window();
      display_quit();
      exit(0);
   }
   else
      *nondet_cat_ptr_ptr = (COMP_SPECS *)alloc;

   return;

}

/* ------------------------------------------------------------------------ */

/* allocate_threat_cat allocates enough memory for the threat catalog and   */
/* sets the current threat catalog pointer                                  */

allocate_threat_cat()

{
   char *alloc;

   alloc = calloc(1,sizeof(THREAT_CATALOG));
   if (alloc == 0l)
   {
      /* create the error message */
      sprintf(error_message,"Insufficient Memory for %s",
              THREAT_CATALOG_FILE_NAME);
      cls();
      popup_fatal_error_window(error_message);
      user_acknowledge_wait();
      remove_fatal_error_window();
      display_quit();
      exit(0);
   }
   else
      current_threat_catalog = (THREAT_CATALOG *)alloc;

   return;
}

/* ------------------------------------------------------------------------ */

/* get_PE_cat_spec_ptr returns a pointer to the PE_spec at element number   */
/* type in the appropriate current_PE_catalog                               */

PE_SPECS *get_PE_cat_spec_ptr(type)
```

```
int type;

{
   if (type < MAX_PE_CATALOG_ENTRIES)
      return(&current_PE_catalog_1[type]);
   else
      return(&current_PE_catalog_2[type-MAX_PE_CATALOG_ENTRIES]);
}

/* ------------------------------------------------------------------------ */

/* retrieve_current_PE_catalog reads the current PE_catalog from disk file   */
/* file_name.                                                                */

retrieve_current_PE_catalog(file_name)
char *file_name;

{
   retrieve_PE_catalog(file_name,current_PE_catalog_1,current_PE_catalog_2);

   return;
}

/* ------------------------------------------------------------------------ */

/* retrieve_PE_catalog retrieves PE_catalog 1 and 2 from disk file file_name */

retrieve_PE_catalog(file_name,PE_catalog_1,PE_catalog_2)
char *file_name;
PE_SPECS PE_catalog_1[];
PE_SPECS PE_catalog_2[];

{
   FILE *stream;

   stream = fopen(file_name,"r+b");
   read_PE_catalog(stream,PE_catalog_1);
   read_PE_catalog(stream,PE_catalog_2);
   fclose(stream);

   return;

}

/* ------------------------------------------------------------------------ */

read_PE_catalog(stream,PE_catalog)
FILE *stream;
PE_SPECS PE_catalog[];

{
   int PE_num;

   for (PE_num = 0; PE_num < MAX_PE_CATALOG_ENTRIES; PE_num++)
```

```
   {
      read_PE_spec(stream,&(PE_catalog[PE_num]));
   }

   return;
}

/* ------------------------------------------------------------------------ */

read_PE_spec(stream,PE_spec_ptr)
FILE *stream;
PE_SPECS *PE_spec_ptr;

{
   int int_size, eqn_num, comp_num;

   int_size = sizeof(int);

   fread(PE_spec_ptr->label,1,MAX_PE_LABEL_LEN,stream);
   fread(PE_spec_ptr->acronym,1,MAX_PE_ACRONYM_LEN,stream);
   fread((char *)&(PE_spec_ptr->authorized),int_size,1,stream);
   fread((char *)&(PE_spec_ptr->ext_int),int_size,1,stream);
   fread((char *)&(PE_spec_ptr->helicopter),int_size,1,stream);

   for (comp_num = 0; comp_num < MAX_COMP_PER_PE; comp_num++)
   {
      fread((char *)&(PE_spec_ptr->del_comp_list[comp_num].comptype),
            int_size,1,stream);
      fread((char *)&(PE_spec_ptr->del_comp_list[comp_num].location),
            int_size,1,stream);
      fread((char *)&(PE_spec_ptr->nondet_comp_list[comp_num].comptype),
            int_size,1,stream);
      fread((char *)&(PE_spec_ptr->nondet_comp_list[comp_num].location),
            int_size,1,stream);
   }

   for (eqn_num = 0; eqn_num < MAX_EQNS_PER_SEGMENT; eqn_num++)
   {
      for (comp_num = 0; comp_num < MAX_COMP_PER_EQN; comp_num++)
      {
         fread((char *)&(PE_spec_ptr->unauthorized_del_eqns.
               entry_input_eqns[eqn_num].comptype[comp_num]),
               int_size,1,stream);
         fread((char *)&(PE_spec_ptr->unauthorized_del_eqns.
               entry_output_eqns[eqn_num].comptype[comp_num]),
               int_size,1,stream);
         fread((char *)&(PE_spec_ptr->unauthorized_del_eqns.
               exit_input_eqns[eqn_num].comptype[comp_num]),
               int_size,1,stream);
         fread((char *)&(PE_spec_ptr->unauthorized_del_eqns.
               exit_output_eqns[eqn_num].comptype[comp_num]),
               int_size,1,stream);
         fread((char *)&(PE_spec_ptr->authorized_del_eqns.
               entry_input_eqns[eqn_num].comptype[comp_num]),
```

```
                int_size,1,stream);
        fread((char *)&(PE_spec_ptr->authorized_del_eqns.
                entry_output_eqns[eqn_num].comptype[comp_num]),
                int_size,1,stream);
        fread((char *)&(PE_spec_ptr->authorized_del_eqns.
                exit_input_eqns[eqn_num].comptype[comp_num]),
                int_size,1,stream);
        fread((char *)&(PE_spec_ptr->authorized_del_eqns.
                exit_output_eqns[eqn_num].comptype[comp_num]),
                int_size,1,stream);
        fread((char *)&(PE_spec_ptr->unauthorized_nondet_eqns.
                entry_input_eqns[eqn_num].comptype[comp_num]),
                int_size,1,stream);
        fread((char *)&(PE_spec_ptr->unauthorized_nondet_eqns.
                entry_output_eqns[eqn_num].comptype[comp_num]),
                int_size,1,stream);
        fread((char *)&(PE_spec_ptr->unauthorized_nondet_eqns.
                exit_input_eqns[eqn_num].comptype[comp_num]),
                int_size,1,stream);
        fread((char *)&(PE_spec_ptr->unauthorized_nondet_eqns.
                exit_output_eqns[eqn_num].comptype[comp_num]),
                int_size,1,stream);
        fread((char *)&(PE_spec_ptr->authorized_nondet_eqns.
                entry_input_eqns[eqn_num].comptype[comp_num]),
                int_size,1,stream);
        fread((char *)&(PE_spec_ptr->authorized_nondet_eqns.
                entry_output_eqns[eqn_num].comptype[comp_num]),
                int_size,1,stream);
        fread((char *)&(PE_spec_ptr->authorized_nondet_eqns.
                exit_input_eqns[eqn_num].comptype[comp_num]),
                int_size,1,stream);
        fread((char *)&(PE_spec_ptr->authorized_nondet_eqns.
                exit_output_eqns[eqn_num].comptype[comp_num]),
                int_size,1,stream);
      }
    }

    return;

}

/* ------------------------------------------------------------------------ */

/* get_delay_comp_spec_ptr returns a pointer to the comp_spec at component   */
/* number comp_num in the appropriate current_delay_catalog                  */

COMP_SPECS *get_delay_comp_spec_ptr(comp_num)
int comp_num;

{
    if (comp_num < MAX_COMP_CATALOG_ENTRIES)
       return(&current_delay_catalog_1[comp_num]);
    else
       return(&current_delay_catalog_2[comp_num-MAX_COMP_CATALOG_ENTRIES]);
```

```
}

/* ------------------------------------------------------------------------ */

/* retrieve_current_delay_catalog retrieves the current delay catalog from  */
/* disk file file_name.                                                     */

retrieve_current_delay_catalog(file_name)
char *file_name;

{
   retrieve_delay_catalog(file_name,current_delay_catalog_1,
                                    current_delay_catalog_2);

   return;
}

/* ------------------------------------------------------------------------ */

/* retrieve_delay_catalog retrieves the delay catalogs 1 and 2 from disk    */
/* file file_name.                                                          */

retrieve_delay_catalog(file_name,delay_catalog_1,delay_catalog_2)
char *file_name;
COMP_SPECS delay_catalog_1[];
COMP_SPECS delay_catalog_2[];

{
   FILE *stream;

   stream = fopen(file_name,"r+b");
   read_del_det_catalog(stream,delay_catalog_1);
   read_del_det_catalog(stream,delay_catalog_2);
   fclose(stream);

   return;

}

/* ------------------------------------------------------------------------ */

/* get_nondet_comp_spec_ptr returns a pointer to the comp_spec at component */
/* number comp_num in the appropriate current_nondet_catalog                */

COMP_SPECS *get_nondet_comp_spec_ptr(comp_num)
int comp_num;

{
   if (comp_num < MAX_COMP_CATALOG_ENTRIES)
      return(&current_nondet_catalog_1[comp_num]);
   else
      return(&current_nondet_catalog_2[comp_num-MAX_COMP_CATALOG_ENTRIES]);
}
```

```
/* ------------------------------------------------------------------------ */

/* retrieve_current_nondet_catalog retrieves the current detection catalog  */
/* from disk file file_name.                                                */

retrieve_current_nondet_catalog(file_name)
char *file_name;

{
   retrieve_nondet_catalog(file_name,current_nondet_catalog_1,
                                     current_nondet_catalog_2);

   return;
}

/* ------------------------------------------------------------------------ */

/* retrieve_nondet_catalog retrieves the detection catalogs 1 and 2 from    */
/* disk file file_name.                                                     */

retrieve_nondet_catalog(file_name,nondet_catalog_1,nondet_catalog_2)
char *file_name;
COMP_SPECS nondet_catalog_1[];
COMP_SPECS nondet_catalog_2[];

{
   FILE *stream;

   stream = fopen(file_name,"r+b");
   read_del_det_catalog(stream,nondet_catalog_1);
   read_del_det_catalog(stream,nondet_catalog_2);
   fclose(stream);

   return;

}

/* ------------------------------------------------------------------------ */

read_del_det_catalog(stream,del_det_catalog)
FILE *stream;
COMP_SPECS del_det_catalog[];

{
   int comp_num;

   for (comp_num = 0; comp_num < MAX_COMP_CATALOG_ENTRIES; comp_num++)
   {
      read_comp_spec(stream,&(del_det_catalog[comp_num]));
   }

   return;
}
```

```
/* ------------------------------------------------------------------------ */

read_comp_spec(stream,comp_spec_ptr)
FILE *stream;
COMP_SPECS *comp_spec_ptr;

{
   int int_size, float_size, choice_num, threat_num;

   int_size = sizeof(int);
   float_size = sizeof(float);

   fread(comp_spec_ptr->label,1,MAX_COMP_LABEL_LEN,stream);

   for (choice_num = 0; choice_num < MAX_CHOICES_PER_COMPTYPE; choice_num++)
   {
      fread(comp_spec_ptr->comp_choice_labels[choice_num],
            1,MAX_COMP_LABEL_LEN,stream);

      for (threat_num = 0; threat_num < MAX_THREATS; threat_num++)
      {
         fread((char *)&(comp_spec_ptr->threat_values_table[choice_num]
                                                           [threat_num]),
               float_size,1,stream);
      }
   }

   fread((char *)&(comp_spec_ptr->panic_bar),int_size,1,stream);
   fread((char *)&(comp_spec_ptr->snm),int_size,1,stream);

   return;

}

/* ------------------------------------------------------------------------ */

/* get_threat_cat_ptr returns a pointer to the current threat catalog       */

THREAT_CATALOG *get_threat_cat_ptr()

{
   return(current_threat_catalog);
}

/* ------------------------------------------------------------------------ */

/* get_threat_label returns the threat label for the given threat           */
/* threat must be in the range (0 to (MAX_THREATS-1))                       */

char *get_threat_label(threat)
int threat;

{
   return(current_threat_catalog->threat_labels[threat]);
```

```
}

/* ------------------------------------------------------------------------ */

/* get_threat_transport_mode returns the transport mode for the given threat */
/* threat must be in the range (0 to (MAX_THREATS-1))                        */

get_threat_transport_mode(threat)
int threat;

{
   return(current_threat_catalog->threat_transport_modes[threat]);
}

/* ------------------------------------------------------------------------ */

/* get_transport_speed returns the speed in meters/second for the given      */
/* transport mode (FOOT, TRUCK or HELICOPTER)                                */

float get_transport_speed(transport_mode)
int transport_mode;

{
   float transport_speed;

   switch (transport_mode)
   {
      case FOOT       : transport_speed = 4.0;
                        break;
      case TRUCK      : transport_speed = 16.0;
                        break;
      case HELICOPTER : transport_speed = 40.0;
                        break;
   }

   return(transport_speed);
}

/* ------------------------------------------------------------------------ */

/* retrieve_current_threat_catalog retrieves the current threat catalog      */
/* from disk file file_name.                                                 */

retrieve_current_threat_catalog(file_name)
char *file_name;

{
   FILE *stream;

   stream = fopen(file_name,"r+b");
   read_threat_catalog(stream,current_threat_catalog);
   fclose(stream);

   return;
```

```
}

/* ------------------------------------------------------------------------ */

read_threat_catalog(stream,threat_catalog)
FILE *stream;
THREAT_CATALOG *threat_catalog;

{
   int threat_num;

   for (threat_num = 0; threat_num < MAX_THREATS; threat_num++)
   {
      fread(threat_catalog->threat_labels[threat_num],
            1,MAX_THREAT_LABEL_LEN,stream);
   }

   fread((char *)threat_catalog->threat_transport_modes,
         sizeof(int),MAX_THREATS,stream);

   return;
}

/* ------------------------------------------------------------------------ */

/* get_number_of_comps returns the number of components of the given type    */
/* which must be selected for a PE of the given type.   Note: comp_type must */
/* be either DELAY or NONDET and PE_type is the index into the PE catalog    */

get_number_of_comps(comp_type, PE_type)
int comp_type, PE_type;

{
   int comp_num, done;
   PE_SPECS *PE_spec_ptr;

   comp_num = 0;
   done = FALSE;
   PE_spec_ptr = get_PE_cat_spec_ptr(PE_type);

   while (!done)
   {
      if (comp_type == DELAY)
      {
         if (PE_spec_ptr->del_comp_list[comp_num].comptype == EMPTY)
            done = TRUE;
         else
         {
            if (comp_num == MAX_COMP_PER_PE-1)
               done = TRUE;
            comp_num++;
         }
      }
      else
```

```
      {
         if (PE_spec_ptr->nondet_comp_list[comp_num].comptype == EMPTY)
            done = TRUE;
         else
         {
            if (comp_num == MAX_COMP_PER_PE-1)
               done = TRUE;
            comp_num++;
         }
      }
   }

return(comp_num);

}

/* ------------------------------------------------------------------------ */

/* get_number_of_choices returns the number of choices in the component      */
/* catalog of the given type (comp_type) for the given component (comp_num)  */
/* in the list of components for a PE of the given type (PE_type).           */
/* Note: comp_type must be either DELAY or NONDET and PE_type is the index   */
/* into the PE catalog                                                       */

get_number_of_choices(comp_type, comp_num, PE_type)
int comp_type, comp_num, PE_type;

{
   int choice_num, done;
   PE_SPECS *PE_spec_ptr;
   COMP_SPECS *comp_spec_ptr;

   PE_spec_ptr = get_PE_cat_spec_ptr(PE_type);
   if (comp_type == DELAY)
      comp_spec_ptr = get_delay_comp_spec_ptr(PE_spec_ptr->
                                              del_comp_list[comp_num].
                                              comptype);
   else
      comp_spec_ptr = get_nondet_comp_spec_ptr(PE_spec_ptr->
                                              nondet_comp_list[comp_num].
                                              comptype);

   choice_num = 0;
   done = FALSE;

   while (!done)
   {
      if (comp_spec_ptr->threat_values_table[choice_num][0] == EMPTY)
         done = TRUE;
      else
      {
         if (choice_num == MAX_CHOICES_PER_COMPTYPE-1)
            done = TRUE;
         choice_num++;
```

```
      }
   }

return(choice_num);

}

/* ********************** end SAVICATS.C source file ********************** */
/*#define LINT_ARGS          1*/

#include <dos.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <malloc.h>
#include <GRconst.h>
#include <GRports.h>
#include <GRextrn.h>
#include "keys.h"
#include "plot.h"
#include "menu.h"
#include "pathdef.h"
#include "pathglob.h"
#include "plotdecs.h"
#include "grafdefs.h"
#include "grafdecs.h"
#include "drawdecs.h"

/* Define variables global to the savigraf routines...                    */
extern int GrafixBoard;
extern int *RFTArrayPtr,NumRFTs,theRFT;
extern int ZoomDepth,AxisType,DownOption,TicOut,zmin,zmax;
extern int NumXVals,BarAlign,BarPattern1,BarPattern2;
extern float BarWidth,MaxY2Val,MainLimit,SubLimit;
extern float XValue[MAX_ARRAY_SIZE];
extern float Y1Value[MAX_ARRAY_SIZE],Y2Value[MAX_ARRAY_SIZE];
extern float OldY1Val[MAX_ARRAY_SIZE],OldY2Val[MAX_ARRAY_SIZE];
extern char plot_title[80];
extern LABEL xlabels[MAX_ARRAY_SIZE+2];
extern LABEL y1labels[MAX_ARRAY_SIZE+2],y2labels[MAX_ARRAY_SIZE+2];
extern char XTitle[20],Y1Title[20],Y2Title[20];
extern GR_ENVIRONMENT MainScale,SubScale,MainHome,SubHome;
extern MENUBAR *GraphMenuBar,*ZoomMenuBar;

extern patRcd SaviPat;
extern int MenusInitialized;
extern MENUBAR MainMenu,SubMenu1;
extern int GraphType;
extern int TypeMenu,ResponseMenu,ZoomMenu,ScaleMenu,PrintMenu,QuitMenu;
extern int ZoomHomeMenu,ZoomOutMenu,ZoomInMenu,ZoomQuitMenu;

char *get_threat_label(),*get_ASD_label();

int SetLog(HomeEnviron,Environ,min,max,inc,count)
```

```
GR_ENVIRONMENT *HomeEnviron,*Environ;
float min,max,inc;
int count;
{
   SetEnviron(HomeEnviron);
   SetMinor(YAXIS,1);
   SetAxis(YAXIS,LOGTYPE);
   SetYStuff(min,max,inc,count);
   CopyEnviron(Environ);
   return;
}

int SetLin(HomeEnviron,Environ)
GR_ENVIRONMENT *HomeEnviron,*Environ;
{
   SetEnviron(HomeEnviron);
   SetMinor(YAXIS,0);
   SetAxis(YAXIS,LINTYPE);
   CopyEnviron(Environ);
   return;
}

int LabelTwo(type1,type2,labels1,labels2,environ1,environ2)
int type1,type2;
LABEL *labels1,*labels2;
GR_ENVIRONMENT *environ1,*environ2;
{
   SetLabels(type1,labels1,TRUE,TRUE,
             environ1->YMin,environ1->YCount,environ1->YInc);
   SetLabels(type2,labels2,TRUE,TRUE,
             environ2->YMin,environ2->YCount,environ2->YInc);
   return;
}

int PushTwo(environ1,environ2)
GR_ENVIRONMENT *environ1,*environ2;
{
   GR_ENVIRONMENT temp;
   CopyEnviron(&temp);
   SetEnviron(environ1);
   PushEnviron();
   SetEnviron(environ2);
   PushEnviron();
   SetEnviron(&temp);
   return;
}

int PopTwo(environ1,environ2)
GR_ENVIRONMENT *environ1,*environ2;
{
   int result1,result2;
   GR_ENVIRONMENT temp;
   CopyEnviron(&temp);
   result2 = PopEnviron();
```

```
   if (result2 == OK)
      CopyEnviron(environ2);
   result1 = PopEnviron();
   if (result1 == OK)
      CopyEnviron(environ1);
   SetEnviron(&temp);
   if ((result1 == OK)&&(result2 == OK))
      return(OK);
   else
      return(NOT_OK);
}

int SetXStuff(min,max,inc,count)
float min,max,inc;
int count;
{
   SetXMin(min);
   SetXMax(max);
   SetXInc(inc);
   SetXCount(count);
   return;
}

int SetYStuff(min,max,inc,count)
float min,max,inc;
int count;
{
   SetYMin(min);
   SetYMax(max);
   SetYInc(inc);
   SetYCount(count);
   return;
}

int BuildMenuStructure()
{
   /* Create the MenuBar records for the new menu structure             */
   /*********************************************************************/
   /* NOTE - the following two lines are the normal way to create the   */
   /*        menubar records, but this method has the drawback of       */
   /*        putting the burden on SAVI to check for 'no memory'        */
   /*        error conditions or for sufficient memory before calling   */
   /*        MakeNewMenuBar.  For this reason, the menubars have been   */
   /*        allocated statically at compile time.                      */
   /* GraphMenuBar = MakeNewMenuBar();                                  */
   /* ZoomMenuBar = MakeNewMenuBar();                                   */
   /*********************************************************************/
   GraphMenuBar = &MainMenu;
   ZoomMenuBar = &SubMenu1;
   InitMenuBar(GraphMenuBar);
   InitMenuBar(ZoomMenuBar);
   /*********************************************************************/

   /* Set the padding and divider fields of the MenuBar records...      */
```

```
   SetMenuSpacing(GraphMenuBar," ","    ");
   SetMenuSpacing(ZoomMenuBar," ","    ");

   SetMenuStyle(GRAY_STYLE);

   /* Add the Menu fields to the new MenuBar records...                    */
   /* First, the Graph Menu:                                               */
   TypeMenu = AddMenuField(GraphMenuBar,"Type",TypeService,NULL,
              "Select the Type of Graph to Be Displayed");
   ResponseMenu = AddMenuField(GraphMenuBar,"Response",ResponseService,NULL,
                  "Select the Display RFT from the Response Force Time Range");
   ZoomMenu = AddMenuField(GraphMenuBar,"Zoom",NULL,ZoomMenuBar,"\0");
   ScaleMenu = AddMenuField(GraphMenuBar,"Scale",ScaleService,NULL,
                            "Select Linear or Logarithmic Scale");
   PrintMenu = AddMenuField(GraphMenuBar,"Print",PrintService,NULL,
                             "Print the Graph");
   QuitMenu = AddMenuField(GraphMenuBar,"Quit",QuitService,NULL,
                            "Return to Results Menu");
   /* Second, the Zoom SubMenu:                                            */
   ZoomInMenu = AddMenuField(ZoomMenuBar,"In",ZoomInService,NULL,
                              "Zoom In on the Graph");
   ZoomOutMenu = AddMenuField(ZoomMenuBar,"Out",ZoomOutService,NULL,
                              "Zoom Out to Previous View");
   ZoomHomeMenu = AddMenuField(ZoomMenuBar,"Home",ZoomHomeService,NULL,
                               "Restore the Home View");
   ZoomQuitMenu = AddMenuField(ZoomMenuBar,"Quit",ZoomQuitService,NULL,
                               "Return to Graphs Menu");
   /* Now add the 'hidden' asterisk (print w/ logo) command... */
   SetSpecialKey(GraphMenuBar,'*',PrintLogoService);
   return;
}

int HomeMenuBars()
{
   HomeMenuBar(GraphMenuBar);
   HomeMenuBar(ZoomMenuBar);
   ActivateMenu(GraphMenuBar,ResponseMenu);
   if (GraphType == SENSTYPE)
      DeactivateMenu(GraphMenuBar,ResponseMenu);
   if (ZoomDepth == 0)
   {
      DeactivateMenu(ZoomMenuBar,ZoomHomeMenu);
      DeactivateMenu(ZoomMenuBar,ZoomOutMenu);
   }
   else
   {
      ActivateMenu(ZoomMenuBar,ZoomHomeMenu);
      ActivateMenu(ZoomMenuBar,ZoomOutMenu);
   }

   return;
}

int DrawGraph()
```

```
{
   int i;
   int SaveAxis;

   SetEnviron(&MainScale);
   ErasePage();
   PenMode(Replace);
   if (GrafixBoard == EGA)
      PenColor(HICYAN);
   DrawAxis(TOPAXIS,TRUE,0,0,0,1);
   DrawAxis(BOTTOMAXIS,TRUE,0,1,0,1);
   DrawAxis(LEFTAXIS,TRUE,100,TicOut,0,1);
   DrawAxis(RIGHTAXIS,TRUE,0,TicOut,0,1);
   LabelAxis(BOTTOMAXIS,xlabels,7);
   LabelAxis(LEFTAXIS,y1labels,TicOut+2);
   if (GraphType == VULNTYPE)
      LabelAxis(RIGHTAXIS,y2labels,TicOut+2);
   TitleAxis(BOTTOMAXIS,XTitle,15);
   TitleAxis(LEFTAXIS,Y1Title,20);
   if (GraphType == VULNTYPE)
   {
      TitleAxis(RIGHTAXIS,Y2Title,20);
/*temp - make this more generic... */
SaveAxis = GetYType();
SetAxis(YAXIS,LINTYPE);
MoveTo(ScaledX(GetXMax()+20*(GetXMax()-GetXMin())/100.0),
       ScaledY(GetYMin()+(GetYMax()-GetYMin())/2.0)-9);
TextAlign(alignCenter,alignMiddle);
Text("(sec)");
SetAxis(YAXIS,SaveAxis);
      Legend();
   }
   if (GrafixBoard == EGA)
      PenColor(HIBLUE);
   PlotTitle(plot_title);

   if (GrafixBoard == EGA)
      PenColor(HIBLUE);
   for (i=0; i<NumXVals; i++)
   {
      if (((AxisType == LINTYPE)&&(Y1Value[i]>GetYMin()))||
          ((AxisType == LOGTYPE)&&(Y1Value[i]>pow(10.0,GetYMin()))))
         DrawBar(XValue[i],BarWidth,Y1Value[i],BarAlign,BarPattern1);
      OldY1Val[i] = Y1Value[i];
   }

   if (GraphType == VULNTYPE)
   {
      SetEnviron(&SubScale);
      if (GrafixBoard == EGA)
         PenColor(HIGREEN);
      for (i=0; i<NumXVals; i++)
      {
         if (((AxisType == LINTYPE)&&(Y2Value[i]>GetYMin()))||
```

```
            ((AxisType == LOGTYPE)&&(Y2Value[i]>pow(10.0,GetYMin()))))
            DrawBar(XValue[i],BarWidth,Y2Value[i],LEFT,BarPattern2);
         OldY2Val[i] = Y2Value[i];
      }
   }
   return;
}

int RedrawGraphAndTitle()
{
   PenMode(Replace);
   TextMode(Replace);
   if (GrafixBoard == EGA)
      PenColor(HIBLUE);
   PlotTitle(plot_title);
   RedrawGraph();
   return;
}

int RedrawGraph()
{
   int i;
   int XMin,XMax,YMax,XCenter,YCenter,RestoreScreen;
   rect MessageR,RestoreR;
   image *RestoreImage;

/*temp - 'redraw window' logic... may reactivate later.
   XMin = ScaledX(GetXMin());
   XMax = ScaledX(GetXMax());
   if (GetYType() == LINTYPE)
      YMax = ScaledY(GetYMax());
   else
      YMax = ScaledY(pow(10.0,GetYMax()));
   XCenter = (XMax+XMin)/2;
   SetRect(&MessageR,XCenter-50,YMax+3,XCenter+50,YMax+16);
   DupRect(&MessageR,&RestoreR);
   RestoreImage = (image *)malloc(ImageSize(&MessageR));
   if (RestoreImage != NULL)
   {
      ReadImage(&RestoreR,RestoreImage);
      EraseRect(&MessageR);
      PenMode(Replace);
      FrameRect(&MessageR);
      MoveTo(XCenter,YMax+9);
      TextAlign(alignCenter,alignMiddle);
      Text("Redraw");
   }
*/

   SetEnviron(&MainScale);

   if (GrafixBoard == EGA)
      PenColor(HIBLUE);
   for (i=0; i<NumXVals; i++)
```

```
   {
      if (OldY1Val[i] != Y1Value[i])
         DrawBar(XValue[i],BarWidth,Y1Value[i],BarAlign,BarPattern1);
      OldY1Val[i] = Y1Value[i];
   }

   if (GraphType == VULNTYPE)
   {
      SetEnviron(&SubScale);
      if (GrafixBoard == EGA)
         PenColor(HIGREEN);
      for (i=0; i<NumXVals; i++)
      {
         if (OldY2Val[i] != Y2Value[i])
            DrawBar(XValue[i],BarWidth,Y2Value[i],LEFT,BarPattern2);
         OldY2Val[i] = Y2Value[i];
      }
   }
/*temp - 'redraw window' logic...
   if (RestoreImage != NULL)
   {
      WriteImage(&RestoreR,RestoreImage);
      free(RestoreImage);
   }
*/
   return;
}

int ErasePage()
{
   int CharHeight;
   rect PageR;

   /*temp - Metagraphics did not include font definitions.  Font height   */
   /* is temporarily hardwired to 8 pixels                                */
   CharHeight = 8;

   SetRect(&PageR,GetPortField(c_portRect+c_xmin),
                  GetPortField(c_portRect+c_ymin),
                  GetPortField(c_portRect+c_xmax),
                  GetPortField(c_portRect+c_ymax)-(2*CharHeight+4));
   EraseRect(&PageR);
   return;
}

int DrawBar(position,width,height,alignment,pattern)
float position,width,height;
int alignment,pattern;
{
   rect ClipR,bar,frame;
   int i,y,LeftSide,RightSide,Count;
   float min,max,y1,y2,ypos,dy,yStart;

   ClipPlot();
```

```
if(GetYType() == LOGTYPE)
{
   if (height > pow(10.0,GetYMax()))
      height = pow(10.0,GetYMax());
   if (height < pow(10.0,GetYMin()))
      height = pow(10.0,GetYMin());
   max = GetYMax();
   min = GetYMin();
   ypos = (log10(height)-min)/(max-min);
}
else
{
   if (height > GetYMax())
      height = GetYMax();
   if (height < GetYMin())
      height = GetYMin();
   max = GetYMax();
   min = GetYMin();
   ypos = (height-min)/(max-min);
}
switch (alignment)
{
   case LEFT:
      LeftSide = ScaledX(position);
      RightSide = ScaledX(position+width)-1;
      break;
   case RIGHT:
      LeftSide = ScaledX(position-width);
      RightSide = ScaledX(position)-1;
      break;
   default:
      LeftSide = ScaledX(position-width/2.);
      RightSide = ScaledX(position+width/2.)-1;
}
PushEnviron();
SetEnviron(&MainScale);
SetAxis(YAXIS,LINTYPE);
dy = GetYInc();
yStart = min = GetYMin();
max = GetYMax();
PenMode(Replace);
Count = GetYCount();
for (i=Count; i>0; i--)
{
   y1 = yStart+((i-1)*dy);
   y2 = yStart+(i*dy);
   SetRect(&bar,LeftSide,ScaledY(y1)+1,RightSide,ScaledY(y2)-1);
   if (ypos < (y1-min)/(max-min))
   {
      EraseRect(&bar);
      MoveTo(LeftSide,ScaledY(y1));
      LineTo(RightSide,ScaledY(y1));
   }
   else
```

```
         if (ypos < (y2-min)/(max-min))
            y = ScaledY(y2)-1;
   }
   PopEnviron();
   SetRect(&bar,LeftSide,ScaledY(height)+1,RightSide,y);
/*temp*/
/*fprintf(stdprn,"L=%d, T=%d, R=%d, B=%d\n",LeftSide,ScaledY(height)+1,RightSide,y)
   EraseRect(&bar);

   /*************************************************************************/
   /* note - the plot routine Bar would normally be used here, but the    */
   /*  special erasing mode of DrawBar interferes with it.  Therefore, a  */
   /*  'custom' Bar routine is included here...                           */
   /* Bar(position,width,height,alignment,pattern);                       */
   /*************************************************************************/
   if (GetYType() == LOGTYPE)
   {
      if (height < pow(10.0,GetYMin()))
      {
         ClipScreen();
         return;
      }
      if (height > pow(10.0,GetYMax()))
         height = pow(10.0,GetYMax());
      min = pow(10.0,GetYMin());
   }
   else
   {
      if (height <= GetYMin())
      {
         ClipScreen();
         return;
      }
      if (height > GetYMax())
         height = GetYMax();
      min = GetYMin();
   }
   switch (alignment)
   {
      case LEFT:
         SetRect(&bar,
                 ScaledX(position),ScaledY(min),
                 ScaledX(position+width)-1,ScaledY(height));
         break;
      case RIGHT:
         SetRect(&bar,
                 ScaledX(position-width),ScaledY(min),
                 ScaledX(position)-1,ScaledY(height));
         break;
      default:
         SetRect(&bar,
                 ScaledX(position-width/2.),ScaledY(min),
                 ScaledX(position+width/2.)-1,ScaledY(height));
   }
```

```
   DupRect(&bar,&frame);
   InsetRect(&bar,1,1);
   FillRect(&bar,pattern);
   FrameRect(&frame);
   /*************************************************************************/

   ClipScreen();
   return;
}

int PlotTitle(title)
char *title;
{
   char work[80];
   int i;

   i = 0;
   while (title[i] != '/')
   {
      work[i] = title[i];
      i++;
   }
   work[i] = '\0';
   TextFace(cBold);
   TitleAxis(TOPAXIS,work,12);
   TextFace(0);
   i++;
   TitleAxis(TOPAXIS,&title[i],5);
}

int Message(MessageString,NLines)
char *MessageString;
int NLines;
{
   int i,XMin,XMax,YMin,YMax,XCenter,YCenter,RestoreScreen,CharHeight,
       Yposn,Line,start;
   char work[200];
   rect MessageR,RestoreR;
   image *RestoreImage;

   RestoreScreen = TRUE;

   XMin = GetPortField(c_portRect+c_xmin);
   XMax = GetPortField(c_portRect+c_xmax);
   YMin = GetPortField(c_portRect+c_ymin);
   YMax = GetPortField(c_portRect+c_ymax);
   XCenter = (XMax+XMin)/2;
   YCenter = (YMax+YMin)/2;
   SetRect(&MessageR,XCenter-(XMax-XMin)/6,YCenter-(YMax-YMin)/10,
                     XCenter+(XMax-XMin)/6,YCenter+(YMax-YMin)/10);
   if (RestoreScreen)
   {
      DupRect(&MessageR,&RestoreR);
      RestoreImage = (image *)malloc(ImageSize(&MessageR));
```

```
      if (RestoreImage == NULL)
         RestoreScreen = FALSE;
      else
         ReadImage(&RestoreR,RestoreImage);
   }
   EraseRect(&MessageR);
/*   InsetRect(&MessageR,2,1);*/
   FrameRect(&MessageR);

   /*temp - CharHeight hardwired to 9 */
   CharHeight = 9;

   Yposn = YCenter+(CharHeight+1)*(NLines-1)/2;
   TextAlign(alignCenter,alignMiddle);
   PenMode(Replace);
   i = 0;
   for (Line=0; Line<NLines; Line++)
   {
      start = i;
      while ((MessageString[i] != '/')&&(MessageString[i] != '\0'))
      {
         work[i] = MessageString[i];
         i++;
      }
      work[i] = '\0';
      i++;
      MoveTo(XCenter,Yposn);
      Yposn -= CharHeight+1;
      DrawString(&work[start]);
   }
   Getch();
   if (RestoreScreen)
   {
      WriteImage(&RestoreR,RestoreImage);
      free(RestoreImage);
   }
   else
      DrawGraph();
   return;
}

int zoom()
{
   int ch,count,MoveTop,ZoomMin,ZoomMax;
   float min,max,inc;

   if (GrafixBoard == EGA)
      PenColor(HIRED);
   SetAxis(YAXIS,LINTYPE);
   ZoomMin = 0;
   ZoomMax = GetYCount();
   MoveTop = TRUE;
   ch = ZoomBound(&ZoomMax,ZoomMin+1,GetYCount(),MoveTop);
   while (TRUE)
```

```
    {
        switch (ch)
        {
            case ESC:
                if (AxisType == LOGTYPE)
                    SetAxis(YAXIS,LOGTYPE);
                return(FALSE);
                break;
            case SPACE:
                MoveTop = !MoveTop;
                if (MoveTop)
                    ch = ZoomBound(&ZoomMax,ZoomMin+1,GetYCount(),MoveTop);
                else
                    ch = ZoomBound(&ZoomMin,0,ZoomMax-1,MoveTop);
                break;
            case RETURN:
                SetAxis(YAXIS,AxisType);
                if ((ZoomMin==0)&&(ZoomMax==GetYCount()))
                    return(FALSE);
                zmin = ZoomMin;
                zmax = ZoomMax;
                return(TRUE);
                break;
        }
    }
}

int ZoomBound(CurVal,MinBound,MaxBound,MoveTop)
int *CurVal,MinBound,MaxBound,MoveTop;
{
    int ArrowOn,ArrowCount,PatRot,wait,ch;
    float x1,y1,x2,y2,ArrowX1,ArrowY1,ArrowX2;
    rect flashR,oldR;

    if (MoveTop)
    {
        x1 = GetXMin();
        y1 = GetYMin()+(MinBound-1)*GetYInc();
        x2 = GetXMax();
        y2 = GetYMin()+(*CurVal)*GetYInc();
    }
    else
    {
        x1 = GetXMin();
        y1 = GetYMin()+(*CurVal)*GetYInc();
        x2 = GetXMax();
        y2 = GetYMin()+(MaxBound+1)*GetYInc();
    }

    SetRect(&flashR,ScaledX(x1),ScaledY(y1),ScaledX(x2),ScaledY(y2));
    InsetRect(&flashR,1,1);

    ch = '\0';
    PenMode(Invert);
```

```
if (GrafixBoard == CGA)
   PenPattern(12);
PatRot = 0;
ArrowOn = FALSE;
ArrowCount = 0;

while (TRUE)
{
   AlignPattern(12,0,PatRot);
   PatRot++;
   PatRot = PatRot % 8;
   FrameRect(&flashR);
   ArrowCount++;
   if (ArrowCount==5)
   {
      PenPattern(1);
      ArrowX1 = 0.3 * (GetXMax() - GetXMin());
      ArrowX2 = 0.7 * (GetXMax() - GetXMin());
      ArrowY1 = GetYMin() + (*CurVal) * GetYInc();
      if (*CurVal < MaxBound)
      {
         ArrowUp(ArrowX1,ArrowY1);
         ArrowUp(ArrowX2,ArrowY1);
      }
      if (*CurVal > MinBound)
      {
         ArrowDown(ArrowX1,ArrowY1);
         ArrowDown(ArrowX2,ArrowY1);
      }
      ArrowOn = !(ArrowOn);
      ArrowCount = 0;
      if (GrafixBoard == CGA)
         PenPattern(12);
   }
   DupRect(&flashR,&oldR);
   if (kbhit())
      ch = GetKey();
   switch (ch)
   {
      case UPARROW:
         ch = '\0';
         if (ArrowOn)
         {
            PenPattern(1);
            if (*CurVal < MaxBound)
            {
               ArrowUp(ArrowX1,ArrowY1);
               ArrowUp(ArrowX2,ArrowY1);
            }
            if (*CurVal > MinBound)
            {
               ArrowDown(ArrowX1,ArrowY1);
               ArrowDown(ArrowX2,ArrowY1);
            }
```

```
      ArrowOn = FALSE;
      if (GrafixBoard == CGA)
         PenPattern(12);
   }
   if (*CurVal < MaxBound)
      (*CurVal)++;
   break;
case DNARROW:
   ch = '\0';
   if (ArrowOn)
   {
      PenPattern(1);
      if (*CurVal < MaxBound)
      {
         ArrowUp(ArrowX1,ArrowY1);
         ArrowUp(ArrowX2,ArrowY1);
      }
      if (*CurVal > MinBound)
      {
         ArrowDown(ArrowX1,ArrowY1);
         ArrowDown(ArrowX2,ArrowY1);
      }
      ArrowOn = FALSE;
      if (GrafixBoard == CGA)
         PenPattern(12);
   }
   if (*CurVal > MinBound)
      (*CurVal)--;
   break;
case ESC:
case RETURN:
case SPACE:
   if (ArrowOn)
   {
      PenPattern(1);
      if (*CurVal < MaxBound)
      {
         ArrowUp(ArrowX1,ArrowY1);
         ArrowUp(ArrowX2,ArrowY1);
      }
      if (*CurVal > MinBound)
      {
         ArrowDown(ArrowX1,ArrowY1);
         ArrowDown(ArrowX2,ArrowY1);
      }
      ArrowOn = FALSE;
      if (GrafixBoard == CGA)
         PenPattern(12);
   }
   FrameRect(&oldR);
   PenPattern(1);
   PenMode(Replace);
   return(ch);
   break;
```

```
        }
        if (MoveTop)
        {
           x1 = GetXMin();
           y1 = GetYMin()+(MinBound-1)*GetYInc();
           x2 = GetXMax();
           y2 = GetYMin()+(*CurVal)*GetYInc();
        }
        else
        {
           x1 = GetXMin();
           y1 = GetYMin()+(*CurVal)*GetYInc();
           x2 = GetXMax();
           y2 = GetYMin()+(MaxBound+1)*GetYInc();
        }
        SetRect(&flashR,ScaledX(x1),ScaledY(y1),ScaledX(x2),ScaledY(y2));
        InsetRect(&flashR,1,1);
        for (wait=0; wait<10; wait++)
           if (kbhit())
              break;
        FrameRect(&oldR);
     }
}

int ArrowUp(Ax,Ay)
float Ax,Ay;
{
   int x,y;
   x = ScaledX(Ax);
   y = ScaledY(Ay);

   MoveTo(x-1,y+1);
   LineTo(x+1,y+1);
   MoveTo(x-1,y+2);
   LineTo(x+1,y+2);
   MoveTo(x-3,y+3);
   LineTo(x+3,y+3);
   MoveTo(x-2,y+4);
   LineTo(x+2,y+4);
   MoveTo(x-1,y+5);
   LineTo(x+1,y+5);
   MoveTo(x,y+6);
   LineTo(x,y+6);
   return;
}

int ArrowDown(Ax,Ay)
float Ax,Ay;
{
   int x,y;
   x = ScaledX(Ax);
   y = ScaledY(Ay);

   MoveTo(x-1,y-2);
```

```
   LineTo(x+1,y-2);
   MoveTo(x-1,y-3);
   LineTo(x+1,y-3);
   MoveTo(x-3,y-4);
   LineTo(x+3,y-4);
   MoveTo(x-2,y-5);
   LineTo(x+2,y-5);
   MoveTo(x-1,y-6);
   LineTo(x+1,y-6);
   MoveTo(x,y-7);
   LineTo(x,y-7);
   return;
}

int PrintScreen(DrawLogo)
int DrawLogo;
{
   union REGS regs;
   int X1,X2,Ypos;
   char buff1[11],buff2[6],buffer[81];
   rect BlankR;

   X1 = GetPortField(c_portRect+c_xmin);
   X2 = GetPortField(c_portRect+c_xmax);
   Ypos = GetPortField(c_portRect+c_ymax)-15;

   EraseMenuBar(GraphMenuBar);
   if (DrawLogo)
   {
      DefinePattern(31,&SaviPat);
      PenMode(Replace);
      PenPattern(1);
      MoveTo(X1+100,Ypos+1);
      LineTo(X2,Ypos+1);
      PlotSaviLogo(X1,Ypos,0.5,0.2,31,31);
      MoveTo(X1+120,Ypos+3);
      TextAlign(alignLeft,alignBottom);
      TextMode(Replace);
      TextFace(cBold);
      Text(get_ASD_label());
      if (get_current_direction() == 0)
         sprintf(buff1,"Entry");
      else
         sprintf(buff1,"Entry/Exit");
      if (get_current_strategy() == 0)
         sprintf(buff2,"Force");
      else
         sprintf(buff2,"Mixed");
      sprintf(buffer,"%s by %s Using %s Tactics",buff1,
              get_threat_label(get_current_threat()),buff2);
      MoveTo((X1+X2)/2,Ypos-5);
      TextFace(0);
      TextAlign(alignCenter,alignMiddle);
      Text(buffer);
```

```
    }
    else
    {
       TextAlign(alignCenter,alignMiddle);
       TextFace(cBold);
       MoveTo((X1+X2)/2,Ypos+10);
       Text(get_ASD_label());
       TextFace(0);
       if (get_current_direction() == 0)
          sprintf(buff1,"Entry");
       else
          sprintf(buff1,"Entry/Exit");
       if (get_current_strategy() == 0)
          sprintf(buff2,"Force");
       else
          sprintf(buff2,"Mixed");
       sprintf(buffer,"%s by %s Using %s Tactics",buff1,
               get_threat_label(get_current_threat()),buff2);
       MoveTo((X1+X2)/2,Ypos);
       Text(buffer);
    }
    if (printer_ready(0))
    {
       int86(PRINT_SCREEN_INTERRUPT,&regs,&regs);
       fprintf(stdprn,"%c",FORMFEED);
    }
    else
    {
       if (GrafixBoard == EGA)
          PenColor(HIRED);
       Message("Printing Not Possible/Check the Printer",2);
    }
    SetRect(&BlankR,X1,Ypos-9,X2,Ypos+15);
    EraseRect(&BlankR);
    return;
}

int init_meta()
{
    int query;

    query = QueryGrafix();
    if (query & 512 /* 0000001000000000 */)
    {
       GrafixBoard = EGA;
       InitGrafix(EGA640x200);
    }
    else
    {
       GrafixBoard = CGA;
       InitGrafix(CGA640x200);
    }
    LoadFont("savifont.fnt");
    return;
```

```
}

int PlotSaviLogo(X,Y,XScale,YScale,FillPattern,FramePattern)
int X,Y,FillPattern,FramePattern;
float XScale,YScale;
{
   int XWidth,YWidth,PolyCount;
   polyHead PolyHeaders[4];
   point PolyPts[42];
   rect PolyR,destR;

   GetSaviPoly(&PolyCount,PolyHeaders,PolyPts,&PolyR);
   XWidth = XScale*(PolyR.Xmax-PolyR.Xmin);
   YWidth = YScale*(PolyR.Ymax-PolyR.Ymin);
   SetRect(&destR,X,Y,X+XWidth,Y+YWidth);
   MapPoly(PolyCount,PolyHeaders,PolyPts,&PolyR,&destR);
   FillPoly(PolyCount,PolyHeaders,PolyPts,FillPattern);
   PenPattern(FramePattern);
   FramePoly(PolyCount,PolyHeaders,PolyPts);
   PenPattern(1);
}

int GetSaviPoly(PolyCount,PolyHeaders,PolyPoints,PolyRect)
int *PolyCount;
polyHead *PolyHeaders;
point *PolyPoints;
rect *PolyRect;
{
   int i;
   static int SaviPolyCount = 4;
   static polyHead SaviPolyHeaders[4] = {0,16, 0,0, 100,72,
                                         17,26, 48,0, 122,72,
                                         27,36, 130,0, 204,72,
                                         37,41, 172,0, 226,72};
   static point SaviLogoPoints[42] = {0,0, 10,17, 36,17, 39,19, 43,25,
                                      67,65, 72,70, 78,72, 100,72, 90,55,
                                      74,55, 71,53, 67,47, 43,7, 38,2, 32,0, 0,0,
                                      48,0, 58,17, 76,17, 108,72, 122,72,
                                      122,0, 108,0, 108,55, 76,0, 48,0,
                                      130,0, 130,72, 144,72, 144,17, 176,72, 204,72,
                                      194,55, 176,55, 144,0, 130,0,
                                      172,0, 212,72, 226,72, 186,0, 172,0};
   static rect SaviPolyRect = {0,0,226,72};

   *PolyCount = SaviPolyCount;
   for (i=0; i<4; i++)
   {
      PolyHeaders[i].polyBgn = SaviPolyHeaders[i].polyBgn;
      PolyHeaders[i].polyEnd = SaviPolyHeaders[i].polyEnd;
      PolyHeaders[i].polyRect.Xmin = SaviPolyHeaders[i].polyRect.Xmin;
      PolyHeaders[i].polyRect.Ymin = SaviPolyHeaders[i].polyRect.Ymin;
      PolyHeaders[i].polyRect.Xmax = SaviPolyHeaders[i].polyRect.Xmax;
      PolyHeaders[i].polyRect.Ymax = SaviPolyHeaders[i].polyRect.Ymax;
   }
```

```
   for (i=0; i<42; i++)
   {
      PolyPoints[i].X = SaviLogoPoints[i].X;
      PolyPoints[i].Y = SaviLogoPoints[i].Y;
   }
   PolyRect->Xmin = SaviPolyRect.Xmin;
   PolyRect->Ymin = SaviPolyRect.Ymin;
   PolyRect->Xmax = SaviPolyRect.Xmax;
   PolyRect->Ymax = SaviPolyRect.Ymax;

   return;
}
/* ------------------------------------------------------------------------ */

/* select_current_rft allows the user to select the current RFT to be used  */
/* in the display of analysis results.                                      */
/* Note: TRUE is returned if the user selects an RFT, FALSE is returned if  */
/* the user presses escape to exit the window                               */

int SetRFT()
{
   int key,done,last_key_was_1,selected,EditMode,OldRFT;
   int XMin,XMax,YMin,YMax,XCenter,YCenter,RestoreScreen;
   rect MessageR,RestoreR;
   image *RestoreImage;

   selected = FALSE;

   RestoreScreen = TRUE;

/*   XMin = GetPortField(c_portRect+c_xmin);
   XMax = GetPortField(c_portRect+c_xmax);
   YMin = GetPortField(c_portRect+c_ymin);
   YMax = GetPortField(c_portRect+c_ymax);*/
   XMin = ScaledX(GetXMin());
   XMax = ScaledX(GetXMax());
   if (GetYType() == LINTYPE)
      YMax = ScaledY(GetYMax());
   else
      YMax = ScaledY(pow(10.0,GetYMax()));
   XCenter = (XMax+XMin)/2;
/*   YCenter = (YMax+YMin)/2;*/
   SetRect(&MessageR,XCenter-130,YMax+6,XCenter+130,YMax+19);
   DupRect(&MessageR,&RestoreR);
   RestoreImage = (image *)malloc(ImageSize(&MessageR));
   if (RestoreImage == NULL)
      RestoreScreen = FALSE;
   else
      ReadImage(&RestoreR,RestoreImage);
   EraseRect(&MessageR);
   PenMode(Replace);
   FrameRect(&MessageR);
   EditMode = FALSE;
   DisplayCurrentRFT(&MessageR,EditMode);
```

```
last_key_was_1 = FALSE;
done = FALSE;

/* process key press */
while (!done)
{
   /* wait for a key press */
   key = GetKey();

   /* process key */
   switch (key)
   {
      case LEFTARROW:
      case UPARROW:
         if (theRFT > 0)
         {
            theRFT--;
            DisplayCurrentRFT(&MessageR,EditMode);
            if (!EditMode)
            {
               BuildGraph();
               RedrawGraph();
            }
         }
         break;

      case RIGHTARROW:
      case DOWNARROW:
         if (theRFT < NumRFTs-1)
         {
            theRFT++;
            DisplayCurrentRFT(&MessageR,EditMode);
            if (!EditMode)
            {
               BuildGraph();
               RedrawGraph();
            }
         }
         break;

      case HOME:
      case PAGEUP:
         theRFT = 0;
         DisplayCurrentRFT(&MessageR,EditMode);
         if (!EditMode)
         {
            BuildGraph();
            RedrawGraph();
         }
         break;

      case END:
      case PAGEDOWN:
```

```
        theRFT = NumRFTs-1;
        DisplayCurrentRFT(&MessageR,EditMode);
        if (!EditMode)
        {
           BuildGraph();
           RedrawGraph();
        }
        break;

    case RETURN:
        if (EditMode)
        {
           EditMode = FALSE;
           DisplayCurrentRFT(&MessageR,EditMode);
           BuildGraph();
           RedrawGraph();
        }
        else
        {
           EditMode = TRUE;
           OldRFT = theRFT;
           DisplayCurrentRFT(&MessageR,EditMode);
        }
        break;

    case ESC:
        if (EditMode)
        {
           EditMode = FALSE;
           theRFT = OldRFT;
           DisplayCurrentRFT(&MessageR,EditMode);
        }
        else
        {
           selected = TRUE;
           done = TRUE;
        }
        break;

    case '0':
        if (EditMode)
        {
           if ( (last_key_was_1) && (NumRFTs == 10) )
           {
              theRFT = 9;
              DisplayCurrentRFT(&MessageR,EditMode);
              if (!EditMode)
              {
                 BuildGraph();
                 RedrawGraph();
              }
           }
           else
              bell();
```

```
            last_key_was_1 = FALSE;
         }
         else
            bell();
         break;

      /* any non-zero single digit key */
      case '1':
      case '2':
      case '3':
      case '4':
      case '5':
      case '6':
      case '7':
      case '8':
      case '9':
         if (EditMode)
         {
            if (key == '1')
               last_key_was_1 = TRUE;
            else
               last_key_was_1 = FALSE;
            if ((key - '0') <= NumRFTs)
            {
               theRFT = (key - '0') - 1;
               DisplayCurrentRFT(&MessageR,EditMode);
            }
            else
               bell();
         }
         else
            bell();
         break;

      default:
         bell();
         break;
   }
   DumpKeyBuffer();
}

/*   remove_current_rft_window();*/
   PenMode(Replace);
   if (RestoreScreen)
   {
      WriteImage(&RestoreR,RestoreImage);
      free(RestoreImage);
      PlotTitle(plot_title);
   }
   else
   {
      BuildGraph();
      DrawGraph();
   }
```

```
   return(selected);
}

int DisplayCurrentRFT(DisplayR,EditMode)
rect *DisplayR;
int EditMode;
{
   int XCenter,YCenter;
   char buffer[80];
   rect InvertR;

   XCenter = (DisplayR->Xmin+DisplayR->Xmax)/2;
   YCenter = (DisplayR->Ymin+DisplayR->Ymax)/2;
   MoveTo(XCenter,YCenter);
   TextMode(Replace);
   TextAlign(alignCenter,alignMiddle);
   SetRect(&InvertR,XCenter-54,YCenter-3,XCenter-34,YCenter+5);
   if (RFTArrayPtr[theRFT] == 1)
      sprintf(buffer,"RFT # %2d  -     1 second",theRFT+1);
   else
      sprintf(buffer,"RFT # %2d  -  %4d seconds",
              theRFT+1,RFTArrayPtr[theRFT]);
   EraseRect(&InvertR);
   DrawString(buffer);
   if (EditMode)
      InvertRect(&InvertR);

   return;
}

int bell()
{
   printf("%c",7);
}


/* ************************ SAVICLER.C source file ************************** */

#include <wfd.h>
#include <savidefs.h>

/* ------------------------------------------------------------------------- */

/* confirm_clear_PE allows the user to select whether all component choices  */
/* should be cleared for the given PE in the current ASD                     */
/* If the user chooses to clear the PE choices, each component choice in the */
/* given PE in the current ASD is cleared.  Otherwise, nothing is done.      */

confirm_clear_PE(layer,PE_num)
int layer, PE_num;

{
   int key, clear_PE, done;
```

```
clear_PE = FALSE;

/* display the confirm clear PE window */
popup_clear_PE_setting_window(layer,PE_num);

/* highlight the clear PE setting within the window */
highlight_clear_PE_setting(clear_PE);

done = FALSE;

/* process key press */
while (!done)
{
   /* dump the keyboard input buffer */
   dump_key_buffer();

   /* wait for a key press */
   key = ki();

   /* process key */
   switch (-key)
   {
      /* LEFT and UP arrows set state to NO (FALSE) */
      case K_LEFT    : ;
      case K_UP      : clear_PE = FALSE;

                       highlight_clear_PE_setting(clear_PE);

                       break;

      /* RIGHT and DOWN arrows set replace to YES (TRUE) */
      case K_RIGHT   : ;
      case K_DN      : clear_PE = TRUE;

                       highlight_clear_PE_setting(clear_PE);

                       break;

      /* ENTER (Return) key */
      /* exit with current clear PE state */
      case -K_ENTER  : done = TRUE;
                       break;

      /* Y and y keys */
      /* set clear PE to YES (TRUE) and exit */
      case -'Y'      : ;
      case -'y'      : clear_PE = TRUE;
                       done = TRUE;
                       break;

      /* ESCAPE, N and n keys */
      /* set clear PE to NO (FALSE) and exit */
      case -K_ESC    : ;
```

```
            case -'N'       : ;
            case -'n'       : clear_PE = FALSE;
                              done = TRUE;
                              break;

            default         : bell();
                              break;
         }
      }

      if (clear_PE)
      {
         clear_PE_delay_choices(layer,PE_num);
         clear_PE_nondet_choices(layer,PE_num);
         set_vulnerability_current(FALSE);
      }

      remove_clear_PE_setting_window();

      return;
}

/* ------------------------------------------------------------------------ */

/* confirm_clear_all allows the user to select whether all component choices */
/* should be cleared for all PEs in the current ASD                          */
/* If the user chooses to clear all choices, each component choice in each   */
/* PE in the current ASD is cleared.  Otherwise, nothing is done.            */

confirm_clear_all()

{
   int key, clear_all, done;

   clear_all = FALSE;

   /* display the confirm clear all window */
   popup_clear_all_setting_window();

   /* highlight the clear all setting within the window */
   highlight_clear_all_setting(clear_all);

   done = FALSE;

   /* process key press */
   while (!done)
   {
      /* dump the keyboard input buffer */
      dump_key_buffer();

      /* wait for a key press */
      key = ki();

      /* process key */
```

```
      switch (-key)
      {
         /* LEFT and UP arrows set state to NO (FALSE) */
         case K_LEFT    : ;
         case K_UP      : clear_all = FALSE;
                          highlight_clear_all_setting(clear_all);
                          break;

         /* RIGHT and DOWN arrows set replace to YES (TRUE) */
         case K_RIGHT   : ;
         case K_DN      : clear_all = TRUE;
                          highlight_clear_all_setting(clear_all);
                          break;

         /* ENTER (Return) key */
         /* exit with current clear all state */
         case -K_ENTER  : done = TRUE;
                          break;

         /* Y and y keys */
         /* set clear all to YES (TRUE) and exit */
         case -'Y'      : ;
         case -'y'      : clear_all = TRUE;
                          done = TRUE;
                          break;

         /* ESCAPE, N and n keys */
         /* set clear all to NO (FALSE) and exit */
         case -K_ESC    : ;
         case -'N'      : ;
         case -'n'      : clear_all = FALSE;
                          done = TRUE;
                          break;

         default        : bell();
                          break;
      }
   }

   if (clear_all)
   {
      clear_all_PE_choices();
      set_vulnerability_current(FALSE);
   }

   remove_clear_all_setting_window();

   return;
}

/* ********************** end SAVICLER.C source file ********************** */

/* ************************ SAVICOMP.C source file ************************ */
```

```
#include <wfd.h>
#include <savidefs.h>

#define MAIN_COMP_CHOICE_MIN_ROW 6
#define MAIN_COMP_CHOICES 2

#define COMP_TYPE_CHOICE_MIN_ROW 3
#define COMP_TYPE_CHOICES 5

#define DIRECT_WINDOW_TOP 10
#define DIRECT_WINDOW_COL 26
#define DIRECT_WINDOW_HEIGHT 11
#define DIRECT_WINDOW_WIDTH 50

WINDOW *get_comp_choices_window_ptr();
WINDOW *get_comp_type_select_window_ptr();
WINDOW *get_main_component_window_ptr();

/* ------------------------------------------------------------------------ */

/* savi_define_component_settings allows the user to highlight any present   */
/* PE within the current ASD, including the task PE using the cursor keys    */
/* The user can then select the delay/detection components for the PE by     */
/* pressing the Enter <Return> key and selecting the appropriate choices     */

savi_define_component_settings()

{
   int layer, layer_temp, PE_num, PE_temp, key, done;
   PROT_ELEM *PE_ptr;

   /* mark all PEs in the current ASD for CLEARed and DIRECT set */
   /* component choices                                          */
   mark_all_present_PEs();

   /* set the layer to the first layer in the ASD with present PEs */
   layer = first_layer_with_PEs_present();

   /* set the PE number to the first present PE on the set layer */
   PE_num = first_present_PE_num_on_layer(layer);

   /* highlight the first PE */
   highlight_PE(layer,PE_num);

   done = FALSE;

   /* process key press */
   while (!done)
   {
      /* dump the keyboard input buffer */
      dump_key_buffer();

      /* wait for a key press */
      key = ki();
```

```
/* process key */
/* Note: ki() returns the negative of the key scan code  */
/* if an extended code key is pressed (ie: UP, DOWN etc) */
/* and returns the ASCII code directly otherwise         */
switch (-key)
{
   /* UP arrow */
   /* highlight PE above current PE */
   /* do not wrap around top to bottom */
   /* move to first layer above the current layer */
   /* with present PEs                            */
   /* if new PE is non-present or null then move to */
   /* the closest present PE on the new layer       */
   case K_UP    : unhighlight_PE(layer,PE_num);
                  mark_PE_comps(layer,PE_num);
                  layer_temp = layer_with_PEs_present_above(layer);
                  if ( (layer > 0) && (layer_temp != layer) )
                     {
                        layer = layer_temp;
                        PE_ptr = get_ASD_PE_ptr(layer,PE_num);
                        if (!(PE_ptr->present) || (PE_ptr->type == 0))
                           PE_num = closest_present_PE_num(layer,PE_num);
                     }
                  else
                     bell();

                  dump_key_buffer();

                  highlight_PE(layer,PE_num);

                  break;

   /* DOWN arrow */
   /* highlight PE below current PE */
   /* do not wrap around bottom to top */
   /* move to first layer below the current layer */
   /* with present PEs                            */
   /* if new PE is non-present or null then move to */
   /* the closest present PE on the new layer       */
   case K_DN    : unhighlight_PE(layer,PE_num);
                  mark_PE_comps(layer,PE_num);
                  layer_temp = layer_with_PEs_present_below(layer);
                  if ( (layer < MAX_LAYERS_PER_ASD) &&
                       (layer_temp != layer) )
                     {
                        layer = layer_temp;
                        PE_ptr = get_ASD_PE_ptr(layer,PE_num);
                        if (layer == MAX_LAYERS_PER_ASD) PE_num = 0;
                        if (!(PE_ptr->present) || (PE_ptr->type == 0))
                           PE_num = closest_present_PE_num(layer,PE_num);
                     }
                  else
                     bell();
```

```
                dump_key_buffer();

                highlight_PE(layer,PE_num);

                break;

/* LEFT arrow */
/* highlight present PE to the left of current PE */
/* wrap around to last present PE on right */
case K_LEFT   : unhighlight_PE(layer,PE_num);
                mark_PE_comps(layer,PE_num);
                PE_temp = closest_left_present_PE_num(layer,PE_num);
                if ( (PE_num > 0) && (PE_temp != PE_num) )
                   PE_num = PE_temp;
                else
                   PE_num = last_present_PE_num_on_layer(layer);

                dump_key_buffer();

                highlight_PE(layer,PE_num);

                break;

/* RIGHT arrow */
/* highlight present PE to the right of current PE */
/* wrap around to first present PE on left */
case K_RIGHT : unhighlight_PE(layer,PE_num);
                mark_PE_comps(layer,PE_num);
                PE_temp = closest_right_present_PE_num(layer,PE_num);
                if ( (PE_num < MAX_PE_PER_LAYER-1) &&
                     (PE_temp != PE_num) )
                   PE_num = PE_temp;
                else
                   PE_num = first_present_PE_num_on_layer(layer);

                dump_key_buffer();

                highlight_PE(layer,PE_num);

                break;

/* HOME key */
/* highlight the first present PE */
case K_HOME   : PE_temp = first_present_PE_num_on_layer(layer);
                if (PE_num == PE_temp) break;
                unhighlight_PE(layer,PE_num);
                mark_PE_comps(layer,PE_num);
                PE_num = PE_temp;

                dump_key_buffer();

                highlight_PE(layer,PE_num);
```

```
                break;

/* END key */
/* highlight the last present PE */
case K_END   : PE_temp = last_present_PE_num_on_layer(layer);
               if (PE_num == PE_temp) break;
               unhighlight_PE(layer,PE_num);
               mark_PE_comps(layer,PE_num);
               PE_num = PE_temp;

               dump_key_buffer();

               highlight_PE(layer,PE_num);

               break;

/* PAGE UP key */
/* highlight present PE on highest layer with present PEs */
case K_PGUP  : unhighlight_PE(layer,PE_num);
               mark_PE_comps(layer,PE_num);
               layer_temp = first_layer_with_PEs_present();
               if ( (layer > 0) && (layer_temp != layer) )
                  {
                     layer = layer_temp;
                     PE_ptr = get_ASD_PE_ptr(layer,PE_num);
                     if (!(PE_ptr->present) || (PE_ptr->type == 0))
                        PE_num = closest_present_PE_num(layer,PE_num);
                  }
               else
                  bell();

               dump_key_buffer();

               highlight_PE(layer,PE_num);

               break;

/* PAGE DOWN key */
/* highlight present PE on lowest layer with present PEs */
case K_PGDN  : unhighlight_PE(layer,PE_num);
               mark_PE_comps(layer,PE_num);
               layer_temp = last_layer_with_PEs_present();
               if ( (layer < MAX_LAYERS_PER_ASD) &&
                    (layer_temp != layer) )
                  {
                     layer = layer_temp;
                     PE_ptr = get_ASD_PE_ptr(layer,PE_num);
                     if (layer == MAX_LAYERS_PER_ASD) PE_num = 0;
                     if (!(PE_ptr->present) || (PE_ptr->type == 0))
                        PE_num = closest_present_PE_num(layer,PE_num);
                  }
               else
                  bell();
```

```
                dump_key_buffer();

                highlight_PE(layer,PE_num);

                break;

/************************/
/* D key */
case -'D'     : if (!any_PE_comp_type_choices_are_val(layer,PE_num,
                                                      DELAY,CLEAR) &&
                    !any_PE_comp_type_choices_are_val(layer,PE_num,
                                                      NONDET,CLEAR) &&
                    printer_ready(0))
                   debug(layer,PE_num);
                break;
/************************/

/* ? key */
/* clear all choices in the current PE */
case -'?'     : confirm_clear_PE(layer,PE_num);
                unhighlight_PE(layer,PE_num);
                mark_PE_comps(layer,PE_num);
                highlight_PE(layer,PE_num);

                break;

/* SUPER key '*' */
/* test for '?' immediately following */
case -'*'     : key = ki();

                if (key == '?')
                /* clear all PE choices in current ASD */
                {
                   confirm_clear_all();
                   unhighlight_PE(layer,PE_num);
                   mark_all_present_PEs();
                   highlight_PE(layer,PE_num);
                }
                else
                   bell();
                break;

/* ENTER <Return> key */
/* allow user to select components on current PE */
case -K_ENTER : begin_main_component_select(layer,PE_num);
                mark_PE_comps(layer,PE_num);
                highlight_PE(layer,PE_num);
                break;

/* ESCAPE key */
case -K_ESC   : done = TRUE;
                break;

default       : bell();
```

```
                        break;
      }
   }

   /* unhighlight the current PE */
   unhighlight_PE(layer,PE_num);

   /* unmark all present PEs */
   unmark_all_present_PEs();

   dump_key_buffer();

   return(0);
}

/* ------------------------------------------------------------------------ */

begin_main_component_select(layer,PE_num)
int layer, PE_num;

{
   int current_choice, done, return_code, toggle,
       start_delay_comp, start_nondet_comp,
       current_delay_choice, current_nondet_choice;
   WINDOW *wn_ptr;

   toggle = FALSE;
   done = FALSE;

   start_delay_comp = 0;
   start_nondet_comp = 0;
   current_delay_choice = 0;
   current_nondet_choice = 0;

   current_choice = 0;

   popup_main_component_window(layer,PE_num);
   mark_main_component_window(layer,PE_num);
   wn_ptr = get_main_component_window_ptr();

   while (!done)
   {
      if (toggle)
      {
         /* fake a return from get_user_choice indicating */
         /* the selection of the opposite of the current  */
         /* component type (DELAY or NONDET)              */
         return_code = 0;
         if (current_choice == 0)
         {
            unhighlight_row(MAIN_COMP_CHOICE_MIN_ROW,wn_ptr);
            highlight_row(MAIN_COMP_CHOICE_MIN_ROW+1,wn_ptr);
            current_choice = 1;
         }
```

```
   else
   {
      unhighlight_row(MAIN_COMP_CHOICE_MIN_ROW+1,wn_ptr);
      highlight_row(MAIN_COMP_CHOICE_MIN_ROW,wn_ptr);
      current_choice = 0;
   }
}
else
   current_choice = get_user_choice(MAIN_COMP_CHOICE_MIN_ROW,
                                    MAIN_COMP_CHOICES,
                                    current_choice, wn_ptr,
                                    &return_code);

switch (return_code)
{
   case -K_ESC : done = TRUE;
                 break;

   case -K_PGUP : ;
   case -K_HOME : current_choice = 0;
                  break;

   case -K_PGDN : ;
   case -K_END  : current_choice = MAIN_COMP_CHOICES - 1;
                  break;

   case '?'     : set_vulnerability_current(FALSE);
                  switch (current_choice)
                  {
                     case 0 : clear_PE_delay_choices(layer,PE_num);
                              mark_main_component_window(layer,PE_num);

                              break;

                     case 1 : clear_PE_nondet_choices(layer,PE_num);
                              mark_main_component_window(layer,PE_num);

                              break;
                  }
                  break;


   case 0       : switch (current_choice)
                  {
                     case 0 : select_components(DELAY,layer,PE_num,
                                                &start_delay_comp,
                                                &current_delay_choice,
                                                &toggle);
                              mark_main_component_window(layer,PE_num);

                              break;

                     case 1 : select_components(NONDET,layer,PE_num,
                                                &start_nondet_comp,
```

```
                                                    &current_nondet_choice,
                                                    &toggle);
                                        mark_main_component_window(layer,PE_num);

                                        break;
                                }
                                break;

                default     : bell();
                              break;
            }
        }

        remove_main_component_window();

        return;
}

/* ------------------------------------------------------------------------ */

/* select_components allows the user to select from the list of component    */
/* types which make up the PE at the given layer and PE_num in the current   */
/* ASD.  Note: comp_type must be DELAY or NONDET.                            */
/* The given current_choice_ptr points to the number of the current choice   */
/* within the range of component choices on a single page.  The current      */
/* choice is returned within this variable.                                  */
/* The given start_comp_ptr points to the number of the starting component.  */
/* The starting component (top component on the current page) is returned in */
/* this variable.
/* Note: if the user requests a toggle of the component type window then the */
/* toggle is set to TRUE, else toggle is set FALSE                           */

select_components(comp_type,layer,PE_num,
                  current_choice_ptr,start_comp_ptr,
                  toggle_ptr)
int comp_type, layer, PE_num;
int *current_choice_ptr, *start_comp_ptr, *toggle_ptr;

{
    int current_choice, start_comp, end_comp, new_comp, num_comps,
        PE_type, done, return_code, page_size, key, selected;
    WINDOW *wn_ptr;

    page_size = COMP_TYPE_CHOICES - 1;

    *toggle_ptr = FALSE;
    done = FALSE;
    selected = FALSE;

    start_comp = *start_comp_ptr;
    current_choice = *current_choice_ptr;

    wn_ptr = get_comp_type_select_window_ptr();
    PE_type = get_ASD_PE_ptr(layer,PE_num)->type;
```

```
num_comps = get_number_of_comps(comp_type,PE_type);

/* popup the component type selection window */
popup_comp_type_select_window(comp_type);

/* display the component type selections */
if (num_comps == 0)
{
   display_no_comp_type_selections(comp_type);
}
else
{
   end_comp = get_end_num_in_range(start_comp, num_comps,
                                   COMP_TYPE_CHOICES);
   display_comp_type_selections(comp_type,layer,PE_num,
                                start_comp,end_comp,num_comps-1);
}

while (!done)
{
   /* if no components of this type exist then */
   /* let any key press remove the comp type   */
   /* window and return                        */
   if (num_comps == 0)
   {
      /* dump the keyboard input buffer */
      dump_key_buffer();

      /* wait for a key press */
      key = ki();

      /* if user pressed SPACE BAR toggle to the */
      /* opposite component type window          */
      if (key == ' ')
      {
         *toggle_ptr = TRUE;
      }

      done = TRUE;
   }
   else
   {
   current_choice = get_user_choice(COMP_TYPE_CHOICE_MIN_ROW,
                                    end_comp - start_comp + 1,
                                    current_choice, wn_ptr,
                                    &return_code);

   switch (return_code)
   {
      /* escape from component select window */
      case -K_ESC  : done = TRUE;
                     break;

      /* move up one line above current page if possible */
```

```
case -K_UP    : if (start_comp > 0)
                {
                   start_comp--;
                   end_comp = get_end_num_in_range(start_comp,
                                                   num_comps,
                                                   COMP_TYPE_CHOICES);
                   display_comp_type_selections(comp_type,layer,PE_num,
                                                start_comp,end_comp,
                                                num_comps-1);
                }
                break;

/* move down one line below current page if possible */
case -K_DN    : if (end_comp < num_comps - 1)
                {
                   start_comp++;
                   end_comp = get_end_num_in_range(start_comp,
                                                   num_comps,
                                                   COMP_TYPE_CHOICES);
                   display_comp_type_selections(comp_type,layer,PE_num,
                                                start_comp,end_comp,
                                                num_comps-1);
                }
                break;

/* Display top page in choice list */
/* with current choice at the top  */
case -K_HOME : start_comp = 0;
                end_comp = get_end_num_in_range(start_comp,
                                                num_comps,
                                                COMP_TYPE_CHOICES);
                current_choice = 0;
                display_comp_type_selections(comp_type,layer,PE_num,
                                             start_comp,end_comp,
                                             num_comps-1);
                break;

/* Display bottom page in choice list */
/* with current choice at the bottom  */
case -K_END   : start_comp = max(0,num_comps-COMP_TYPE_CHOICES);
                end_comp = num_comps - 1;
                current_choice = end_comp - start_comp;
                display_comp_type_selections(comp_type,layer,PE_num,
                                             start_comp,end_comp,
                                             num_comps-1);
                break;

/* Page Up COMP_TYPE_CHOICES-1 lines or to top of choice list */
case -K_PGUP : /* if the top of the list is less than one page  */
                /* above the current page top, then adjust the   */
                /* new current_choice within the new top page    */
                if (start_comp < page_size)
                {
                   current_choice = current_choice +
```

```
                                    (start_comp - page_size);
                  if (current_choice < 0) current_choice = 0;
                  start_comp = 0;
               }
               else
                  start_comp = start_comp - page_size;

               end_comp = get_end_num_in_range(start_comp,
                                               num_comps,
                                               COMP_TYPE_CHOICES);

               display_comp_type_selections(comp_type,layer,PE_num,
                                            start_comp,end_comp,
                                            num_comps-1);
               break;

/* Page Down COMP_TYPE_CHOICES-1 lines or to bottom of choice list */
case -K_PGDN : /* if the bottom of the list is on the current page */
               /* then just move the current_choice to the bottom  */
               /* of the list                                      */
               if (end_comp == num_comps - 1)
                  current_choice = end_comp - start_comp;
               else
               /* if the bottom of the list is less than one page  */
               /* below the current page bottom, then adjust the   */
               /* new current_choice within the new bottom page    */
               if ((end_comp + page_size) > num_comps - 1)
               {
                  current_choice = current_choice + page_size -
                                   (num_comps - 1 - end_comp);
                  if (current_choice > COMP_TYPE_CHOICES - 1)
                     current_choice = COMP_TYPE_CHOICES - 1;
               }

               end_comp = min(num_comps - 1,
                              end_comp + page_size);

               start_comp = max(0,end_comp + 1 - COMP_TYPE_CHOICES);

               display_comp_type_selections(comp_type,layer,PE_num,
                                            start_comp,end_comp,
                                            num_comps-1);
               break;

/* user pressed ? key */
/* clear the choice for the current component */
case '?'     :  set_comp_choice_num(comp_type,
                                    current_choice + start_comp,
                                    layer, PE_num, CLEAR);

                set_vulnerability_current(FALSE);

                /* redisplay the component type selections to */
                /* update the clear indicator (?)             */
```

```
                display_comp_type_selections(comp_type,layer,PE_num,
                        start_comp,end_comp,num_comps-1);

                break;

/* user pressed SPACE BAR */
/* toggle to the opposite component type window */
case ' '      :  *toggle_ptr = TRUE;
                 done = TRUE;
                 break;

/* user has selected a component to make a specific choice */
case  0       :  selected = TRUE;
                 break;

default       :  /* if the return code contains an alphabetic letter */
                 /* highlight the corresponding component type       */
                 /* and allow user to make component choice          */
                 if (return_code >= 'A' && return_code <= 'Z')
                 {
                    new_comp = return_code - 'A';

                    if ( new_comp > num_comps - 1)
                       /* out-of-range alphabet key was pressed */
                       bell();
                    else
                    {

                    if (new_comp > num_comps-COMP_TYPE_CHOICES)
                    /* start with current choice within last page */
                    {
                       start_comp = max(0, num_comps -
                                        COMP_TYPE_CHOICES);
                       current_choice = new_comp - start_comp;
                    }
                    else
                    /* start with current component at top of */
                    /* appropriate page                       */
                    {
                       start_comp = new_comp;
                       current_choice = 0;
                    }

                    /* display the component types */
                    end_comp = get_end_num_in_range(start_comp,
                                                    num_comps,
                                                    COMP_TYPE_CHOICES);

                    display_comp_type_selections(comp_type,
                                                 layer, PE_num,
                                                 start_comp,end_comp,
                                                 num_comps-1);

                    selected = TRUE;
```

```
                        }

                     }
                     else
                        bell();

                     break;
               }

               if (selected)
               {
                  /* allow user to select choice for this component */
                  select_component_choice(comp_type, current_choice + start_comp,
                                          layer, PE_num);

                  /* redisplay the component type selections to */
                  /* update the cleared indicators (?)          */
                  display_comp_type_selections(comp_type,layer,PE_num,
                                               start_comp,end_comp,num_comps-1);

                  selected = FALSE;
               }

               }

            }

            remove_comp_type_select_window();

            *start_comp_ptr = start_comp;
            *current_choice_ptr = current_choice;

            return;
         }

         /* ------------------------------------------------------------------------ */

         /* select_component_choice allows the user to select from the list of       */
         /* component choices for the given component type (DELAY or NONDET) for the */
         /* component (comp_num) found in the component list of the PE type of the PE */
         /* at the given layer and PE_num in the current ASD.                        */

         select_component_choice(comp_type,comp_num,layer,PE_num)
         int comp_type, comp_num, layer, PE_num;

         {
            int current_choice, start_choice, end_choice, new_choice,
                num_choices, choice_setting, init_choice,
                PE_type, done, return_code, page_size, escaped, selected;
            WINDOW *wn_ptr;

            done = FALSE;
            selected = FALSE;
```

```
page_size = COMP_TYPE_CHOICES - 1;

wn_ptr = get_comp_choices_window_ptr();
PE_type = get_ASD_PE_ptr(layer,PE_num)->type;

/* the number of choices offered to the user includes all choices for */
/* the current component plus CLEAR and DIRECT INPUT                  */
num_choices = get_number_of_choices(comp_type,comp_num,PE_type) + 2;

/* determine the initial current_choice and start_choice */
init_choice = get_comp_choice_num(comp_type, comp_num, layer, PE_num);
choice_setting = init_choice;

switch(choice_setting)
{
   case CLEAR  : /* current_choice is top of first page */
                 /* clear choices option */
                 choice_setting = 0;
                 current_choice = 0;
                 start_choice = 0;
                 break;

   case DIRECT : /* current_choice is bottom of last page */
                 /* direct setting option */
                 choice_setting = num_choices - 1;
                 start_choice = max(0,num_choices-COMP_TYPE_CHOICES);
                 current_choice = choice_setting - start_choice;
                 break;

                 /* Note: increment choice setting since first */
                 /* choice in the list is the CLEAR SETTING    */
   default     : choice_setting++;
                 if (choice_setting > num_choices-COMP_TYPE_CHOICES)
                 /* start with current choice within last page */
                 {
                    start_choice = max(0,num_choices-COMP_TYPE_CHOICES);
                    current_choice = choice_setting - start_choice;
                 }
                 else
                 /* start with current choice at top of appropriate page */
                 {
                    start_choice = choice_setting;
                    current_choice = 0;
                 }
                 break;
}

/* popup the component choices window */
popup_comp_choices_window(comp_type,PE_type,comp_num);

/* display the component choices */
end_choice = get_end_num_in_range(start_choice, num_choices,
                                  COMP_TYPE_CHOICES);
```

```
display_comp_choices(comp_type,PE_type,comp_num,
                     start_choice,end_choice,num_choices-1,
                     layer,PE_num);

/* mark the current choice */
mark_component_choice(choice_setting, start_choice, end_choice);

while (!done)
{
   current_choice = get_user_choice(COMP_TYPE_CHOICE_MIN_ROW,
                                    end_choice - start_choice + 1,
                                    current_choice, wn_ptr,
                                    &return_code);

   switch (return_code)
   {
      /* escape from component choice window */
      case -K_ESC  : done = TRUE;
                     break;

      /* move up one line above current page if possible */
      case -K_UP   : if (start_choice > 0)
                     {
                        start_choice--;
                        end_choice = get_end_num_in_range(start_choice,
                                                          num_choices,
                                                          COMP_TYPE_CHOICES);
                        display_comp_choices(comp_type,PE_type,comp_num,
                                             start_choice,end_choice,
                                             num_choices-1,
                                             layer,PE_num);
                        mark_component_choice(choice_setting,
                                              start_choice, end_choice);
                     }
                     break;

      /* move down one line below current page if possible */
      case -K_DN   : if (end_choice < num_choices - 1)
                     {
                        start_choice++;
                        end_choice = get_end_num_in_range(start_choice,
                                                          num_choices,
                                                          COMP_TYPE_CHOICES);
                        display_comp_choices(comp_type,PE_type,comp_num,
                                             start_choice,end_choice,
                                             num_choices-1,
                                             layer,PE_num);
                        mark_component_choice(choice_setting,
                                              start_choice, end_choice);
                     }
                     break;

      /* Display top page in choice list */
      /* with current choice at the top  */
```

```
case -K_HOME : start_choice = 0;
               end_choice = get_end_num_in_range(start_choice,
                                                 num_choices,
                                                 COMP_TYPE_CHOICES);
               current_choice = 0;
               display_comp_choices(comp_type,PE_type,comp_num,
                                    start_choice,end_choice,
                                    num_choices-1,
                                    layer,PE_num);
               mark_component_choice(choice_setting,
                                     start_choice, end_choice);
               break;

/* Display bottom page in choice list */
/* with current choice at the bottom  */
case -K_END  : start_choice = max(0,num_choices-COMP_TYPE_CHOICES);
               end_choice = num_choices - 1;
               current_choice = end_choice - start_choice;
               display_comp_choices(comp_type,PE_type,comp_num,
                                    start_choice,end_choice,
                                    num_choices-1,
                                    layer,PE_num);
               mark_component_choice(choice_setting,
                                     start_choice, end_choice);
               break;

/* Page Up COMP_TYPE_CHOICES-1 lines or to top of choice list */
case -K_PGUP : /* if the top of the list is less than one page  */
               /* above the current page top, then adjust the   */
               /* new current_choice within the new top page    */
               if (start_choice < page_size)
               {
                  current_choice = current_choice +
                                   (start_choice - page_size);
                  if (current_choice < 0) current_choice = 0;
                  start_choice = 0;
               }
               else
                  start_choice = start_choice - page_size;

               end_choice = get_end_num_in_range(start_choice,
                                                 num_choices,
                                                 COMP_TYPE_CHOICES);

               display_comp_choices(comp_type,PE_type,comp_num,
                                    start_choice,end_choice,
                                    num_choices-1,
                                    layer,PE_num);
               mark_component_choice(choice_setting,
                                     start_choice, end_choice);

               break;

/* Page Down COMP_TYPE_CHOICES-1 lines or to bottom of choice list */
```

```
case -K_PGDN : /* if the bottom of the list is on the current page */
               /* then just move the current_choice to the bottom  */
               /* of the list                                      */
               if (end_choice == num_choices - 1)
                  current_choice = end_choice - start_choice;
               else
               /* if the bottom of the list is less than one page  */
               /* below the current page bottom, then adjust the   */
               /* new current_choice within the new bottom page    */
               if ((end_choice + page_size) > num_choices - 1)
               {
                  current_choice = current_choice + page_size -
                                   (num_choices - 1 - end_choice);
                  if (current_choice > COMP_TYPE_CHOICES - 1)
                     current_choice = COMP_TYPE_CHOICES - 1;
               }

               end_choice = min(num_choices - 1,
                                end_choice + page_size);

               start_choice = max(0,end_choice + 1 -
                                  COMP_TYPE_CHOICES);

               display_comp_choices(comp_type,PE_type,comp_num,
                                    start_choice,end_choice,
                                    num_choices-1,
                                    layer,PE_num);

               mark_component_choice(choice_setting,
                                     start_choice, end_choice);

               break;

/* user has made a specific choice */
case '?'     : ;
case '#'     : ;
case  0      :  if (return_code == '?')
                /* CLEAR the current component */
                {
                   start_choice = 0;
                   end_choice = get_end_num_in_range(start_choice,
                                                     num_choices,
                                                     COMP_TYPE_CHOICES);
                   current_choice = 0;
                }

                if (return_code == '#')
                /* DIRECT set the current component */
                {
                   start_choice = max(0,num_choices-COMP_TYPE_CHOICES);
                   end_choice = num_choices - 1;
                   current_choice = end_choice - start_choice;
                }
```

```
                    display_comp_choices(comp_type,PE_type,comp_num,
                       start_choice,end_choice,num_choices-1,
                       layer,PE_num);

                    choice_setting = start_choice + current_choice;

                    mark_component_choice(choice_setting,
                                          start_choice, end_choice);

                    selected = TRUE;

                    break;

 default        :   /* if the return code contains an alphabetic letter */
                    /* highlight the corresponding component choice     */
                    if (return_code >= 'A' && return_code <= 'Z')
                    {
                       new_choice = 1 + return_code - 'A';

                       if ( new_choice > num_choices - 2)
                          /* out-of-range alphabet key was pressed */
                          bell();
                       else
                       {
                       if (new_choice > num_choices-COMP_TYPE_CHOICES)
                       /* start with current choice within last page */
                       {
                          start_choice = max(0, num_choices -
                                                COMP_TYPE_CHOICES);
                          current_choice = new_choice - start_choice;
                       }
                       else
                       /* start with current choice at top of */
                       /* appropriate page                    */
                       {
                          start_choice = new_choice;
                          current_choice = 0;
                       }

                       /* display the component choices */
                       end_choice = get_end_num_in_range(start_choice,
                                                         num_choices,
                                                         COMP_TYPE_CHOICES);

                       display_comp_choices(comp_type,PE_type,comp_num,
                                            start_choice,end_choice,
                                            num_choices-1,
                                            layer,PE_num);

                       choice_setting = start_choice + current_choice;

                       mark_component_choice(choice_setting,
                                             start_choice, end_choice);
```

```
                        selected = TRUE;

                        }
                     }
                     else
                        bell();

                     break;
}

if (selected)
{
   if (choice_setting == 0)
   /* set component choice to CLEAR */
   {
      if (init_choice != CLEAR)
         set_vulnerability_current(FALSE);

      set_comp_choice_num(comp_type, comp_num,
                          layer, PE_num, CLEAR);

      done = TRUE;
   }
   else if (choice_setting == num_choices - 1)
   /* allow user to select direct component setting */
   /* and select panic bar setting if applicable    */
   {
      if (init_choice != DIRECT)
         set_vulnerability_current(FALSE);

      set_comp_choice_num(comp_type, comp_num,
                          layer, PE_num, DIRECT);
      escaped = select_direct_comp_setting(comp_type,
                                           comp_num,
                                           layer,PE_num);

      /* update choices display to show direct changes */
      display_comp_choices(comp_type,PE_type,comp_num,
                           start_choice,end_choice,num_choices-1,
                           layer,PE_num);

      mark_component_choice(choice_setting,
                            start_choice, end_choice);

      if (comp_type == DELAY && !escaped)
         select_panic_bar_setting(comp_num, layer, PE_num);
   }
   else
   /* set user selected component choice and */
   /* select panic bar setting if applicable */
   {
      if (init_choice != choice_setting-1)
         set_vulnerability_current(FALSE);
```

```
            set_comp_choice_num(comp_type, comp_num,
                                layer, PE_num,
                                choice_setting - 1);

            /* exit window if panic bar setting is not possible */
            if (comp_type == DELAY)
            {
               if (!select_panic_bar_setting(comp_num, layer, PE_num))
                  done = TRUE;
            }
            else
               done = TRUE;
         }

         selected = FALSE;
      }
   }

   remove_comp_choices_window();

   return;
}

/* ------------------------------------------------------------------------ */

/* select_panic_bar_setting allows the user to select whether a panic bar is */
/* present on the given delay component for the given PE and sets the state  */
/* If the given delay component is not panic bar capable then the user is    */
/* not prompted for input and FALSE is returned.                             */
/* TRUE is returned if the given delay component is panic bar capable        */

select_panic_bar_setting(comp_num, layer, PE_num)
int comp_num, layer, PE_num;

{
   PROT_ELEM *PE_ptr;
   int key, start_state, marked_state, panic_bar, done, delay_comp_type;

   PE_ptr = get_ASD_PE_ptr(layer, PE_num);

   /* if the given component type cannot have a panic bar then return */
   delay_comp_type = get_PE_cat_spec_ptr(PE_ptr->type)->
                     del_comp_list[comp_num].comptype;
   if (get_delay_comp_spec_ptr(delay_comp_type)->panic_bar == FALSE)
      return(FALSE);

   start_state = PE_ptr->del_comp_settings[comp_num].panic_bar;
   panic_bar = start_state;

   /* display the panic bar setting window */
   popup_panic_bar_setting_window();

   /* mark the current state */
   mark_panic_bar(panic_bar);
```

```
marked_state = panic_bar;

highlight_panic_bar_setting(panic_bar);

done = FALSE;

/* process key press */
while (!done)
{
   dump_key_buffer();

   /* wait for a key press */
   key = ki();

   /* process key */
   switch (-key)
   {
      /* LEFT, UP, Home and PgUp highlight NO panic bar */
      case K_LEFT   : ;
      case K_UP     : ;
      case K_HOME   : ;
      case K_PGUP   : if (panic_bar)
                      {
                         unhighlight_panic_bar_setting(panic_bar);
                         panic_bar = FALSE;
                         highlight_panic_bar_setting(panic_bar);
                      }
                      break;

      /* RIGHT, DOWN, End and PgDn highlight YES panic bar */
      case K_RIGHT  : ;
      case K_DN     : ;
      case K_END    : ;
      case K_PGDN   : if (!panic_bar)
                      {
                         unhighlight_panic_bar_setting(panic_bar);
                         panic_bar = TRUE;
                         highlight_panic_bar_setting(panic_bar);
                      }
                      break;

      /* ESCAPE key exits with start state */
      case -K_ESC    : panic_bar = start_state;
                       done = TRUE;
                       break;

      /* ENTER key sets the current state and exits */
      case -K_ENTER  : if (panic_bar != marked_state)
                       {
                          unmark_panic_bar(marked_state);
                          mark_panic_bar(panic_bar);
                          marked_state = panic_bar;
                          highlight_panic_bar_setting(panic_bar);
                       }
```

```
                        done = TRUE;
                        break;

         /* N and n keys */
         /* set panic_bar to NO and exit */
         case -'N'      : ;
         case -'n'      : if (panic_bar)
                          {
                             unhighlight_panic_bar_setting(panic_bar);
                             panic_bar = FALSE;
                          }
                          if (panic_bar != marked_state)
                          {
                             unmark_panic_bar(marked_state);
                             mark_panic_bar(panic_bar);
                             marked_state = panic_bar;
                          }
                          highlight_panic_bar_setting(panic_bar);
                          done = TRUE;
                          break;

         /* Y and y keys */
         /* set panic_bar to YES and exit */
         case -'Y'      : ;
         case -'y'      : if (!panic_bar)
                          {
                             unhighlight_panic_bar_setting(panic_bar);
                             panic_bar = TRUE;
                          }
                          if (panic_bar != marked_state)
                          {
                             unmark_panic_bar(marked_state);
                             mark_panic_bar(panic_bar);
                             marked_state = panic_bar;
                          }
                          highlight_panic_bar_setting(panic_bar);
                          done = TRUE;
                          break;

         default        : bell();
                          break;
      }
   }

   if (panic_bar != start_state)
      set_vulnerability_current(FALSE);

   /* set the chosen panic bar state within the given PE */
   PE_ptr->del_comp_settings[comp_num].panic_bar = panic_bar;

   remove_panic_bar_setting_window();

   return(TRUE);
}
```

```
/* ------------------------------------------------------------------------ */

/* select_direct_comp_setting allows the user to directly set the value of   */
/* the given component (comp_num) of the given comp_type in the given PE     */
/* Note: comp_type must be DELAY or NONDET.                                  */
/* Note: this function returns TRUE if the user escapes from the window      */

select_direct_comp_setting(comp_type, comp_num, layer, PE_num)
int comp_type, comp_num, layer, PE_num;

{
   PROT_ELEM *PE_ptr;
   DFIELDPTR field_ptr;
   float start_delay, new_delay_setting, start_det, new_det_setting;
   int new_del_minutes, new_del_seconds, new_delay, escaped;
   long int new_det;

   PE_ptr = get_ASD_PE_ptr(layer,PE_num);

   /* set the input form window size */
   mod_wn(DIRECT_WINDOW_TOP,
          DIRECT_WINDOW_COL,
          DIRECT_WINDOW_HEIGHT,
          DIRECT_WINDOW_WIDTH,
          STDWNP);

   /* set the input form window border */
   sw_border(BDR_LNP,STDWNP);

   /* create the input form */
   def_txt(1,1,
           "DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD",
           LBLUE);
   def_txt(5,12,
           "DDDDDDDDDDDDDDDDDDDDDDDDD",
           LBLUE);

   if (comp_type == DELAY)
   {
      start_delay = PE_ptr->del_comp_settings[comp_num].direct_setting;

      /* partition the delay value into minutes and seconds */
      /* so that the edit fields can display them           */
      partition_seconds((int)start_delay,
                        &new_del_minutes,&new_del_seconds);

      def_txt(0,14,
              "DIRECT DELAY SETTING",
              LWHITE);
      def_txt(3,8,
              "Delay Time Associated With This",
              LWHITE);
      def_txt(4,16,
```

```
            "Delay Component",
            LWHITE);
      def_txt(8,12,
            "DDDDDDDDDDDDDDDDDDDDDDD",
            LBLUE);

      if (new_del_minutes == 1)
         def_txt(6,22,
               "minute",
               LCYAN);
      else
         def_txt(6,22,
               "minutes",
               LCYAN);

      if (new_del_seconds == 1)
         def_txt(7,22,
               "second",
               LCYAN);
      else
         def_txt(7,22,
               "seconds",
               LCYAN);

      /* define the form input fields       */
      /* independent of the prompt location */
      field_ptr = def_fld(0,0,"","99",
                          F_INT,&new_del_minutes);
      sf_location(6,19,field_ptr);
      sf_atts(LWHITBLUE,LHIGHLITE,field_ptr);

      field_ptr = def_fld(0,0,"","59",
                          F_INT,&new_del_seconds);
      sf_location(7,19,field_ptr);
      sf_atts(LWHITBLUE,LHIGHLITE,field_ptr);

      /* accept input from user */
      rd_flds();

      /* convert the new minutes and seconds back to all seconds */
      unpartition_seconds(&new_delay,new_del_minutes,
                          new_del_seconds);

      new_delay_setting = (float)new_delay;

      if (new_delay_setting != start_delay)
         set_vulnerability_current(FALSE);

      PE_ptr->del_comp_settings[comp_num].direct_setting = new_delay_setting;
   }
   else
   {
      start_det = PE_ptr->nondet_comp_settings[comp_num].direct_setting;
```

```
      new_det = (long)(1000.0*invert_det_nondet(start_det)+.5);

      def_txt(0,12,
              "DIRECT DETECTION SETTING",
              LWHITE);
      def_txt(3,3,
              "Detection Probability Associated With This",
              LWHITE);
      def_txt(4,14,
              "Detection Component",
              LWHITE);
      def_txt(6,21,
              "probability",
              LCYAN);
      def_txt(7,12,
              "DDDDDDDDDDDDDDDDDDDDDDDD",
              LBLUE);

      /* define the form input fields       */
      /* independent of the prompt location */
      field_ptr = def_fld(0,0,"","1.999",
                          F_LDECIMAL,&new_det);
      sf_location(6,15,field_ptr);
      sf_atts(LWHITBLUE,LHIGHLITE,field_ptr);

      /* accept input from user */
      rd_flds();

      if (new_det > 1000) new_det = 1000;

      new_det_setting = invert_det_nondet((float)(new_det)/1000.0);

      if (new_det_setting != start_det)
         set_vulnerability_current(FALSE);

      PE_ptr->nondet_comp_settings[comp_num].direct_setting = new_det_setting;
   }

   /* check if user escaped from form */
   escaped = (fld_term == ABORT_FORM);

   return(escaped);
}

/* ------------------------------------------------------------------------ */

savi_define_component_quit()

{
   /* return a 1 to jump up one menu level */
   return(1);
}

/* ********************** end SAVICOMP.C source file ********************** */
```

```
/* ************************ SAVIDEFI.C source file ************************** */

#include <wfd.h>
#include <savidefs.h>

#define RESPONSE_INPUT_TOP 11
#define RESPONSE_INPUT_COL 41
#define RESPONSE_INPUT_HEIGHT 2
#define RESPONSE_INPUT_WIDTH 11

/* ------------------------------------------------------------------------ */

/* savi_define_response allows the user to set the current ASD Response      */
/* Force Time Range using Windows for Data editing functions                 */
/* The user is prompted for the number of RFTs in the range first.  If the   */
/* number of RFTs is one (1) then the user is prompted for the RFT value     */
/* otherwise, the user is prompted for the RFT range min and max values      */

savi_define_response()

{
   int ok_input, old_min_rft, old_max_rft, new_min_rft, new_max_rft,
       num_rfts, range_error;

   popup_rft_range_window();

   popup_rft_values_window();
   display_rft_values();

   while (select_number_of_rfts() != K_ESC)
   {
      ok_input = FALSE;

      while (!ok_input)
      {
         /* save rft values in case user changes to invalid settings */
         old_min_rft = get_ASD_rft(0);
         old_max_rft = get_ASD_rft(get_num_rfts()-1);

         /* allow user to set the rft range values */
         ok_input = set_rft_range(&range_error);

         /* display the changed values */
         display_rft_values();

         /* if new range values are invalid display error window */
         if (!ok_input)
         {
            popup_rft_error_window(range_error);
            user_acknowledge_wait();
            remove_rft_error_window();

            /* reset rft values to last valid settings */
```

```
            set_rft_vals(old_min_rft, old_max_rft);

            /* redisplay last valid settings */
            display_rft_values();
         }

         /* check if range values have been changed */
         new_min_rft = get_ASD_rft(0);
         new_max_rft = get_ASD_rft(get_num_rfts()-1);

         if ( (new_min_rft != old_min_rft) ||
              (new_max_rft != old_max_rft) )
            set_vulnerability_current(FALSE);
      }
   }

   remove_rft_values_window();
   remove_rft_range_window();

   return(0);
}

/* ------------------------------------------------------------------------ */

/* select_number_of_rfts allows the user to specify the number of RFTs in    */
/* the current RFT range.  This number will range from 1 to MAX_RFTS and is  */
/* set into the current ASD num_rfts field upon return                       */
/* This function returns K_ESC or K_ENTER dependent upon whether the user    */
/* pressed ESCAPE or ENTER (Return) to exit the window                       */
/* Note: the rft range window must be displayed prior to calling this        */
/* function                                                                  */

select_number_of_rfts()

{
   int key, num_rfts, start_num_rfts, old_num_rfts,
       min_rft, max_rft, done, last_key_was_1;

   num_rfts = get_num_rfts();
   start_num_rfts = num_rfts;
   old_num_rfts = num_rfts;

   last_key_was_1 = FALSE;
   done = FALSE;

   /* display the start number of rfts */
   display_num_rfts(num_rfts);

   /* highlight the start number of rfts */
   highlight_num_rfts();

   /* dump the keyboard input buffer */
   dump_key_buffer();
```

```
/* process key press */
while (!done)
{
   /* wait for a key press */
   key = ki();

   /* process key */
   /* Note: ki() returns the negative of the key scan code  */
   /* if an extended code key is pressed (ie: UP, DOWN etc) */
   /* and returns the ASCII code directly otherwise         */
   switch (-key)
   {
      /* LEFT and UP arrows decrement the number of rfts */
      /* not including wrap around                       */
      case K_LEFT   : ;
      case K_UP     : if (num_rfts > 1)
                      {
                         num_rfts--;
                         display_num_rfts(num_rfts);
                         highlight_num_rfts();
                      }

                      break;

      /* RIGHT and DOWN arrows increment the number of rfts */
      /* not including wrap around                          */
      case K_RIGHT  : ;
      case K_DN     : if (num_rfts < MAX_RFTS)
                      {
                         num_rfts++;
                         display_num_rfts(num_rfts);
                         highlight_num_rfts();
                      }

                      break;

      /* HOME and PAGE UP keys */
      /* set the number of rfts to 1 */
      case K_HOME   : ;
      case K_PGUP   : num_rfts = 1;

                      display_num_rfts(num_rfts);
                      highlight_num_rfts();

                      break;

      /* END and PAGE DOWN keys */
      /* set the number of rfts to MAX_RFTS */
      case K_END    : ;
      case K_PGDN   : num_rfts = MAX_RFTS;

                      display_num_rfts(num_rfts);
                      highlight_num_rfts();
```

```
                    break;

    /* ESCAPE and ENTER (Return) keys */
    /* user has selected number of rfts */
    /* return the current key */
    case -K_ESC    : ;
    case -K_ENTER  : display_num_rfts(num_rfts);
                     done = TRUE;
                     break;

    /* ZERO key (only valid after a 1) */
    case ('0')     : if (last_key_was_1)
                     {
                        num_rfts = 10;
                        display_num_rfts(num_rfts);
                        highlight_num_rfts();
                     }
                     else
                        bell();

                     last_key_was_1 = FALSE;

                     break;

    /* any non-zero single digit key */
    case ('1')     : ;
    case ('2')     : ;
    case ('3')     : ;
    case ('4')     : ;
    case ('5')     : ;
    case ('6')     : ;
    case ('7')     : ;
    case ('8')     : ;
    case ('9')     : if (key == '1')
                        last_key_was_1 = TRUE;
                     else
                        last_key_was_1 = FALSE;

                     num_rfts = key - '0';

                     display_num_rfts(num_rfts);
                     highlight_num_rfts();

                     break;

    default        : bell();
                     break;
}

/* if the number of RFTs was changed FROM 1 */
/* get the maximum RFT value from RFT #1    */
/* and adjust the RFT range if necessary    */
/*  - or -                                  */
/* if the number of RFTs was changed TO 1   */
```

```
/* set the max value in the second RFT (#1) */
/* for storage so that no values are lost   */
if (old_num_rfts != num_rfts)
{
   if (old_num_rfts == 1)
   {
      /* RFT changed FROM 1 */
      /* get the max RFT value from RFT #1 */
      min_rft = get_ASD_rft(0);
      max_rft = get_ASD_rft(1);

      /* ensure that min is less than max by at least (MAX_RFTS-1) */
      if (min_rft > max_rft - (MAX_RFTS-1))
      {
         if (min_rft > 5999 - (MAX_RFTS-1))
         {
            min_rft = 5999 - (MAX_RFTS-1);
            max_rft = 5999;
         }
         else
            max_rft = min_rft + (MAX_RFTS-1);
      }

      /* set the new number of RFTs in the current ASD */
      /* along with the rft values range               */
      set_num_rfts(num_rfts);
      set_rft_vals(min_rft, max_rft);
   }
   else if (num_rfts == 1)
   {
      /* RFT changed TO 1 */
      min_rft = get_ASD_rft(0);
      max_rft = get_ASD_rft(old_num_rfts-1);

      /* set the new number of RFTs in the current ASD */
      /* along with the rft values range               */
      /* note: store the max RFT value in RFT #1       */
      set_num_rfts(2);
      set_rft_vals(min_rft, max_rft);
      set_num_rfts(1);
   }
   else
   {
      /* RFT was NOT changed to or from 1 */
      min_rft = get_ASD_rft(0);
      max_rft = get_ASD_rft(old_num_rfts-1);

      /* set the new number of RFTs in the current ASD */
      /* along with the rft values range               */
      set_num_rfts(num_rfts);
      set_rft_vals(min_rft, max_rft);
   }

   /* update the rft values window if  */
```

```
         /* number of rfts was changed to or */
         /* from 1                           */
         if ( (old_num_rfts == 1) || (num_rfts == 1) )
         {
            remove_rft_values_window();
            popup_rft_values_window();
            display_rft_values();
         }

      }

      old_num_rfts = num_rfts;

   }

   if (num_rfts != start_num_rfts)
   {
      set_vulnerability_current(FALSE);
   }

   dump_key_buffer();

   return(key);
}

/* ------------------------------------------------------------------------ */

/* set_rft_range allows the user to specify the RFTs in the current range.   */
/* If the current number of RFTs is one (1) then the user is prompted for    */
/* the RFT value. otherwise, the user is prompted for the RFT range min and  */
/* max values                                                                */
/* Note: set_rft_range returns FALSE if the user inputs an invalid min or    */
/* max value so the caller can display an error message and/or adjust values */
/* When an input range error occurs, the variable that range_error_ptr is    */
/* pointing to is set to MAX_LESS_THAN_MIN or RANGE_TOO_SMALL depending on   */
/* the type of range error.  TRUE is returned otherwise                      */

set_rft_range(range_error_ptr)
int *range_error_ptr;

{
   DFIELDPTR field_ptr;
   int num_rfts, min_rft, max_rft,
       min_rft_minutes, min_rft_seconds,
       max_rft_minutes, max_rft_seconds,
       start_min_rft, start_max_rft,
       rft_num, rft_step, ok_input,
       correction;

   num_rfts = get_num_rfts();
   min_rft = get_ASD_rft(0);
   start_min_rft = min_rft;

   /* partition the current minimum ASD RFT into minutes and seconds */
```

```
/* so that the edit window can display it                              */
partition_seconds(min_rft,&min_rft_minutes,&min_rft_seconds);

/* correct the input form height for number of RFTs */
if (num_rfts == 1)
   correction = 0;
else
   correction = 5;

/* set the input form window size */
mod_wn(RESPONSE_INPUT_TOP,
       RESPONSE_INPUT_COL,
       RESPONSE_INPUT_HEIGHT + correction,
       RESPONSE_INPUT_WIDTH,
       STDWNP);

/* set the input form to have no border */
sw_border(BDR_OP,STDWNP);

if (num_rfts == 1)
/* set only the minimum RFT */
{
   /* create the input form */
   if (min_rft_minutes == 1)
      def_txt(0,3,
              "minute",
              LCYAN);
   else
      def_txt(0,3,
              "minutes",
              LCYAN);

   if (min_rft_seconds == 1)
      def_txt(1,3,
              "second",
              LCYAN);
   else
      def_txt(1,3,
              "seconds",
              LCYAN);

   /* define the form input fields */
   /* independent of the prompt location */
   field_ptr = def_fld(0,0,"","99",
                       F_INT,&min_rft_minutes);
   sf_location(0,0,field_ptr);
   sf_atts(LWHITBLUE,LHIGHLITE,field_ptr);
   field_ptr = def_fld(0,0,"","59",
                       F_INT,&min_rft_seconds);
   sf_location(1,0,field_ptr);
   sf_atts(LWHITBLUE,LHIGHLITE,field_ptr);

   /* accept input from user */
   rd_flds();
```

```
   /* convert the new minutes and seconds */
   /* back into all seconds form          */
   unpartition_seconds(&min_rft,min_rft_minutes,min_rft_seconds);

   /* note: any input for a single rft is valid */
   ok_input = TRUE;
   *range_error_ptr = 0;

   /* set the new RFT in the current ASD */
   set_ASD_rft(0,min_rft);

}
else
/* set both the minimum and maximum RFTs */
{
   max_rft = get_ASD_rft(num_rfts-1);
   start_max_rft = max_rft;

   /* partition the current maximum ASD RFT into minutes and seconds */
   /* so that the edit window can display it                         */
   partition_seconds(max_rft,&max_rft_minutes,&max_rft_seconds);

   /* display the MINIMUM and MAXIMUM headings */
   def_txt(0,2,
           "Minimum",
           LCYAN);
   def_txt(3,0,
           "DDDDDDDDDDD",
           LBLUE);
   def_txt(4,2,
           "Maximum",
           LCYAN);

   /* create the input form */
   if (min_rft_minutes == 1)
      def_txt(1,3,
              "minute",
              LCYAN);
   else
      def_txt(1,3,
              "minutes",
              LCYAN);

   if (min_rft_seconds == 1)
      def_txt(2,3,
              "second",
              LCYAN);
   else
      def_txt(2,3,
              "seconds",
              LCYAN);

   if (max_rft_minutes == 1)
```

```
   def_txt(5,3,
           "minute",
           LCYAN);
else
   def_txt(5,3,
           "minutes",
           LCYAN);

if (max_rft_seconds == 1)
   def_txt(6,3,
           "second",
           LCYAN);
else
   def_txt(6,3,
           "seconds",
           LCYAN);

/* define the form input fields */
/* independent of the prompt location */
field_ptr = def_fld(0,0,"","99",
                    F_INT,&min_rft_minutes);
sf_location(1,0,field_ptr);
sf_atts(LWHITBLUE,LHIGHLITE,field_ptr);
field_ptr = def_fld(0,0,"","59",
                    F_INT,&min_rft_seconds);
sf_location(2,0,field_ptr);
sf_atts(LWHITBLUE,LHIGHLITE,field_ptr);
field_ptr = def_fld(0,0,"","99",
                    F_INT,&max_rft_minutes);
sf_location(5,0,field_ptr);
sf_atts(LWHITBLUE,LHIGHLITE,field_ptr);
field_ptr = def_fld(0,0,"","59",
                    F_INT,&max_rft_seconds);
sf_location(6,0,field_ptr);
sf_atts(LWHITBLUE,LHIGHLITE,field_ptr);

/* accept input from user */
rd_flds();

/* convert the new minutes and seconds */
/* back into all seconds form          */
unpartition_seconds(&min_rft,min_rft_minutes,min_rft_seconds);
unpartition_seconds(&max_rft,max_rft_minutes,max_rft_seconds);

/* ensure that max is greater than min */
/* by at least MAX_RFTS-1 seconds      */
if (max_rft < min_rft)
{
   ok_input = FALSE;
   *range_error_ptr = MAX_LESS_THAN_MIN;
}
else if (max_rft < (min_rft + (MAX_RFTS-1)))
{
   ok_input = FALSE;
```

```
         *range_error_ptr = RANGE_TOO_SMALL;
      }
      else
      {
         ok_input = TRUE;
         *range_error_ptr = 0;
      }

      /* set the RFTs in the current ASD */
      set_rft_vals(min_rft, max_rft);

   }

   dump_key_buffer();

   return(ok_input);
}

/* ------------------------------------------------------------------------ */

/* set_rft_vals sets the rfts in the current range based on the given rft    */
/* minimum and maximum                                                       */

set_rft_vals(min_rft, max_rft)
int min_rft, max_rft;

{
   int num_rfts, rft_num;
   float rft_step;

   num_rfts = get_num_rfts();

   /* set the minimum and maximum RFTs in the current ASD */
   set_ASD_rft(0,min_rft);
   if (num_rfts == 1) return;
   set_ASD_rft(num_rfts-1,max_rft);

   /* calculate and set the intermediate RFTs */
   rft_step = (float)(max_rft - min_rft)/(float)(num_rfts - 1);
   for (rft_num = 1; rft_num < num_rfts-1; rft_num++)
   {
      set_ASD_rft(rft_num,
                  (int)((float)min_rft + ((float)rft_num * rft_step) + .5));
   }

   return;
}

/* ------------------------------------------------------------------------ */

savi_define_quit()

{
   dump_key_buffer();
```

```
   /* return a 1 to jump up one menu level */
   return(1);
}

/* ********************** end SAVIDEFI.C source file ********************** */

/* ************************** SAVIDISP.C source file ************************** */

#include <wfd.h>
#include <savidefs.h>
#include <stdlib.h>
#include <stdio.h>

/* These refer to coordinates within ASD_wn */
#define ASD_LABEL_ROW 21
#define ASD_LABEL_COL 27
#define ASD_AREA_LABEL_COL 3

#define BYPASS_COL 9

#define TASK_PE_LAYER 5
#define TASK_PE_NUM 0

#define LEFT 0
#define RIGHT 1

#define ESC 27

WINDOW ASD_wn;
WINDOW intro_wn;

/* ------------------------------------------------------------------------ */

/* init_display initializes the display capabilities                        */

init_display()

{
   /* hide the screen cursor */
   csr_hide();

   /* set the clear screen color attribute */
   cl_att = LNORMAL;

   /* define the current ASD window */
   defs_wn(&ASD_wn,2,0,23,80,BDR_OP);

   /* set current ASD window for no margins */
   sw_margin(0,0,&ASD_wn);

   /* set the current ASD window color attribute */
   sw_att(LNORMAL,&ASD_wn);
```

```
   /* set current ASD window */
   /* to not clear to end of */
   /* line (ie: support      */
   /* character addressing)  */
   sw_cleor(NO_CLEAR,&ASD_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* display_intro displays the SAVI intro screen                             */

display_intro()

{
   char SAVI_string[50], temp_string[12];
   int sqweezable[50], row, i;

   sprintf(SAVI_string,"%s",
           "Systematic Analysis of Vulnerability to Intrusion");
   for (i=0;i<50;i++)
      sqweezable[i] = TRUE;
   sqweezable[0] = FALSE;
   sqweezable[11] = FALSE;
   sqweezable[23] = FALSE;
   sqweezable[40] = FALSE;
   sqweezable[49] = FALSE;

   /* clear the screen */
   cls();

   /* define the intro window to be the entire screen */
   defs_wn(&intro_wn,0,0,25,80,BDR_OP);

   /* set the intro window   */
   /* to not clear to end of */
   /* line (ie: support      */
   /* character addressing)  */
   sw_cleor(NO_CLEAR,&intro_wn);

   /* set the intro window on the screen */
   set_wn(&intro_wn);

   /* display the intro message */
   sw_att(LGREEN,&intro_wn);
   v_plst(0,10,"ZDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD?",
          &intro_wn);
   for (row=1;row<24;row++)
   {
      v_plst(row,10,"3",&intro_wn);
      v_plst(row,68,"3",&intro_wn);
   }
   v_plst(24,10,"@DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDY",
```

```
            &intro_wn);
    v_plst(15,10,"CDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD4",
            &intro_wn);
    v_plst(19,10,"CDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD4",
            &intro_wn);

    sw_att(LBLUE,&intro_wn);
    v_plst(3,13,
            "VDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD7",
            &intro_wn);
    v_plst(4,13,
            ":                                                    :",
            &intro_wn);
    v_plst(5,13,
            "SDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD=",
            &intro_wn);

    display_SAVI_string(15,SAVI_string,sqweezable);

    sw_att(LCYAN,&intro_wn);
    v_plst(21,19,"Developed by Sandia National Laboratories",
            &intro_wn);
    v_plst(22,19,"and Science & Engineering Associates, Inc.",
            &intro_wn);

    pause_sec(2);

    animate_SAVI_intro(SAVI_string,sqweezable);

    /* load the temporary string with the current SAVI version number */
    sprintf(temp_string,"Version %3.1fE",SAVI_VERSION);
    sw_att(LCYANHI,&intro_wn);
    v_plst(8,33,temp_string,&intro_wn);

    return;
}

/* ------------------------------------------------------------------------ */

animate_SAVI_intro(SAVI_string,sqweezable)
char SAVI_string[];
int sqweezable[];

{
    int left_col, right_col, left_PE_col, right_PE_col,
        done, sqweeze_count, direct;

    left_col = 13;
    right_col = 65;
    done = FALSE;
    left_PE_col = 12;
    right_PE_col = 62;
    direct = RIGHT;
    sqweeze_count = 0;
```

```
   while (!done)
   {
      /* sqweeze the SAVI string */
      if (sqweeze_SAVI_string(SAVI_string,sqweezable))
      {
         /* delay the sqweeze */
         wait_tics(1);

         /* adjust the SAVI box */
         shift_SAVI_box(direct,&left_col,&right_col);

         /* display the sqweezed string */
         display_SAVI_string(left_col+2,SAVI_string,sqweezable);

         /* toggle the sqweeze direction */
         if (direct == RIGHT)
            direct = LEFT;
         else
            direct = RIGHT;

         /* display new intro PE every 9th sqweeze */
         if (sqweeze_count % 9 == 0)
         {
            display_intro_PE(left_PE_col);
            display_intro_PE(right_PE_col);
            left_PE_col = left_PE_col + 5;
            right_PE_col = right_PE_col - 5;
         }

         sqweeze_count++;
      }
      else
         done = TRUE;
   }

   /* display the center intro PE */
   display_intro_PE(37);

   return;
}

/* ------------------------------------------------------------------------ */

/* sqweeze_SAVI_string removes the right-most sqweezable character from the  */
/* given SAVI string as marked by the sqweezable array.  Returns TRUE if a   */
/* character was sqweezed and FALSE otherwise                                */

sqweeze_SAVI_string(SAVI_string,sqweezable)
char SAVI_string[];
int sqweezable[];

{
   int last_char_num, sqweezed, i, j;
```

```
   sqweezed = FALSE;

   last_char_num = strlen(SAVI_string) - 1;

   /* find right-most sqweezable character */
   for (i=last_char_num;i>=0;i--)
   {
      /* sqweeze out right-most sqweezable character */
      if (sqweezable[i])
      {
         for (j=i;j<=last_char_num;j++)
         {
            SAVI_string[j] = SAVI_string[j+1];
            sqweezable[j] = sqweezable[j+1];
         }
         sqweezed = TRUE;
         break;
      }
   }

   return(sqweezed);
}

/* ------------------------------------------------------------------------ */

display_SAVI_string(left_col,SAVI_string,sqweezable)
int left_col;
char SAVI_string[];
int sqweezable[];

{
   int done, i, string_len;

   done = FALSE;

   sw_att(LCYAN,&intro_wn);
   v_plst(4,left_col,SAVI_string,&intro_wn);

   /* highlight the not-sqweezable letters */
   string_len = strlen(SAVI_string);
   for (i=0;i<string_len;i++)
   {
      if (!sqweezable[i])
      {
         mv_cs(4,left_col+i,&intro_wn);
         v_natt(LWHITE,CH,&intro_wn);
      }
   }

   return;
}

/* ------------------------------------------------------------------------ */
```

```
/* shift_SAVI_box moves the right or left end of the SAVI box one character  */
/* toward the center of the screen and adjusts the left or right col value   */
/* as appropriate.  direct must be set to LEFT or RIGHT                      */

shift_SAVI_box(direct,left_col,right_col)
int direct, *left_col, *right_col;

{
   sw_att(LBLUE,&intro_wn);
   switch(direct)
   {
      case LEFT  : v_plst(3,*left_col," ZD",&intro_wn);
                   v_plst(4,*left_col," 3 ",&intro_wn);
                   v_plst(5,*left_col," ∂D",&intro_wn);
                   *left_col = *left_col + 1;
                   break;

      case RIGHT : v_plst(3,*right_col-2,"D? ",&intro_wn);
                   v_plst(4,*right_col-2," 3 ",&intro_wn);
                   v_plst(5,*right_col-2,"DY ",&intro_wn);
                   *right_col = *right_col - 1;
                   break;
   }

   return;
}

/* ------------------------------------------------------------------------- */

/* display_intro_PE displays a PE on the intro screen at the given upper     */
/* left column location                                                      */

display_intro_PE(upper_left_col)
int upper_left_col;

{
   sw_att(LGREEN,&intro_wn);

   v_plst(15,upper_left_col+2,"B",&intro_wn);
   v_plst(16,upper_left_col,"ZDAD?",&intro_wn);
   v_plst(17,upper_left_col,"3   3",&intro_wn);
   v_plst(18,upper_left_col,"∂DBDY",&intro_wn);
   v_plst(19,upper_left_col+2,"A",&intro_wn);

   return;
}

/* ------------------------------------------------------------------------- */

/* display_now_loading displays "NOW LOADING" on the SAVI intro screen       */

display_now_loading()
```

```
{
   sw_att(LBLUE,&intro_wn);
   mv_cs(11,32,&intro_wn);
   v_st("ZDDDDDDDDDDDDD?",&intro_wn);
   mv_cs(12,32,&intro_wn);
   v_st("3             3",&intro_wn);
   mv_cs(13,32,&intro_wn);
   v_st("@DDDDDDDDDDDDDY",&intro_wn);

   sw_att(LWHITBLNK,&intro_wn);
   mv_cs(12,34,&intro_wn);
   v_st("NOW LOADING",&intro_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* remove_intro removes the intro screen                                    */

remove_intro()

{
  unset_wn(&intro_wn);
  return;
}

/* ------------------------------------------------------------------------ */

/* display_quit displays the SAVI quit screen and prepares to quit SAVI     */

display_quit()

{
   /* place the screen cursor at the lower left corner */
   mv_csr(23,0,&wn0);

   /* set the clear screen color attribute back to DOS */
   cl_att = LDOS;

   /* clear the screen */
   cls();

   return;
}

/* ------------------------------------------------------------------------ */

/* display_current_ASD displays the current ASD in the ASD window           */

display_current_ASD()

{
   int layer, PE_num, area_num;
```

```
   /* set the ASD window on the screen */
   set_wn(&ASD_wn);

   /* display the ASD frame */
   display_ASD_frame();

   /* display the current ASD's modifiable area labels */
   for (area_num = 1; area_num <= MAX_MODIFIABLE_AREAS; area_num++)
      display_area_label(area_num);

   /* display the ASD label centered in the ASD frame's label window */
   display_ASD_label(get_ASD_label());

   /* display the ASD analysis state */
   display_analysis_state();

   /* display all present PE's in the ASD */
   display_all_present_PEs();

   /* display the task PE */
   display_PE(get_ASD_task_PE_ptr(),TASK_PE_LAYER,TASK_PE_NUM);

   return;
}

/* ------------------------------------------------------------------------ */

/* display_ASD_label displays the given ASD_label within the label window   */
/* on the current ASD frame in the ASD window                               */

display_ASD_label(ASD_label)
char *ASD_label;

{
   /* set the ASD label attribute */
   sw_att(LCYANHI,&ASD_wn);

   /* clear the ASD label field */
   mv_cs(ASD_LABEL_ROW,ASD_LABEL_COL,&ASD_wn);
   v_qch(' ',MAX_ASD_LABEL_LEN-1,&ASD_wn);

   /* display the ASD label centered in the ASD frame's label window */
   mv_cs(ASD_LABEL_ROW,
         ASD_LABEL_COL+(MAX_ASD_LABEL_LEN-1-strlen(ASD_label))/2,
         &ASD_wn);
   v_st(ASD_label,&ASD_wn);

   /* set the ASD window attribute back to normal */
   sw_att(LNORMAL,&ASD_wn);

   return;
}
```

```
/* ------------------------------------------------------------------------ */

/* display_area_label displays the current area label for the given area     */
/* and the associated area label box on the current ASD frame                */
/* area_num must be in the range (1 to MAX_MODIFIABLE_AREAS)                 */

display_area_label(area_num)
int area_num;

{
   int label_row;
   char *temp_area_label;

   if ( (area_num < 1) || (area_num > MAX_MODIFIABLE_AREAS) ) return;

   /* calculate the area label's top line row within the ASD window */
   label_row = area_num * 4 - 1;

   /* display the area label box */
   /* dash if this area allows exterior traversal */

   sw_att(LGREEN,&ASD_wn);

   if (get_ASD_area_type(area_num) == EXTERIOR)
   {
      mv_cs(label_row-1,1,&ASD_wn);
      v_st("Z D D D D D D D ?",&ASD_wn);
      mv_cs(label_row,1,&ASD_wn);
      v_st("3               3",&ASD_wn);
      mv_cs(label_row+1,1,&ASD_wn);
      v_st("3               3",&ASD_wn);
      mv_cs(label_row+2,1,&ASD_wn);
      v_st("@ D D D D D D D Y",&ASD_wn);
   }
   else
   {
      mv_cs(label_row-1,1,&ASD_wn);
      v_st("ZDDDDDDDDDDDDDDD?",&ASD_wn);
      mv_cs(label_row,1,&ASD_wn);
      v_st("3               3",&ASD_wn);
      mv_cs(label_row+1,1,&ASD_wn);
      v_st("3               3",&ASD_wn);
      mv_cs(label_row+2,1,&ASD_wn);
      v_st("@DDDDDDDDDDDDDDDY",&ASD_wn);
      mv_cs(label_row-1,1,&ASD_wn);
   }


   /* display the area label */

   /* set the area label attribute */
   sw_att(LWHITE,&ASD_wn);

   /* display the area label's first and second lines */
```

```
   temp_area_label = get_ASD_area_label_first_line(area_num);
   mv_cs(label_row,
         ASD_AREA_LABEL_COL+(MAX_HALF_AREA_LABEL_LEN-1-
                             strlen(temp_area_label))/2,&ASD_wn);
   v_st(temp_area_label,&ASD_wn);
   temp_area_label = get_ASD_area_label_second_line(area_num);
   mv_cs(label_row+1,
         ASD_AREA_LABEL_COL+(MAX_HALF_AREA_LABEL_LEN-1-
                             strlen(temp_area_label))/2,&ASD_wn);
   v_st(temp_area_label,&ASD_wn);

   /* set the ASD window attribute back to normal */
   sw_att(LNORMAL,&ASD_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* print_current_ASD prints the current ASD to the given stream             */

print_current_ASD(stream)
FILE *stream;

{
   char window_row[81];
   int line_count, row_count, escaped, i;

   escaped = FALSE;
   line_count = 0;

   /* print the ASD header */
   fprintf(stream,"\n\n\n\n\n\n");
   line_count = line_count + 6;
   if (search_key_buf(ESC))
      escaped = TRUE;
   if (!escaped)
   {
      fprintf(stream,"                          \
- SAVI Adversary Sequence Diagram -\n");
      line_count++;
   }
   if (search_key_buf(ESC))
      escaped = TRUE;
   if (!escaped)
   {
      fprintf(stream,"\n\n\n\n\n\n");
      line_count = line_count + 6;
   }

   /* print the ASD */
   row_count = 0;
   while ( (row_count < 23) && !escaped)
   {
```

```
            if (search_key_buf(ESC))
            {
               escaped = TRUE;
            }
            else
            {
               mv_cs(row_count,0,&ASD_wn);
               v_mova(window_row, &ASD_wn, ROW, OUT);

               /* replace any non-printable arrow characters with */
               /* printable characters                            */
               for (i = 0; i < 81; i++)
               {
') window_row[i] = '^';
                  if (window_row[i] == '') window_row[i] = 'v';
               }

               fprintf(stream,"%s\n",window_row);
               row_count++;
               line_count++;
            }
         }

         line_feed(66-line_count);

         return;
      }

      /* ------------------------------------------------------------------------ */

      /* display_all_present_PEs displays all PEs which are present in the current */
      /* ASD                                                                       */

      display_all_present_PEs()

      {
         int layer;

         for (layer = 0; layer < MAX_LAYERS_PER_ASD; layer++)
         {
            display_present_PE_layer(layer,0);
         }

         return;
      }

      /* ------------------------------------------------------------------------ */

      /* erase_all_PEs_except_task erases all PEs on the screen except the task PE */

      erase_all_PEs_except_task()

      {
         int layer, PE_num;
```

```
   for (layer = 0; layer < MAX_LAYERS_PER_ASD; layer++)
   {
      for (PE_num = 0; PE_num < MAX_DISPLAY_PE_PER_LAYER; PE_num++)
      {
         erase_PE(layer,PE_num);
      }
   }

   return;
}

/* ------------------------------------------------------------------------ */

/* display_all_non_present_PEs displays all non-null PEs in the current ASD  */
/* which are not currently present                                           */

display_all_non_present_PEs()

{
   int layer;

   for (layer = 0; layer < MAX_LAYERS_PER_ASD; layer++)
   {
      display_non_present_PE_layer(layer,0);
   }

   return;
}

/* ------------------------------------------------------------------------ */

/* erase_all_non_present_PEs erases all non-null PEs in the current ASD      */
/* which are not currently present                                           */

erase_all_non_present_PEs()

{
   int layer;

   for (layer = 0; layer < MAX_LAYERS_PER_ASD; layer++)
   {
      erase_non_present_PE_layer(layer,0);
   }

   return;
}

/* ------------------------------------------------------------------------ */

/* display_present_PE_layer displays all present PE's in the current ASD on  */
/* the given layer starting at the given PE. If PE's exist to the left or    */
/* right of the PE's which are displayed, arrows indicating this are also    */
/* displayed                                                                 */
```

```
/* If no present PE's exist on the current layer (bypass is set in current   */
/* ASD for this layer) then a bypass is drawn                                 */

display_present_PE_layer(layer,start_PE_num)
int layer, start_PE_num;

{
   int PE_num, end_PE_num;
   PROT_ELEM *PE_ptr;

   end_PE_num = MAX_DISPLAY_PE_PER_LAYER + start_PE_num - 1;

   /* check for layer bypass */
   if (get_ASD_layer_bypass(layer))
   {
      display_layer_bypass(layer);
      return;
   }

   /* display left arrow if PEs are present to left of start PE */
   if (start_PE_num != 0 && PEs_present(layer,0,start_PE_num-1))
      display_PE_layer_arrow(LEFT,layer);

   /* display all present PEs on given layer from start_PE to end_PE */
   for (PE_num = start_PE_num; PE_num <= end_PE_num; PE_num++)
      {
         PE_ptr = get_ASD_PE_ptr(layer,PE_num);
         if (PE_ptr->present)
            display_PE(PE_ptr,layer,PE_num);
         else
            erase_PE(layer,PE_num);
      }

   /* display right arrow if PEs are present to right of end PE */
   if (end_PE_num != (MAX_PE_PER_LAYER-1) &&
       PEs_present(layer,end_PE_num+1,MAX_PE_PER_LAYER-1))
      display_PE_layer_arrow(RIGHT,layer);

   return;
}

/* ------------------------------------------------------------------------- */

/* display_non_null_PE_layer displays all non-null PE's in the current       */
/* ASD (present or not) on the given layer starting at the given PE          */
/* If the given layer's bypass is set then the bypass is also drawn          */

display_non_null_PE_layer(layer, start_PE_num)
int layer, start_PE_num;

{
   int PE_num, end_PE_num;
   PROT_ELEM *PE_ptr;
```

```
   /* display bypass state */
   if (get_ASD_layer_bypass(layer))
      display_layer_bypass(layer);
   else
      erase_layer_bypass(layer);

   end_PE_num = MAX_DISPLAY_PE_PER_LAYER + start_PE_num - 1;

   /* display all non-null PEs on given layer */
   /* from start_PE to end_PE                 */
   for (PE_num = start_PE_num; PE_num <= end_PE_num; PE_num++)
      {
         PE_ptr = get_ASD_PE_ptr(layer,PE_num);
         if (PE_ptr->type != 0)
            display_PE(PE_ptr,layer,PE_num);
         else
            erase_PE(layer,PE_num);
      }

   return;
}

/* ------------------------------------------------------------------------ */

/* display_non_present_PE_layer displays all non-null PE's in the current    */
/* ASD which are not currently present on the given layer starting at the    */
/* given PE                                                                  */
/* If non-null PE's exist to the left or right of the PE's which are         */
/* displayed, arrows indicating this are also displayed                      */

display_non_present_PE_layer(layer,start_PE_num)
int layer, start_PE_num;

{
   int PE_num, end_PE_num;
   PROT_ELEM *PE_ptr;

   end_PE_num = MAX_DISPLAY_PE_PER_LAYER + start_PE_num - 1;

   /* display left arrow if non_null PEs exist to left of start PE */
   if (start_PE_num != 0 &&
       non_null_PEs_exist(layer,0,start_PE_num-1))
      display_PE_layer_arrow(LEFT,layer);

   /* display all non-null non-present PEs on given layer */
   /* from start_PE to end_PE                             */
   for (PE_num = start_PE_num; PE_num <= end_PE_num; PE_num++)
      {
         PE_ptr = get_ASD_PE_ptr(layer,PE_num);
         if ( (PE_ptr->type != 0) && !(PE_ptr->present) )
            display_PE(PE_ptr,layer,PE_num);
      }

   /* display right arrow if non_null PEs exist to right of end PE */
```

```
   if (end_PE_num != (MAX_PE_PER_LAYER-1) &&
       non_null_PEs_exist(layer,end_PE_num+1,MAX_PE_PER_LAYER-1))
      display_PE_layer_arrow(RIGHT,layer);

   return;
}

/* ------------------------------------------------------------------------ */

/* erase_non_present_PE_layer erases all non-null PE's in the current       */
/* ASD which are not currently present on the given layer starting at the   */
/* given PE                                                                 */
/* If non-null PE's exist to the left or right of the PE's which are        */
/* displayed, arrows indicating this are also erased                        */

erase_non_present_PE_layer(layer,start_PE_num)
int layer, start_PE_num;

{
   int PE_num, end_PE_num;
   PROT_ELEM *PE_ptr;

   end_PE_num = MAX_DISPLAY_PE_PER_LAYER + start_PE_num - 1;

   /* erase left arrow if non_null PEs exist to left of start PE */
   if (start_PE_num != 0 &&
       non_null_PEs_exist(layer,0,start_PE_num-1))
      erase_PE_layer_arrow(LEFT,layer);

   /* erase all non-null non-present PEs on given layer   */
   /* from start_PE to end_PE                             */
   for (PE_num = start_PE_num; PE_num <= end_PE_num; PE_num++)
      {
         PE_ptr = get_ASD_PE_ptr(layer,PE_num);
         if ( (PE_ptr->type != 0) && !(PE_ptr->present) )
            erase_PE(layer,PE_num);
      }

   /* erase right arrow if non_null PEs exist to right of end PE */
   if (end_PE_num != (MAX_PE_PER_LAYER-1) &&
       non_null_PEs_exist(layer,end_PE_num+1,MAX_PE_PER_LAYER-1))
      erase_PE_layer_arrow(RIGHT,layer);

   return;
}

/* ------------------------------------------------------------------------ */

/* PEs_present returns TRUE if any PEs are present from start_PE_num to     */
/* end_PE_num on the given layer and returns FALSE otherwise                */

PEs_present(layer,start_PE_num,end_PE_num)
int layer, start_PE_num, end_PE_num;
```

```
{
   int PE_num;

   /* if layer is the task PE layer return TRUE */
   /* since the task PE is always present       */
   if (layer == MAX_LAYERS_PER_ASD) return(TRUE);

   for (PE_num = start_PE_num; PE_num <= end_PE_num; PE_num++)
   {
      if (get_ASD_PE_ptr(layer,PE_num)->present) return(TRUE);
   }

   return(FALSE);
}

/* ------------------------------------------------------------------------ */

/* non_null_PEs_exist returns TRUE if any non-null PEs exist from           */
/* start_PE_num to end_PE_num on the given layer and returns FALSE otherwise */

non_null_PEs_exist(layer,start_PE_num,end_PE_num)
int layer, start_PE_num, end_PE_num;

{
   int PE_num;

   /* if layer is the task PE layer return TRUE */
   /* since the task PE is always present       */
   if (layer == MAX_LAYERS_PER_ASD) return(TRUE);

   for (PE_num = start_PE_num; PE_num <= end_PE_num; PE_num++)
   {
      if (get_ASD_PE_ptr(layer,PE_num)->type != 0) return(TRUE);
   }

   return(FALSE);
}

/* ------------------------------------------------------------------------ */

/* last_non_null_PE_num_on_layer returns the PE number of the last non-null  */
/* PE on the given layer. (Note: the caller must ensure that at least one    */
/* non-null PE exists on the given layer before making the call)             */

last_non_null_PE_num_on_layer(layer)
int layer;

{
   int PE_num;

   /* if layer is the task PE layer return the task PE number */
   if (layer == MAX_LAYERS_PER_ASD) return(0);

   for (PE_num = MAX_PE_PER_LAYER-1; PE_num >= 0; PE_num--)
```

```
   {
      if (get_ASD_PE_ptr(layer,PE_num)->type != 0) return(PE_num);
   }

   return(0);
}

/* ------------------------------------------------------------------------ */

/* first_present_PE_num_on_layer returns the PE number of the first         */
/* present PE on the given layer. Note: the task PE is always present       */
/* and is considered to reside on layer MAX_LAYERS_PER_ASD                  */

first_present_PE_num_on_layer(layer)
int layer;

{
   int PE_num;

   /* if layer is the task PE layer return the task PE number */
   if (layer == MAX_LAYERS_PER_ASD) return(0);

   for (PE_num = 0; PE_num < MAX_PE_PER_LAYER; PE_num++)
   {
      if (get_ASD_PE_ptr(layer,PE_num)->present) return(PE_num);
   }

   return(0);
}

/* ------------------------------------------------------------------------ */

/* last_present_PE_num_on_layer returns the PE number of the last           */
/* present PE on the given layer. Note: the task PE is always present       */
/* and is considered to reside on layer MAX_LAYERS_PER_ASD                  */

last_present_PE_num_on_layer(layer)
int layer;

{
   int PE_num;

   /* if layer is the task PE layer return the task PE number */
   if (layer == MAX_LAYERS_PER_ASD) return(0);

   for (PE_num = MAX_PE_PER_LAYER-1; PE_num >= 0; PE_num--)
   {
      if (get_ASD_PE_ptr(layer,PE_num)->present) return(PE_num);
   }

   return(0);
}

/* ------------------------------------------------------------------------ */
```

```
/* closest_present_PE_num returns the number of the closest present PE to    */
/* the given start PE on the given layer. If no other present PEs exist, the */
/* start_PE_num is returned                                                   */

closest_present_PE_num(layer,start_PE_num)
int layer, start_PE_num;

{
   int PE_left, PE_right;

   /* if layer is the task PE layer return the task PE number */
   if (layer == MAX_LAYERS_PER_ASD) return(0);

   PE_left = closest_left_present_PE_num(layer,start_PE_num);
   PE_right = closest_right_present_PE_num(layer,start_PE_num);

   /* if no PEs are present to the right then return */
   /* the closest present PE on the left             */
   if (PE_right == start_PE_num) return(PE_left);

   /* if no PEs are present to the left then return */
   /* the closest present PE on the right          */
   if (PE_left == start_PE_num) return(PE_right);

   /* present PEs exist on both sides of the start PE */

   /* if the closest PE on the right is further  */
   /* from the start PE than the closest PE on   */
   /* the left then return the PE on the left    */
   /* in case of a tie return the PE on the left */
   if  ( (PE_right - start_PE_num) >= (start_PE_num - PE_left) )
      return(PE_left);
   else
      return(PE_right);
}

/* ------------------------------------------------------------------------ */

/* closest_left_present_PE_num returns the number of the closest present PE  */
/* to the left of the given start PE on the given layer. If no present PEs   */
/* exist to the left of the given start PE, the start_PE_num is returned     */

closest_left_present_PE_num(layer,start_PE_num)
int layer, start_PE_num;

{
   int PE_num;

   /* if layer is the task PE layer return the task PE number */
   if (layer == MAX_LAYERS_PER_ASD) return(0);

   if (start_PE_num == 0) return(start_PE_num);
```

```
   for (PE_num = start_PE_num-1; PE_num >= 0; PE_num--)
   {
      if (get_ASD_PE_ptr(layer,PE_num)->present) return(PE_num);
   }

   return(start_PE_num);
}

/* ------------------------------------------------------------------------ */

/* closest_right_present_PE_num returns the number of the closest present    */
/* PE to the right of the given start PE on the given layer. If no present   */
/* PEs exist to the right of the given start PE, the start_PE_num is         */
/* returned                                                                  */

closest_right_present_PE_num(layer,start_PE_num)
int layer, start_PE_num;

{
   int PE_num;

   /* if layer is the task PE layer return the task PE number */
   if (layer == MAX_LAYERS_PER_ASD) return(0);

   if (start_PE_num == MAX_PE_PER_LAYER-1) return(start_PE_num);

   for (PE_num = start_PE_num+1; PE_num < MAX_PE_PER_LAYER; PE_num++)
   {
      if (get_ASD_PE_ptr(layer,PE_num)->present) return(PE_num);
   }

   return(start_PE_num);
}

/* ------------------------------------------------------------------------ */

/* first_layer_with_PEs_present returns the first layer in the current ASD   */
/* which has at least one present PE. Note: the task PE is always present    */
/* and is considered to reside on layer MAX_LAYERS_PER_ASD                   */

first_layer_with_PEs_present()

{
   int layer;

   for (layer = 0; layer <= MAX_LAYERS_PER_ASD; layer++)
   {
      if (PEs_present(layer,0,MAX_PE_PER_LAYER-1))
         return(layer);
   }

}

/* ------------------------------------------------------------------------ */
```

```
/* last_layer_with_PEs_present returns the last layer in the current ASD     */
/* which has at least one present PE. Note: the task PE is always present    */
/* and is considered to reside on layer MAX_LAYERS_PER_ASD                    */

last_layer_with_PEs_present()

{
   int layer;

   for (layer = MAX_LAYERS_PER_ASD; layer >= 0; layer--)
   {
      if (PEs_present(layer,0,MAX_PE_PER_LAYER-1))
         return(layer);
   }

}

/* ------------------------------------------------------------------------- */

/* layer_with_PEs_present_above returns the first layer in the current ASD   */
/* above the given start_layer which has at least one present PE             */
/* If no layers above the start_layer contain present PEs then the start     */
/* layer is returned                                                         */

layer_with_PEs_present_above(start_layer)
int start_layer;

{
   int layer;

   if (start_layer == 0) return(start_layer);

   for (layer = start_layer-1; layer >= 0; layer--)
   {
      if (PEs_present(layer,0,MAX_PE_PER_LAYER-1))
         return(layer);
   }

   return(start_layer);
}

/* ------------------------------------------------------------------------- */

/* layer_with_PEs_present_below returns the first layer in the current ASD   */
/* below the given start_layer which has at least one present PE             */
/* If no layers below the start_layer contain present PEs then the start     */
/* layer is returned                                                         */

layer_with_PEs_present_below(start_layer)
int start_layer;

{
   int layer;
```

```
   if (start_layer == MAX_LAYERS_PER_ASD) return(start_layer);

   for (layer = start_layer+1; layer <= MAX_LAYERS_PER_ASD; layer++)
   {
      if (PEs_present(layer,0,MAX_PE_PER_LAYER-1))
         return(layer);
   }

   return(start_layer);
}

/* ------------------------------------------------------------------------ */

/* returns the serial number of the given PE on the given layer             */

PE_num_on_layer(layer, PE_num)
int layer, PE_num;

{
   int PE, count;

   count = 0;

   /* if layer is the task PE layer return 1 since the task PE */
   /* is always present and in the first position on the layer */
   if (layer == MAX_LAYERS_PER_ASD) return(1);

   for (PE = 0; PE < MAX_PE_PER_LAYER; PE++)
   {
      if (get_ASD_PE_ptr(layer,PE)->present) count++;
      if (PE == PE_num) return(count);
   }

   return(0);
}

/* ------------------------------------------------------------------------ */

/* display_PE_layer_arrow displays an arrow at the end of the given layer   */
/* pointing in the direction of dir to indicate that more PEs are present   */
/* on the given layer in the given direction                                */

display_PE_layer_arrow(dir,layer)
int dir, layer;

{
   int base_row, base_col;

   base_row = 3 + (layer * 4);

   /* set the arrow attribute */
   sw_att(LRED,&ASD_wn);
```

```
    if (dir == LEFT)
    {
       if (layer%2 != 0)
          base_col = 20;
       else
          base_col = 18;

       mv_cs(base_row,base_col,&ASD_wn);
       v_st("",&ASD_wn);
    }
    else
    {
       if (layer%2 != 0)
          base_col = 76;
       else
          base_col = 74;

       mv_cs(base_row,base_col,&ASD_wn);
       v_st("",&ASD_wn);
    }

    /* set the ASD attribute back to normal */
    sw_att(LNORMAL,&ASD_wn);

    return;
}

/* ------------------------------------------------------------------------ */

/* erase_PE_layer_arrow erases the arrow at the end of the given layer      */
/* pointing in the direction of dir                                         */

erase_PE_layer_arrow(dir,layer)
int dir, layer;

{
    int base_row, base_col;

    base_row = 3 + (layer * 4);

    if (dir == LEFT)
    {
       if (layer%2 != 0)
          base_col = 20;
       else
          base_col = 18;

       mv_cs(base_row,base_col,&ASD_wn);
       v_st(" ",&ASD_wn);
    }
    else
    {
       if (layer%2 != 0)
          base_col = 76;
```

```
      else
         base_col = 74;

      mv_cs(base_row,base_col,&ASD_wn);
      v_st(" ",&ASD_wn);
   }

   return;
}

/* ------------------------------------------------------------------------ */

/* display_PE displays the PE which PE_ptr is pointing to at the location    */
/* specified by layer and PE_num in the ASD window                           */
/* If the PE is currently present in the current ASD then the PE is rendered */
/* completely, else it is rendered as a non-present PE                       */

/* examples:          Present PE's              Non_Present PE's             */
/*                      Q     Q                    M     M                   */
/*                    ZDAD? ZDAD?                Z A ? Z A ?                 */
/*                    3PER3 3MAT3                 PER   MAT                  */
/*                    @?CZY @DBDY                @?CZY @ B Y                 */
/*                      M     0                    M     M                   */

display_PE(PE_ptr,layer,PE_num)
PROT_ELEM *PE_ptr;
int layer, PE_num;

{
   int base_row, base_col, PE_present;
   char *jump_tab();

   PE_present = get_ASD_PE_ptr(layer,PE_num)->present;

   /* determine the base row and column values for the PE */
   base_row = PE_base_row(layer,PE_num);
   base_col = PE_base_col(layer,PE_num);

   /* set the ASD window attribute to normal */
   sw_att(LNORMAL,&ASD_wn);

   /* draw the direct input path segment to area link    */
   if (layer != 0)
   {
      mv_cs(base_row,
            base_col+2,
            &ASD_wn);
      if (PE_present)
         v_st("Q",&ASD_wn);
      else
         v_st("M",&ASD_wn);
   }

   mv_cs(base_row+1,
```

```
      base_col,
      &ASD_wn);
if (PE_present)
   v_st("ZDAD?",&ASD_wn);
else
   v_st("Z A ?",&ASD_wn);

/* draw the left PE box edge */
mv_cs(base_row+2,
      base_col,
      &ASD_wn);
if (PE_present)
   v_st("3",&ASD_wn);
else
   v_st(" ",&ASD_wn);

/* set the PE acronym attribute */
if (PE_present) sw_att(LWHITE,&ASD_wn);

/* display the PE acronym found in the PE_catalog */
v_st(get_PE_cat_spec_ptr(PE_ptr->type)->acronym,&ASD_wn);

/* set the ASD window attribute back to normal */
sw_att(LNORMAL,&ASD_wn);

/* draw the right PE box edge */
if (PE_present)
   v_st("3",&ASD_wn);
else
   v_st(" ",&ASD_wn);

/* if the output path segment is direct then draw */
/* the direct output path segment to area link    */
/* else draw the output jump tab */
if (layer+1 == PE_ptr->output_area)
{
   mv_cs(base_row+3,
         base_col,
         &ASD_wn);
   if (PE_present)
      v_st("3DBDY",&ASD_wn);
   else
      v_st("3 B Y",&ASD_wn);

   if (layer != MAX_LAYERS_PER_ASD)
   {
      mv_cs(base_row+4,
            base_col+2,
            &ASD_wn);
      if (PE_present)
         v_st("D",&ASD_wn);
      else
         v_st("M",&ASD_wn);
   }
```

```
    }
    else
    {
       mv_cs(base_row+3,
             base_col,
             &ASD_wn);
       v_st("@?",&ASD_wn);

       /* set the jump tab attribute */
       sw_att(LGREENHI,&ASD_wn);
       v_st(jump_tab(PE_ptr->output_area),&ASD_wn);

       /* set the ASD window attribute back to normal */
       sw_att(LNORMAL,&ASD_wn);

       v_st("ZY",&ASD_wn);

       if (layer != MAX_LAYERS_PER_ASD)
       {
          mv_cs(base_row+4,
                base_col+2,
                &ASD_wn);
          v_st("M",&ASD_wn);
       }
    }

    return;
}

/* ------------------------------------------------------------------------ */

/* mark_all_PEs marks the state of the delay/detection component choices on  */
/* all PEs (present or not) in the current ASD                               */

mark_all_PEs()

{
    int layer, PE_num;

    for (layer = 0; layer < MAX_LAYERS_PER_ASD; layer++)
    {
       for (PE_num = 0; PE_num < MAX_DISPLAY_PE_PER_LAYER; PE_num++)
       {
          mark_PE_comps(layer,PE_num);
       }
    }

    /* mark the task PE */
    mark_PE_comps(MAX_LAYERS_PER_ASD,0);

    return;
}

/* ------------------------------------------------------------------------ */
```

```
/* mark_all_present_PEs marks the state of the delay/detection component     */
/* choices on all present PEs in the current ASD                             */

mark_all_present_PEs()

{
   int layer, PE_num;

   for (layer = 0; layer < MAX_LAYERS_PER_ASD; layer++)
   {
      for (PE_num = 0; PE_num < MAX_DISPLAY_PE_PER_LAYER; PE_num++)
      {
         /* do not mark if the PE is not currently present in the ASD */
         if (get_ASD_PE_ptr(layer,PE_num)->present)
            mark_PE_comps(layer,PE_num);
      }
   }

   /* mark the task PE */
   mark_PE_comps(MAX_LAYERS_PER_ASD,0);

   return;
}

/* ------------------------------------------------------------------------- */

/* unmark_all_present_PEs unmarks the state of the delay/detection component */
/* choices on all present PEs in the current ASD                             */

unmark_all_present_PEs()

{
   int layer, PE_num;

   for (layer = 0; layer < MAX_LAYERS_PER_ASD; layer++)
   {
      for (PE_num = 0; PE_num < MAX_DISPLAY_PE_PER_LAYER; PE_num++)
      {
         /* do not unmark if the PE is not currently present in the ASD */
         if (get_ASD_PE_ptr(layer,PE_num)->present)
            unmark_PE_comps(layer,PE_num);
      }
   }

   /* unmark the task PE */
   unmark_PE_comps(MAX_LAYERS_PER_ASD,0);

   return;
}

/* ------------------------------------------------------------------------- */

/* mark_PE_comps marks the state of the delay/detection component choices on */
```

```
/* the given PE specified by layer and PE_num with the current ASD.  If any  */
/* components are CLEAR a ? is marked on the upper left side of the PE.  If  */
/* any components have been directly set (DIRECT) by the user then a # is    */
/* marked on the upper right side of the PE.  Otherwise the PE is unmarked.  */

/* examples:    clear & direct    clear    direct    all chosen              */
/*                    Q              Q        Q           Q                   */
/*                  Z?A#?          Z?AD?    ZDA#?       ZDAD?               */
/*                  3PER3          3MAT3    3PER3       3MAT3               */
/*                  @?CZY          @DBDY    @?CZY       @DBDY               */
/*                    M              O        M           O                   */

mark_PE_comps(layer,PE_num)
int layer, PE_num;

{
   PROT_ELEM *PE_ptr;
   int base_row, base_col;

   PE_ptr = get_ASD_PE_ptr(layer,PE_num);

   /* determine the base row and column values for the PE */
   base_row = PE_base_row(layer,PE_num);
   base_col = PE_base_col(layer,PE_num);

   /* mark the PE with a ? if any delay or detection components are */
   /* CLEAR within this PE                                          */
   if (any_PE_comp_type_choices_are_val(layer,PE_num,DELAY,CLEAR) ||
       any_PE_comp_type_choices_are_val(layer,PE_num,NONDET,CLEAR))
   {
      sw_att(LWHITEHI,&ASD_wn);
      v_plst(base_row+1,base_col+1,"?",&ASD_wn);
   }
   else
   {
      sw_att(LNORMAL,&ASD_wn);
      if (PE_ptr->present)
         v_plst(base_row+1,base_col+1,"D",&ASD_wn);
      else
         v_plst(base_row+1,base_col+1," ",&ASD_wn);
   }

   /* mark the PE with a # if any delay or detection components are */
   /* DIRECT within this PE                                         */
   if (any_PE_comp_type_choices_are_val(layer,PE_num,DELAY,DIRECT) ||
       any_PE_comp_type_choices_are_val(layer,PE_num,NONDET,DIRECT))
   {
      sw_att(LWHITEHI,&ASD_wn);
      v_plst(base_row+1,base_col+3,"#",&ASD_wn);
   }
   else
   {
      sw_att(LNORMAL,&ASD_wn);
      if (PE_ptr->present)
```

```
         v_plst(base_row+1,base_col+3,"D",&ASD_wn);
      else
         v_plst(base_row+1,base_col+3," ",&ASD_wn);
   }

   /* set the ASD attribute back to normal */
   sw_att(LNORMAL,&ASD_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* unmark_PE_comps unmarks the state of the delay/detection component       */
/* choices on the given PE specified by layer and PE_num in the current ASD */

unmark_PE_comps(layer,PE_num)
int layer, PE_num;

{
   PROT_ELEM *PE_ptr;
   int base_row, base_col;

   PE_ptr = get_ASD_PE_ptr(layer,PE_num);

   /* determine the base row and column values for the PE */
   base_row = PE_base_row(layer,PE_num);
   base_col = PE_base_col(layer,PE_num);

   sw_att(LNORMAL,&ASD_wn);

   /* unmark the PE */
   if (PE_ptr->present)
   {
      v_plst(base_row+1,base_col+1,"D",&ASD_wn);
      v_plst(base_row+1,base_col+3,"D",&ASD_wn);
   }
   else
   {
      v_plst(base_row+1,base_col+1," ",&ASD_wn);
      v_plst(base_row+1,base_col+3," ",&ASD_wn);
   }

   return;
}

/* ------------------------------------------------------------------------ */

/* mark_PE_traversal marks the traversal direction of the given PE including */
/* indication of authorized or unauthorized traversal                        */

/* examples:    authorized  unauthorized  authorized    authorized entry     */
/*                entry         exit      entry/exit   unauthorized exit     */
/*                  Q           Q            Q              Q               */
```

```
?          */
          /*                3PER3          3MAT          3PER3          3MAT          */
Y          */
          /*                  M             0             M             0             */

          mark_PE_traversal(layer, PE_num, direction, auth_unauth)
          int layer, PE_num, direction, auth_unauth;

          {
             int base_row, base_col;

             /* determine the base row and column values for the PE */
             base_row = PE_base_row(layer,PE_num);
             base_col = PE_base_col(layer,PE_num);

             sw_att(LWHITEHI,&ASD_wn);

             if (direction == ENTRY)
             {
                v_plst(base_row+1,base_col+1,"",&ASD_wn);
                v_plst(base_row+3,base_col+1,"",&ASD_wn);
                if (auth_unauth == UNAUTH)
                   v_plst(base_row+2,base_col," ",&ASD_wn);
             }
             else
             {
",&ASD_wn);
",&ASD_wn);
                if (auth_unauth == UNAUTH)
                   v_plst(base_row+2,base_col+4," ",&ASD_wn);
             }

             return;
          }

          /* ------------------------------------------------------------------------ */

          /* jump_tab returns a single character string containing the appropriate    */
          /* character representing area number area                                  */

          char *jump_tab(area)
          int area;

          {
             static char c[2] = {0,0};

             /* area 2 is labeled with jump tab A */
             c[0] = 'A' + area - 2;

             return(c);
          }

          /* ------------------------------------------------------------------------ */
```

```
/* display_layer_bypass displays a bypass bar between the areas which        */
/* bracket the given layer                                                   */

display_layer_bypass(layer)
int layer;

{
   int row_num;

   /* area 0 and the target area are not represented */
   /* so do not display bypass bars                  */
   if ((layer == 0) || (layer == MAX_LAYERS_PER_ASD)) return;

   row_num = (layer * 4) + 1;

   mv_cs(row_num,BYPASS_COL,&ASD_wn);
   v_st("B",&ASD_wn);

   mv_cs(++row_num,BYPASS_COL,&ASD_wn);
   v_st("A",&ASD_wn);

   return;
}

/* ------------------------------------------------------------------------- */

/* erase_layer_bypass erases the bypass bar between the areas which          */
/* bracket the given layer                                                   */

erase_layer_bypass(layer)
int layer;

{
   int row_num;

   /* area 0 and the target area are not represented */
   /* so do not display bypass bars                  */
   if ((layer == 0) || (layer == MAX_LAYERS_PER_ASD)) return;

   row_num = (layer * 4) + 1;

   mv_cs(row_num,BYPASS_COL,&ASD_wn);
   v_st("D",&ASD_wn);

   mv_cs(++row_num,BYPASS_COL,&ASD_wn);
   v_st("D",&ASD_wn);

   return;
}

/* ------------------------------------------------------------------------- */

/* display_ASD_frame displays the standard ASD frame in the ASD window       */
```

```
display_ASD_frame()

{
   /* display OFFSITE label */
   mv_cs(1,4,&ASD_wn);
",&ASD_wn);
   sw_att(LGREENHI,&ASD_wn);
   v_st("OFFSITE",&ASD_wn);
   sw_att(LNORMAL,&ASD_wn);
",&ASD_wn);

   /* display area 1 bar */
   mv_cs(3,0,&ASD_wn);
   v_st("MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM\
MMMMMMMMMMMMMMMM",&ASD_wn);

   /* display area 2 bar */
   mv_cs(7,0,&ASD_wn);
   v_st("MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM\
MMMMMMMMMMMMMM)",&ASD_wn);
   sw_att(LGREENHI,&ASD_wn);
   v_st("A",&ASD_wn);
   sw_att(LNORMAL,&ASD_wn);
   v_st("TM",&ASD_wn);

   /* display area 3 bar */
   mv_cs(11,0,&ASD_wn);
   v_st("MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM\
MMMMMMMMMMMMMM)",&ASD_wn);
   sw_att(LGREENHI,&ASD_wn);
   v_st("B",&ASD_wn);
   sw_att(LNORMAL,&ASD_wn);
   v_st("TM",&ASD_wn);

   /* display area 4 bar */
   mv_cs(15,0,&ASD_wn);
   v_st("MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM\
MMMMMMMMMMMMMM)",&ASD_wn);
   sw_att(LGREENHI,&ASD_wn);
   v_st("C",&ASD_wn);
   sw_att(LNORMAL,&ASD_wn);
   v_st("TM",&ASD_wn);

   /* display area 5 bar */
   mv_cs(19,0,&ASD_wn);
   v_st("MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM\
MMMMMMMMMMMMMM)",&ASD_wn);
   sw_att(LGREENHI,&ASD_wn);
   v_st("D",&ASD_wn);
   sw_att(LNORMAL,&ASD_wn);
   v_st("TM",&ASD_wn);

   /* display TARGET label */
   mv_cs(22,5,&ASD_wn);
```

```
    v_st(" ",&ASD_wn);
    sw_att(LGREENHI,&ASD_wn);
    v_st("TARGET",&ASD_wn);
    sw_att(LNORMAL,&ASD_wn);
    v_st(" ",&ASD_wn);

    /* display the ASD label border */
    sw_att(LBLUE,&ASD_wn);
    v_plst(20,25,
           "VDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD7",
           &ASD_wn);
    v_plst(21,25,
           ":                                                            :",
           &ASD_wn);
    v_plst(22,25,
           "SDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD=",
           &ASD_wn);
    sw_att(LNORMAL,&ASD_wn);

    return;
}

/* ------------------------------------------------------------------------ */

/* display_analysis_state indicates on the current ASD title whether the    */
/* current ASD has been analyzed                                            */

display_analysis_state()

{
    int analysis_current;

    analysis_current = get_analysis_state();

    sw_att(LBLUE,&ASD_wn);

    if (analysis_current)
    {
        v_plst(20,76,"7  ",&ASD_wn);
        v_plst(21,76,":  ",&ASD_wn);
        v_plst(22,76,"=  ",&ASD_wn);
    }
    else
    {
        v_plst(20,76,"BD?",&ASD_wn);
        v_plst(21,76,"3 3",&ASD_wn);
        v_plst(22,76,"ADY",&ASD_wn);

        sw_att(LGREENHI,&ASD_wn);
        v_plst(21,77,"?",&ASD_wn);
    }

    sw_att(LNORMAL,&ASD_wn);
```

```
   return;
}

/* ------------------------------------------------------------------------ */

/* erase_PE erases the PE specified by layer and PE_num in the ASD window    */

erase_PE(layer,PE_num)
int layer, PE_num;

{
   int base_row, base_col;

   /* determine the base row and column values for the PE */
   base_row = PE_base_row(layer,PE_num);
   base_col = PE_base_col(layer,PE_num);

   sw_att(LNORMAL,&ASD_wn);

   /* replace the input area entry point */
   /* except in the case of PEs leading from OFFSITE */
   if (layer != 0)
   {
      mv_cs(base_row,
            base_col+2,
            &ASD_wn);
      v_st("M",&ASD_wn);
   }

   /* erase the top of the PE box */
   mv_cs(base_row+1,
         base_col,
         &ASD_wn);
   v_st("     ",&ASD_wn);

   /* erase the middle of the PE box */
   mv_cs(base_row+2,
         base_col,
         &ASD_wn);
   v_st("     ",&ASD_wn);

   /* erase the bottom of the PE box */
   mv_cs(base_row+3,
         base_col,
         &ASD_wn);
   v_st("     ",&ASD_wn);

   /* replace the output area entry point */
   /* except in the case of the TASK PE */
   if (layer != MAX_LAYERS_PER_ASD)
   {
      mv_cs(base_row+4,
            base_col+2,
            &ASD_wn);
```

```
      v_st("M",&ASD_wn);
   }

   return;
}

/* ------------------------------------------------------------------------ */

/* blink_PE blinks the PE specified by layer and PE_num                     */

blink_PE(layer,PE_num)
int layer, PE_num;

{
   int base_row, base_col;

   base_row = PE_base_row(layer,PE_num);
   base_col = PE_base_col(layer,PE_num);

   /* blink the PE box and path segments */
   set_area_att(base_row+1,base_col,3,5,LWHITBLNK,&ASD_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* highlight_PE highlights the PE specified by layer and PE_num             */

highlight_PE(layer,PE_num)
int layer, PE_num;

{
   int base_row, base_col;

   base_row = PE_base_row(layer,PE_num);
   base_col = PE_base_col(layer,PE_num);

   /* highlight the PE box and path segments */
   set_area_att(base_row+1,base_col,3,5,LWHITEHI,&ASD_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* unhighlight_PE unhighlights the PE specified by layer and PE_num by      */
/* redisplaying the PE in the current ASD window                            */

unhighlight_PE(layer,PE_num)
int layer, PE_num;

{
   display_PE(get_ASD_PE_ptr(layer,PE_num),layer,PE_num);
```

```
   return;
}

/* ------------------------------------------------------------------------ */

/* highlight_PE_segment highlights the PE segment specified by segment for  */
/* the PE given by layer and PE_num and highlights the correct direction    */
/* arrow based on direction                                                 */

highlight_PE_segment(layer,PE_num,segment,direction)
int layer, PE_num, segment, direction;

{
   int base_row, base_col;

   base_row = PE_base_row(layer,PE_num);
   base_col = PE_base_col(layer,PE_num);

   if (segment == PATH_IN)
   {
      if (direction == ENTRY)
      {
         mv_cs(base_row+1, base_col+1, &ASD_wn);
         v_natt(LURGENT,CH,&ASD_wn);
      }
      else
      {
         mv_cs(base_row+1, base_col+3, &ASD_wn);
         v_natt(LURGENT,CH,&ASD_wn);
      }
   }
   else
   {
      if (direction == ENTRY)
      {
         mv_cs(base_row+3, base_col+1, &ASD_wn);
         v_natt(LURGENT,CH,&ASD_wn);
      }
      else
      {
         mv_cs(base_row+3, base_col+3, &ASD_wn);
         v_natt(LURGENT,CH,&ASD_wn);
      }
   }

   return;
}

/* ------------------------------------------------------------------------ */

/* highlight_area_label highlights the area label associated with area_num  */
/* If area_num is not within the range 1 to MAX_MODIFIABLE_AREAS nothing is */
/* done                                                                     */
```

```
highlight_area_label(area_num)
int area_num;

{
   int label_row;

   if ( (area_num < 1) || (area_num > MAX_MODIFIABLE_AREAS) ) return;

   /* calculate the area label's top line row within the ASD window */
   label_row = area_num * 4 - 1;

   /* highlight the entire area label */
   set_area_att(label_row-1,ASD_AREA_LABEL_COL-2,
                4,MAX_HALF_AREA_LABEL_LEN+3,
                LWHITEHI,&ASD_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* unhighlight_area_label unhighlights the area label associated with       */
/* area_num by simply redisplaying the current area label                   */

unhighlight_area_label(area_num)
int area_num;

{
   if ( (area_num < 1) || (area_num > MAX_MODIFIABLE_AREAS) ) return;

   display_area_label(area_num);

   /* redraw the bypass indicators if present */
   if (get_ASD_layer_bypass(area_num-1)) display_layer_bypass(area_num-1);
   if (get_ASD_layer_bypass(area_num)) display_layer_bypass(area_num);

   return;
}

/* ------------------------------------------------------------------------ */

/* set_area_att sets the attribute to att for the area defined by base_row, */
/* base_col, num_rows and num_cols within window pointed to by wn           */

set_area_att(base_row,base_col,num_rows,num_cols,att,wn_ptr)
int base_row, base_col, num_rows, num_cols, att;
WINDOW *wn_ptr;

{
   int row_count, col_count;

   for (row_count = base_row; row_count < base_row+num_rows; row_count++)
   for (col_count = base_col; col_count < base_col+num_cols; col_count++)
```

```
   {
      mv_cs(row_count, col_count, wn_ptr);
      /* set the area attribute */
      v_natt(att,CH,wn_ptr);
   }

   return;
}

/* ------------------------------------------------------------------------ */

/* PE_on_left returns TRUE if the PE specified by layer and PE_num is on the */
/* left side of the ASD frame and FALSE otherwise                             */

PE_on_left(layer,PE_num)
int layer,PE_num;

{
   if (PE_num > 3) return(FALSE);
   else return(TRUE);
}

/* ------------------------------------------------------------------------ */

/* PE_base_row returns the ASD window row number of the base of the PE       */
/* specified by layer and PE_num                                              */
/* The PE base row is defined as the row which contains the area bar for the */
/* input path segment to the PE box                                           */

PE_base_row(layer,PE_num)
int layer, PE_num;

{
   return((layer * 4) - 1);
}

/* ------------------------------------------------------------------------ */

/* PE_base_col returns the ASD window column number of the base of the PE    */
/* specified by layer and PE_num                                              */
/* The PE base column is defined as the column which contains the left edge  */
/* of the PE box                                                              */

PE_base_col(layer,PE_num)
int layer, PE_num;

{
   int base_col;

   base_col = 19 + (PE_num * 5);

   /* even layer PE's are indented */
   if (layer%2 == 0) return(base_col + 2);
   else return(base_col);
```

```
}

/* ********************** end SAVIDISP.C source file ********************** */

/* ************************ SAVIEXTR.C source file ************************ */

/* save_current_PE_catalog writes the current PE_catalog to disk file        */
/* file_name.                                                                */

save_current_PE_catalog(file_name)
char *file_name;

{
   save_PE_catalog(file_name,current_PE_catalog_1,current_PE_catalog_2);

   return;
}

/* ------------------------------------------------------------------------- */

/* save_PE_catalog saves PE_catalog 1 and 2 to disk file file_name           */

save_PE_catalog(file_name,PE_catalog_1,PE_catalog_2)
char *file_name;
PE_SPECS PE_catalog_1[];
PE_SPECS PE_catalog_2[];

{
   FILE *stream;

   stream = fopen(file_name,"w+b");
   write_PE_catalog(stream,PE_catalog_1);
   write_PE_catalog(stream,PE_catalog_2);
   fclose(stream);

   return;

}

/* ------------------------------------------------------------------------- */

write_PE_catalog(stream,PE_catalog)
FILE *stream;
PE_SPECS PE_catalog[];

{
   int PE_num;

   for (PE_num = 0; PE_num < MAX_PE_CATALOG_ENTRIES; PE_num++)
   {
      write_PE_spec(stream,&(PE_catalog[PE_num]));
   }

   return;
```

```
}

/* ------------------------------------------------------------------------ */

write_PE_spec(stream,PE_spec_ptr)
FILE *stream;
PE_SPECS *PE_spec_ptr;

{
   int int_size, eqn_num, comp_num;

   int_size = sizeof(int);

   fwrite(PE_spec_ptr->label,1,MAX_PE_LABEL_LEN,stream);
   fwrite(PE_spec_ptr->acronym,1,MAX_PE_ACRONYM_LEN,stream);
   fwrite((char *)&(PE_spec_ptr->authorized),int_size,1,stream);
   fwrite((char *)&(PE_spec_ptr->ext_int),int_size,1,stream);
   fwrite((char *)&(PE_spec_ptr->helicopter),int_size,1,stream);

   for (comp_num = 0; comp_num < MAX_COMP_PER_PE; comp_num++)
   {
      fwrite((char *)&(PE_spec_ptr->del_comp_list[comp_num].comptype),
             int_size,1,stream);
      fwrite((char *)&(PE_spec_ptr->del_comp_list[comp_num].location),
             int_size,1,stream);
      fwrite((char *)&(PE_spec_ptr->nondet_comp_list[comp_num].comptype),
             int_size,1,stream);
      fwrite((char *)&(PE_spec_ptr->nondet_comp_list[comp_num].location),
             int_size,1,stream);
   }

   for (eqn_num = 0; eqn_num < MAX_EQNS_PER_SEGMENT; eqn_num++)
   {
      for (comp_num = 0; comp_num < MAX_COMP_PER_EQN; comp_num++)
      {
         fwrite((char *)&(PE_spec_ptr->unauthorized_del_eqns.
                entry_input_eqns[eqn_num].comptype[comp_num]),
                int_size,1,stream);
         fwrite((char *)&(PE_spec_ptr->unauthorized_del_eqns.
                entry_output_eqns[eqn_num].comptype[comp_num]),
                int_size,1,stream);
         fwrite((char *)&(PE_spec_ptr->unauthorized_del_eqns.
                exit_input_eqns[eqn_num].comptype[comp_num]),
                int_size,1,stream);
         fwrite((char *)&(PE_spec_ptr->unauthorized_del_eqns.
                exit_output_eqns[eqn_num].comptype[comp_num]),
                int_size,1,stream);
         fwrite((char *)&(PE_spec_ptr->authorized_del_eqns.
                entry_input_eqns[eqn_num].comptype[comp_num]),
                int_size,1,stream);
         fwrite((char *)&(PE_spec_ptr->authorized_del_eqns.
                entry_output_eqns[eqn_num].comptype[comp_num]),
                int_size,1,stream);
         fwrite((char *)&(PE_spec_ptr->authorized_del_eqns.
```

```
                exit_input_eqns[eqn_num].comptype[comp_num]),
                int_size,1,stream);
        fwrite((char *)&(PE_spec_ptr->authorized_del_eqns.
                exit_output_eqns[eqn_num].comptype[comp_num]),
                int_size,1,stream);
        fwrite((char *)&(PE_spec_ptr->unauthorized_nondet_eqns.
                entry_input_eqns[eqn_num].comptype[comp_num]),
                int_size,1,stream);
        fwrite((char *)&(PE_spec_ptr->unauthorized_nondet_eqns.
                entry_output_eqns[eqn_num].comptype[comp_num]),
                int_size,1,stream);
        fwrite((char *)&(PE_spec_ptr->unauthorized_nondet_eqns.
                exit_input_eqns[eqn_num].comptype[comp_num]),
                int_size,1,stream);
        fwrite((char *)&(PE_spec_ptr->unauthorized_nondet_eqns.
                exit_output_eqns[eqn_num].comptype[comp_num]),
                int_size,1,stream);
        fwrite((char *)&(PE_spec_ptr->authorized_nondet_eqns.
                entry_input_eqns[eqn_num].comptype[comp_num]),
                int_size,1,stream);
        fwrite((char *)&(PE_spec_ptr->authorized_nondet_eqns.
                entry_output_eqns[eqn_num].comptype[comp_num]),
                int_size,1,stream);
        fwrite((char *)&(PE_spec_ptr->authorized_nondet_eqns.
                exit_input_eqns[eqn_num].comptype[comp_num]),
                int_size,1,stream);
        fwrite((char *)&(PE_spec_ptr->authorized_nondet_eqns.
                exit_output_eqns[eqn_num].comptype[comp_num]),
                int_size,1,stream);
      }
   }

   return;

}

/* ------------------------------------------------------------------------ */

/* save_current_delay_catalog saves the current delay catalog to disk file   */
/* file_name.                                                                 */

save_current_delay_catalog(file_name)
char *file_name;

{
   save_delay_catalog(file_name,current_delay_catalog_1,
                              current_delay_catalog_2);

   return;
}

/* ------------------------------------------------------------------------ */

/* save_delay_catalog saves the delay catalogs 1 and 2 to disk file           */
```

```
/* file_name.                                                              */

save_delay_catalog(file_name,delay_catalog_1,delay_catalog_2)
char *file_name;
COMP_SPECS delay_catalog_1[];
COMP_SPECS delay_catalog_2[];

{
   FILE *stream;

   stream = fopen(file_name,"w+b");
   write_del_det_catalog(stream,delay_catalog_1);
   write_del_det_catalog(stream,delay_catalog_2);
   fclose(stream);

   return;

}

/* ------------------------------------------------------------------------ */

/* save_current_nondet_catalog saves the current detection catalog to disk  */
/* file file_name.                                                          */

save_current_nondet_catalog(file_name)
char *file_name;

{
   save_nondet_catalog(file_name,current_nondet_catalog_1,
                                 current_nondet_catalog_2);

   return;
}

/* ------------------------------------------------------------------------ */

/* save_nondet_catalog saves the detection catalogs 1 and 2 to disk file    */
/* file_name.                                                               */

save_nondet_catalog(file_name,nondet_catalog_1,nondet_catalog_2)
char *file_name;
COMP_SPECS nondet_catalog_1[];
COMP_SPECS nondet_catalog_2[];

{
   FILE *stream;

   stream = fopen(file_name,"w+b");
   write_del_det_catalog(stream,nondet_catalog_1);
   write_del_det_catalog(stream,nondet_catalog_2);
   fclose(stream);

   return;
```

```
}

/* ------------------------------------------------------------------------ */

write_del_det_catalog(stream,del_det_catalog)
FILE *stream;
COMP_SPECS del_det_catalog[];

{
   int comp_num;

   for (comp_num = 0; comp_num < MAX_COMP_CATALOG_ENTRIES; comp_num++)
   {
      write_comp_spec(stream,&(del_det_catalog[comp_num]));
   }

   return;
}

/* ------------------------------------------------------------------------ */

write_comp_spec(stream,comp_spec_ptr)
FILE *stream;
COMP_SPECS *comp_spec_ptr;

{
   int int_size, float_size, choice_num, threat_num;

   int_size = sizeof(int);
   float_size = sizeof(float);

   fwrite(comp_spec_ptr->label,1,MAX_COMP_LABEL_LEN,stream);

   for (choice_num = 0; choice_num < MAX_CHOICES_PER_COMPTYPE; choice_num++)
   {
      fwrite(comp_spec_ptr->comp_choice_labels[choice_num],
             1,MAX_COMP_LABEL_LEN,stream);

      for (threat_num = 0; threat_num < MAX_THREATS; threat_num++)
      {
         fwrite((char *)&(comp_spec_ptr->threat_values_table[choice_num]
                                                            [threat_num]),
                float_size,1,stream);
      }
   }

   fwrite((char *)&(comp_spec_ptr->panic_bar),int_size,1,stream);
   fwrite((char *)&(comp_spec_ptr->snm),int_size,1,stream);

   return;

}

/* ------------------------------------------------------------------------ */
```

```
/* save_current_threat_catalog saves the current threat catalog to disk file */
/* file_name.                                                                 */

save_current_threat_catalog(file_name)
char *file_name;

{
   FILE *stream;

   stream = fopen(file_name,"w+b");
   write_threat_catalog(stream,current_threat_catalog);
   fclose(stream);

   return;
}

/* -------------------------------------------------------------------------- */

write_threat_catalog(stream,threat_catalog)
FILE *stream;
THREAT_CATALOG *threat_catalog;

{
   int threat_num;

   for (threat_num = 0; threat_num < MAX_THREATS; threat_num++)
   {
      fwrite(threat_catalog->threat_labels[threat_num],
            1,MAX_THREAT_LABEL_LEN,stream);
   }

   fwrite((char *)threat_catalog->threat_transport_modes,
         sizeof(int),MAX_THREATS,stream);

   return;
}

/* ********************** end SAVIEXTR.C source file ********************** */

/* ************************ SAVIFILE.C source file ************************ */

#include <wfd.h>
#include <savidefs.h>
#include <stdio.h>
#include <direct.h>
#include <string.h>

#define FILE_NAME_WINDOW_TOP 9
#define FILE_NAME_WINDOW_COL 29
#define FILE_NAME_WINDOW_HEIGHT 5
#define FILE_NAME_WINDOW_WIDTH 36

#define DIRECTORY_WINDOW_TOP 9
```

```
#define DIRECTORY_WINDOW_COL 29
#define DIRECTORY_WINDOW_HEIGHT 5
#define DIRECTORY_WINDOW_WIDTH 36

/* initialize the default ASD file name for the save function */
char default_save_file_name[9] = {'\0',0,0,0,0,0,0,0,0};

/* initialize the default ASD file name for the retrieve function */
char default_retrieve_file_name[9] = {'\0',0,0,0,0,0,0,0,0};

/* allow for "A:\xxxxxxxx\xxxxxxxx\xxxxxxxx\" as maximum path */
char current_data_file_directory_path[31];

char current_drive_letter;

/* ------------------------------------------------------------------------ */

/* savi_file_init initializes the current data file directory to the current */
/* directory on the current default disk                                     */

savi_file_init()

{
   /* initialize the current data file directory path */

   /* if current directory is too large then simply set */
   /* the directory to be the root of the current drive */
   if (getcwd(current_data_file_directory_path,31) == NULL)
      sprintf(current_data_file_directory_path,"%c:\\",
              get_current_drive_letter());

   /* ensure that all characters are uppercase */
   strupr(current_data_file_directory_path);

   /* initialize the current disk drive letter */
   current_drive_letter = get_current_drive_letter();

   return;
}

/* ------------------------------------------------------------------------ */

/* ensure_file_existence ensures that the given file exists.  If it does not */
/* an error message is output to the screen and the code is exited.          */

ensure_file_existence(file_name)
char *file_name;

{
   char error_message[MAX_ERROR_STRING_LEN];

   if (!file_exists(file_name))
   {
      /* create the error message */
```

```
      sprintf(error_message,"%s Not Found",file_name);
      cls();
      popup_fatal_error_window(error_message);
      user_acknowledge_wait();
      remove_fatal_error_window();
      display_quit();
      exit(0);
   }

   return;

}

/* ------------------------------------------------------------------------ */

savi_file_retrieve()

{
   DFIELDPTR field_ptr;
   int file_accepted, test, retrieve_ok;
   char input_file_name[9], full_ASD_file_name[13],
        full_ASD_file_name_with_path[44];

   /* check if the current ASD has been saved */
   /* prior to retrieving a new ASD           */
   if (get_current_ASD_saved())
     retrieve_ok = TRUE;
   else
   {
     /* offer the user the chance to save the current ASD */
     /* prior to retrieving a new ASD                     */
     if (offer_file_save("Before Retrieving New ASD ?"))
        /* user chose to accept or reject save offer */
        /* ok to retrieve new ASD                    */
        retrieve_ok = TRUE;
     else
        /* user has escaped from save offer */
        /* do not retrieve new ASD          */
        retrieve_ok = FALSE;
   }

   if (!retrieve_ok) return(0);

   file_accepted = FALSE;

   /* set the input form size */
   mod_wn(FILE_NAME_WINDOW_TOP, FILE_NAME_WINDOW_COL,
          FILE_NAME_WINDOW_HEIGHT, FILE_NAME_WINDOW_WIDTH,
          STDWNP);

   /* set the input form border */
   sw_border(BDR_LNP,STDWNP);

   while (!file_accepted)
```

```
{
   /* create the input form */
   def_txt(0,0,
           " Name of ASD File to be Retrieved ",
           LCYAN);
   def_txt(1,0,
           " DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD ",
           LBLUE);
   def_txt(2,0,
           "                    .ASD           ",
           LCYAN);

   /* define the input form field */
   field_ptr = def_fld(0,0,"",def_pic('F',8),
                       F_STRING,input_file_name);

   /* set the input field location within the window */
   /* independent of the prompt                      */
   sf_location(2,11,field_ptr);

   /* require at least a single character input */
   sf_required(field_ptr);

   /* copy the default retrieve file name into the input  */
   /* file name so that the edit function can display     */
   strcpy(input_file_name,default_retrieve_file_name);

   /* accept input from user */
   if (rd_flds() == ABORT_FORM) break;

   /* check if the current drive is ready  */
   /* if not, display the error window and */
   /* abort this function                  */
   test = check_drive(current_drive_letter);
   if (test != DRIVE_OK)
   {
      popup_drive_error_window(test,current_drive_letter);
      user_acknowledge_wait();
      remove_drive_error_window();
      break;
   }

   /* load the input file name into the full ASD file name */
   /* and append the .ASD file extension onto the string   */
   strcpy(full_ASD_file_name,input_file_name);
   strcat(full_ASD_file_name,".ASD");

   /* append the full ASD file name onto the current data     */
   /* file directory path with a back slash between if needed */
   strcpy(full_ASD_file_name_with_path,current_data_file_directory_path);
   if (last_char(current_data_file_directory_path) != '\\')
      strcat(full_ASD_file_name_with_path,"\\");
   strcat(full_ASD_file_name_with_path,full_ASD_file_name);
```

```
      /* check if file to be retrieved exists */
      if (file_exists(full_ASD_file_name_with_path))
      {
         /* file exists */
         /* check if version is compatible */
         if (file_version_ok(full_ASD_file_name_with_path))
            file_accepted = TRUE;
         else
         {
            popup_incorrect_version_window(full_ASD_file_name);
            user_acknowledge_wait();
            remove_incorrect_version_window();
         }
      }
      else
      {
         popup_no_file_window(full_ASD_file_name);
         user_acknowledge_wait();
         remove_no_file_window();
      }
   }

   /* if user has input an acceptable file name then */
   /* retrieve the ASD and display it                */
   if (file_accepted)
   {
      /* set the default save and retrieve file names */
      strcpy(default_save_file_name,input_file_name);
      strcpy(default_retrieve_file_name,input_file_name);
      popup_wait_window("Retrieving ASD File");
      retrieve_current_ASD(full_ASD_file_name_with_path);
      remove_wait_window();
      display_current_ASD();
   }

   dump_key_buffer();

   /* if the file was accepted then  */
   /* return a -1 to return to main  */
   /* menu level, otherwise return 0 */
   if (file_accepted)
      return(-1);
   else
      return(0);

}

/* ------------------------------------------------------------------------ */

savi_file_save()

{
   DFIELDPTR field_ptr;
   int file_accepted, test;
```

```
char input_file_name[9], full_ASD_file_name[13],
     full_ASD_file_name_with_path[44];

file_accepted = FALSE;

/* set the input form size */
mod_wn(FILE_NAME_WINDOW_TOP, FILE_NAME_WINDOW_COL,
       FILE_NAME_WINDOW_HEIGHT, FILE_NAME_WINDOW_WIDTH,
       STDWNP);

/* set the input form border */
sw_border(BDR_LNP,STDWNP);

while (!file_accepted)
{
   /* create the input form */
   def_txt(0,0,
           "   Name of ASD File to be Saved   ",
           LCYAN);
   def_txt(1,0,
           " DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD ",
           LBLUE);
   def_txt(2,0,
           "                    .ASD          ",
           LCYAN);

   /* define the input form field */
   field_ptr = def_fld(0,0,"",def_pic('F',8),
                       F_STRING,input_file_name);

   /* set the input field location within the window */
   /* independent of the prompt                      */
   sf_location(2,11,field_ptr);

   /* require at least a single character input */
   sf_required(field_ptr);

   /* copy the default save file name into the input  */
   /* file name so that the edit function can display */
   strcpy(input_file_name,default_save_file_name);

   /* accept input from user */
   if (rd_flds() == ABORT_FORM) break;

   /* check if the current drive is ready  */
   /* if not, display the error window and */
   /* abort this function                  */
   test = check_drive(current_drive_letter);
   if (test != DRIVE_OK)
   {
      popup_drive_error_window(test,current_drive_letter);
      user_acknowledge_wait();
      remove_drive_error_window();
      break;
```

```
   }

   /* load the input file name into the full ASD file name */
   /* and append the .ASD file extension onto the string   */
   strcpy(full_ASD_file_name,input_file_name);
   strcat(full_ASD_file_name,".ASD");

   /* append the full ASD file name onto the current data     */
   /* file directory path with a back slash between if needed */
   strcpy(full_ASD_file_name_with_path,current_data_file_directory_path);
   if (last_char(current_data_file_directory_path) != '\\')
      strcat(full_ASD_file_name_with_path,"\\");
   strcat(full_ASD_file_name_with_path,full_ASD_file_name);

   /* check if new file name already exists */
   if (!file_exists(full_ASD_file_name_with_path))
   {
      /* check for sufficient free space on the */
      /* current drive to store a new ASD file  */
      /* and the analysis results if current    */
      /* if not, display the error window and   */
      /* abort this function                    */
      if (free_space_on_drive(current_drive_letter) <
          sizeof(ASD_STRUCT)+
          (get_analysis_state()?get_results_size():0))
      {
         popup_lack_disk_space_window(current_drive_letter);
         user_acknowledge_wait();
         remove_lack_disk_space_window();
         break;
      }
      file_accepted = TRUE;
   }
   else
   {
       if (replace_requested(full_ASD_file_name))
          file_accepted = TRUE;
   }
}

/* if user has input a new file name, or has requested */
/* to replace an existing file then save the ASD       */
if (file_accepted)
{
   /* set the default save and retrieve file names */
   strcpy(default_save_file_name,input_file_name);
   strcpy(default_retrieve_file_name,input_file_name);
   popup_wait_window("Saving ASD File");
   save_current_ASD(full_ASD_file_name_with_path);
   remove_wait_window();
}

dump_key_buffer();
```

```
   /* if the file was accepted then  */
   /* return a -1 to return to main  */
   /* menu level, otherwise return 0 */
   if (file_accepted)
      return(-1);
   else
      return(0);
}

/* ------------------------------------------------------------------------ */

/* file_exists returns TRUE if the given file exists within the current     */
/* disk directory. Returns FALSE otherwise.                                 */

file_exists(file_name)
char *file_name;

{
   FILE *stream;

   if ((stream = fopen(file_name, "r")) == NULL)
   {
      return(FALSE);
   }
   else
   {
      fclose(stream);
      return(TRUE);
   }
}

/* ------------------------------------------------------------------------ */

/* file_version_ok returns TRUE if the SAVI version # of the given file     */
/* is compatible with the current version and returns FALSE otherwise.      */

file_version_ok(file_name)
char *file_name;

{
   FILE *stream;
   float version;
   int version_ok;

   stream = fopen(file_name,"r+b");

   /* read the current SAVI Version Number */
   fread((char *)&version,sizeof(float),1,stream);

   fclose(stream);

   if (version != 1.0)
      return(TRUE);
   else
```

```
      return(FALSE);
}

/* ------------------------------------------------------------------------ */

/* directory_exists returns TRUE if the given directory exists and FALSE     */
/* otherwise                                                                 */

directory_exists(directory_path)
char *directory_path;

{
   FILE *stream;
   char full_file_name_with_path[44];

   /* append a temporary file name onto */
   /* the given directory path with a   */
   /* back slash to separate if needed  */
   strcpy(full_file_name_with_path,directory_path);
   if (last_char(directory_path) != '\\')
      strcat(full_file_name_with_path,"\\");
   strcat(full_file_name_with_path,"#.#");

   /* attempt to open the temporary file */
   /* in the given directory             */
   if ((stream = fopen(full_file_name_with_path,"w")) == NULL)
   {
      /* directory does not exist */
      return(FALSE);
   }
   else
   {
      /* directory is valid */
      /* close the file and */
      /* delete it          */
      fclose(stream);
      unlink(full_file_name_with_path);
      return(TRUE);
   }
}

/* ------------------------------------------------------------------------ */

/* replace_requested returns TRUE if the user requests the given file to be  */
/* replaced by the current ASD and returns FALSE otherwise.                  */

replace_requested(full_ASD_file_name)
char *full_ASD_file_name;

{
   int key, replace, done;

   replace = FALSE;
   done = FALSE;
```

```
/* display the replace request window */
popup_replace_request_window(full_ASD_file_name);

/* highlight the replace request within the window */
highlight_replace_request(replace);

/* process key press */
while (!done)
{
   /* dump the keyboard input buffer */
   dump_key_buffer();

   /* wait for a key press */
   key = ki();

   /* process key */
   /* Note: ki() returns the negative of the key scan code  */
   /* if an extended code key is pressed (ie: UP, DOWN etc) */
   /* and returns the ASCII code directly otherwise         */
   switch (-key)
   {
      /* LEFT and UP arrows set replace to NO (FALSE) */

      /* LEFT arrow */
      case K_LEFT   : ;

      /* UP arrow */
      case K_UP     : replace = FALSE;

                      highlight_replace_request(replace);

                      break;

      /* RIGHT and DOWN arrows set replace to YES (TRUE) */

      /* RIGHT arrow */
      case K_RIGHT  : ;

      /* DOWN arrow */
      case K_DN     : replace = TRUE;

                      highlight_replace_request(replace);

                      break;

      /* ENTER (Return) key */
      /* exit with current replace state */
      case -K_ENTER : done = TRUE;
                      break;

      /* Y and y keys */
      /* set replace to YES (TRUE) and exit */
      case -'Y'     : ;
```

```
            case -'y'       : replace = TRUE;
                              done = TRUE;
                              break;

            /* ESCAPE, N and n keys */
            /* set replace to NO (FALSE) and exit */
            case -'N'       : ;

            case -'n'       : ;

            case -K_ESC     : replace = FALSE;
                              done = TRUE;
                              break;

            default         : bell();
                              break;
         }
      }

      remove_replace_request_window();

      return(replace);
}

/* ------------------------------------------------------------------------ */

savi_file_directory()

{
      DFIELDPTR field_ptr;
      int path_accepted, test;
      char input_path[31], new_drive_letter;

      path_accepted = FALSE;

      /* set the input form size */
      mod_wn(DIRECTORY_WINDOW_TOP, DIRECTORY_WINDOW_COL,
             DIRECTORY_WINDOW_HEIGHT, DIRECTORY_WINDOW_WIDTH,
             STDWNP);

      /* set the input form border */
      sw_border(BDR_LNP,STDWNP);

      while (!path_accepted)
      {
         /* create the input form */
         def_txt(0,0,
                 "   Current Data File Directory    ",
                 LCYAN);
         def_txt(1,0,
                 " DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD ",
                 LBLUE);
```

```
   /* define the input form field */
   /* convert all input to upper case */
   field_ptr = def_fld(0,0,"",def_pic('!',30),
                       F_STRING,input_path);

   /* set the input field location within the window */
   /* independent of the prompt                      */
   sf_location(2,2,field_ptr);

   /* require at least a single character input */
   sf_required(field_ptr);

   /* copy the current directory path into the input  */
   /* path name so that the edit function can display */
   strcpy(input_path,current_data_file_directory_path);

   /* accept input from user */
   if (rd_flds() == ABORT_FORM) break;

   /* determine the new drive letter */
   new_drive_letter = first_non_blank_char(input_path);

   /* check if the new drive is ready      */
   /* if not, display the error window and */
   /* abort this function                  */
   test = check_drive(new_drive_letter);
   if (test != DRIVE_OK)
   {
      popup_drive_error_window(test,new_drive_letter);
      user_acknowledge_wait();
      remove_drive_error_window();
      break;
   }

   /* check if new directory exists */
   if (directory_exists(input_path))
   {
      path_accepted = TRUE;
   }
   else
   {
      popup_no_directory_window(input_path);
      user_acknowledge_wait();
      remove_no_directory_window();
   }
}

/* if user has input an acceptable path then set    */
/* the current data file directory and drive letter */
if (path_accepted)
{
   /* set the current data file directory path */
   strcpy(current_data_file_directory_path,input_path);
```

```
      /* set the current data disk drive letter */
      current_drive_letter = new_drive_letter;
   }

   dump_key_buffer();

   return(0);
}

/* ------------------------------------------------------------------------ */

savi_file_quit()

{
   dump_key_buffer();

   /* return a -1 to return to main menu level */
   return(-1);
}

/* ********************** end SAVIFILE.C source file ********************** */
#define LINT_ARGS          1

#include <dos.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <malloc.h>
#include <GRconst.h>
#include <GRports.h>
#include <GRextrn.h>
#include "keys.h"
#include "plot.h"
#include "menu.h"
#include "pathdef.h"
#include "pathglob.h"
#include "plotdecs.h"
#include "grafdefs.h"
#include "grafdecs.h"
#include "drawdecs.h"

#include "metatrim.c"

/* Define variables global to the savidraw routines...                    */
int GrafixBoard;
int *RFTArrayPtr,NumRFTs,theRFT;
int ZoomDepth,AxisType,DownOption,TicOut,zmin,zmax;
int NumXVals,BarAlign,BarPattern1,BarPattern2;
float BarWidth,MaxY2Val,MainLimit,SubLimit;
float XValue[MAX_ARRAY_SIZE];
float Y1Value[MAX_ARRAY_SIZE],Y2Value[MAX_ARRAY_SIZE];
float OldY1Val[MAX_ARRAY_SIZE],OldY2Val[MAX_ARRAY_SIZE];
char plot_title[80];
LABEL xlabels[MAX_ARRAY_SIZE+2];
```

```
LABEL y1labels[MAX_ARRAY_SIZE+2],y2labels[MAX_ARRAY_SIZE+2];
char XTitle[20],Y1Title[20],Y2Title[20];
GR_ENVIRONMENT MainScale,SubScale,MainHome,SubHome;
MENUBAR *GraphMenuBar,*ZoomMenuBar;

patRcd SaviPat = {8,8,0,1,1,1,255,0,255,0,255,0,255,0};
int MenusInitialized = FALSE;
MENUBAR MainMenu,SubMenu1;
int GraphType = VULNTYPE;
int TypeMenu,ResponseMenu,ZoomMenu,ScaleMenu,PrintMenu,QuitMenu;
int ZoomHomeMenu,ZoomOutMenu,ZoomInMenu,ZoomQuitMenu;

int graph_service(rft_array_ptr,num_rfts,rft)
int *rft_array_ptr,num_rfts,*rft;
{
   RFTArrayPtr = rft_array_ptr;
   NumRFTs = num_rfts;
   theRFT = *rft;

   SetDisplay(GrafPg0);
   if (GrafixBoard == CGA)
      SetHiResColor(LOCYAN);

   InitPlot();

   if (!MenusInitialized)
   {
      BuildMenuStructure();
      MenusInitialized = TRUE;
   }
/*temp*/
AxisType = LINTYPE;
DownOption = TRUE;
   HomeMenuBars();
   RefreshMenuBar(GraphMenuBar);

   DrawCurrentType();

   MenuService(GraphMenuBar);

   *rft = theRFT;

/*   ClearText();*/
   SetDisplay(TextPg0);
/*   i = QueryError();*/
/*   printf("QueryError=%d/%d\n",i>>7,i&127);*/
   return;
}

int TypeService()
{
   int i,XMin,XMax,YMin,YMax,XCenter,YCenter,RestoreScreen,CharHeight,
       Yposn,Line,start,Done,Choice,OldChoice,Key,OldType;
   char work[200];
```

```
   rect MessageR,RestoreR,BarR;
   image *RestoreImage;

   RestoreScreen = TRUE;

   OldType = GraphType;

   XMin = GetPortField(c_portRect+c_xmin);
   XMax = GetPortField(c_portRect+c_xmax);
   YMin = GetPortField(c_portRect+c_ymin);
   YMax = GetPortField(c_portRect+c_ymax);
   XCenter = (XMax+XMin)/2;
   YCenter = (YMax+YMin)/2;
   SetRect(&MessageR,XCenter-180,YCenter-33,
                     XCenter+180,YCenter+44);
   DupRect(&MessageR,&RestoreR);
   RestoreImage = (image *)malloc(ImageSize(&MessageR));
   if (RestoreImage == NULL)
      RestoreScreen = FALSE;
   else
      ReadImage(&RestoreR,RestoreImage);
   EraseRect(&MessageR);
/*   InsetRect(&MessageR,2,1);*/
   PenMode(Replace);
   PenPattern(1);
   FrameRect(&MessageR);
   MoveTo(XCenter-170,YCenter+31);
   LineTo(XCenter+170,YCenter+31);
   MoveTo(XCenter-170,YCenter-8);
   LineTo(XCenter+170,YCenter-8);

   /*temp - CharHeight hardwired to 9 */
   CharHeight = 9;

   TextAlign(alignCenter,alignMiddle);
   MoveTo(XCenter,YCenter+37);
   DrawString("Graph Type");
   TextAlign(alignLeft,alignBaseline);
   MoveTo(XCenter-140,YCenter+19);
   DrawString("Vulnerability");
   MoveTo(XCenter-140,YCenter+9);
   DrawString("Sensitivity");
   MoveTo(XCenter-140,YCenter-1);
   DrawString("Distribution");

   MoveTo(XCenter-160,YCenter+(MAX_GRAPH_TYPE+1-GraphType)*(CharHeight+1)-11);
   TextMode(Invert);
   DrawString("\20");

   Done = FALSE;
   Choice = GraphType;
   SetRect(&BarR,XCenter-169,
           YCenter+(MAX_GRAPH_TYPE+1-Choice)*(CharHeight+1)-13,
           XCenter+169,YCenter+(MAX_GRAPH_TYPE+1-Choice)*(CharHeight+1)-4);
```

```
InvertRect(&BarR);
TextAlign(alignCenter,alignMiddle);
TextMode(Replace);
switch (Choice)
{
   case VULNTYPE:
      MoveTo(XCenter,YCenter-16);
      DrawString("Plot P(I) & TRI for Most Vulnerable Paths");
      MoveTo(XCenter,YCenter-25);
      DrawString("       for the Current Display RFT       ");
      break;
   case SENSTYPE:
      MoveTo(XCenter,YCenter-16);
      DrawString("          Plot Worst Case P(I)           ");
      MoveTo(XCenter,YCenter-25);
      DrawString("          for Each RFT in Range          ");
      break;
   case DISTTYPE:
      MoveTo(XCenter,YCenter-16);
      DrawString("   Plot the Distribution of All P(I)s    ");
      MoveTo(XCenter,YCenter-25);
      DrawString("       for the Current Display RFT       ");
      break;
}
OldChoice = Choice;
do
{
   if (Choice != OldChoice)
   {
      SetRect(&BarR,XCenter-169,
              YCenter+(MAX_GRAPH_TYPE+1-OldChoice)*(CharHeight+1)-13,
              XCenter+169,
              YCenter+(MAX_GRAPH_TYPE+1-OldChoice)*(CharHeight+1)-4);
      InvertRect(&BarR);
      SetRect(&BarR,XCenter-169,
              YCenter+(MAX_GRAPH_TYPE+1-Choice)*(CharHeight+1)-13,
              XCenter+169,
              YCenter+(MAX_GRAPH_TYPE+1-Choice)*(CharHeight+1)-4);
      InvertRect(&BarR);
      OldChoice = Choice;
      TextAlign(alignCenter,alignMiddle);
      TextMode(Replace);
      switch (Choice)
      {
         case VULNTYPE:
            MoveTo(XCenter,YCenter-16);
            DrawString("Plot P(I) & TRI for Most Vulnerable Paths");
            MoveTo(XCenter,YCenter-25);
            DrawString("       for the Current Display RFT       ");
            break;
         case SENSTYPE:
            MoveTo(XCenter,YCenter-16);
            DrawString("          Plot Worst Case P(I)           ");
            MoveTo(XCenter,YCenter-25);
```

```
            DrawString("          for Each RFT in Range          ");
            break;
         case DISTTYPE:
            MoveTo(XCenter,YCenter-16);
            DrawString("   Plot the Distribution of All P(I)s    ");
            MoveTo(XCenter,YCenter-25);
            DrawString("       for the Current Display RFT       ");
            break;
      }
   }
   Key = GetKey();
   switch (Key)
   {
      case ESC:
         Done = TRUE;
         Choice = OldType;
         break;
      case RETURN:
         TextAlign(alignLeft,alignBaseline);
         TextMode(Invert);
         MoveTo(XCenter-160,
                YCenter+(MAX_GRAPH_TYPE+1-GraphType)*(CharHeight+1)-11);
         DrawString("\20");
         GraphType = Choice;
         MoveTo(XCenter-160,
                YCenter+(MAX_GRAPH_TYPE+1-GraphType)*(CharHeight+1)-11);
         DrawString("\20");
         TextMode(Replace);
         TextAlign(alignCenter,alignMiddle);
         Done = TRUE;
         break;
      case PAGEUP:
      case HOME:
         Choice = 1;
         break;
      case PAGEDOWN:
      case END:
         Choice = MAX_GRAPH_TYPE;
         break;
      case LEFTARROW:
      case UPARROW:
         Choice--;
         if (Choice == NOCHOICE)
            Choice = 1;
         break;
      case RIGHTARROW:
      case DOWNARROW:
         Choice++;
         if (Choice > MAX_GRAPH_TYPE)
            Choice = MAX_GRAPH_TYPE;
         break;
      case 'u':
      case 'U':
         Choice = 1;
```

```
            TextAlign(alignLeft,alignBaseline);
            TextMode(Invert);
            MoveTo(XCenter-160,
                   YCenter+(MAX_GRAPH_TYPE+1-GraphType)*(CharHeight+1)-11);
            DrawString("\20");
            GraphType = Choice;
            MoveTo(XCenter-160,
                   YCenter+(MAX_GRAPH_TYPE+1-GraphType)*(CharHeight+1)-11);
            DrawString("\20");
            TextMode(Replace);
            TextAlign(alignCenter,alignMiddle);
            Done = TRUE;
            break;
         case 's':
         case 'S':
            Choice = 2;
            TextAlign(alignLeft,alignBaseline);
            TextMode(Invert);
            MoveTo(XCenter-160,
                   YCenter+(MAX_GRAPH_TYPE+1-GraphType)*(CharHeight+1)-11);
            DrawString("\20");
            GraphType = Choice;
            MoveTo(XCenter-160,
                   YCenter+(MAX_GRAPH_TYPE+1-GraphType)*(CharHeight+1)-11);
            DrawString("\20");
            TextMode(Replace);
            TextAlign(alignCenter,alignMiddle);
            Done = TRUE;
            break;
         case 'd':
         case 'D':
            Choice = 3;
            TextAlign(alignLeft,alignBaseline);
            TextMode(Invert);
            MoveTo(XCenter-160,
                   YCenter+(MAX_GRAPH_TYPE+1-GraphType)*(CharHeight+1)-11);
            DrawString("\20");
            GraphType = Choice;
            MoveTo(XCenter-160,
                   YCenter+(MAX_GRAPH_TYPE+1-GraphType)*(CharHeight+1)-11);
            DrawString("\20");
            TextMode(Replace);
            TextAlign(alignCenter,alignMiddle);
            Done = TRUE;
            break;
         default:
            bell();
            break;
      }
   } while (!Done);

   if (GraphType == OldType)
   {
      if (RestoreScreen)
```

```
      {
         WriteImage(&RestoreR,RestoreImage);
         free(RestoreImage);
      }
      else
         DrawGraph();
   }
   else
   {
      DrawCurrentType();
      if (RestoreScreen)
         free(RestoreImage);
   }

   return(0);
}

int DrawCurrentType()
{
   switch (GraphType)
   {
      case VULNTYPE:
         DrawVulnType();
         break;
      case SENSTYPE:
         DrawSensType();
         break;
      default:
         DrawDistType();
         break;
   }
   return;
}

int DrawVulnType()
{
   int i,j,Y2Val;
   float min,max,inc;

   GraphType = VULNTYPE;
   InitPlot();
   SetMaxCount(10);
   SetBorderArea(18,18,11,23);

/*temp   HomeMenuBars();*/

   BarAlign = RIGHT;
   BarPattern1 = PATTERN1;
   BarPattern2 = PATTERN2;

   MainLimit = .0005;
   SubLimit = .05;

/*temp - change to get 'max' numxvals until feedback algorithm is decided on*/
```

```
   NumXVals = get_displayable_paths(theRFT);
   if (NumXVals > MAX_PATHS_RETURNED)
      NumXVals = MAX_PATHS_RETURNED;

   /* get max time value... */
   MaxY2Val = 9;
   for (j=0; j<NumRFTs; j++)
      for (i=0; i<NumXVals; i++)
      {
         /* note, TRI is always 0 if cdp is on the offsite node (bp=0) */
         Y2Val = MaxOf(get_path_ptr(j,i)->cum_sum-RFTArrayPtr[j],0);
         MaxY2Val = MaxOf(MaxY2Val,Y2Val);
         if (get_path_ptr(j,i)->break_point == 0)
            Y2Val = 0;
      }
   MaxY2Val++;

   BarWidth = 0.4;

   AxisType = LINTYPE;

   SetXStuff(0.0,((float)NumXVals+1.0),1.0,(NumXVals+1));
   SetYStuff(0.0,1.0,0.1,10);
   CopyEnviron(&MainScale);
   CopyEnviron(&MainHome);
   SetLabels(FL1_LABELS,y1labels,TRUE,TRUE,0.0,10,0.1);
   SetCount(0.0,MaxY2Val,LINTYPE,10,&min,&max,&inc);
   SetYStuff(min,max,inc,10);
   CopyEnviron(&SubScale);
   CopyEnviron(&SubHome);
   SetLabels(INT_LABELS,y2labels,TRUE,TRUE,0.0,10,inc);
   SetLabels(INT_LABELS,xlabels,FALSE,FALSE,0.0,NumXVals+1,1.0);
   sprintf(XTitle,"Path Number");
   sprintf(Y1Title,"P(I)");
   sprintf(Y2Title,"TRI");
   ZoomHome();
   TicOut = 0;

   BuildVuln();

   HomeMenuBars();

   DrawGraph();

   return;
}

int DrawSensType()
{
   int i,j;
   float min,max,inc;

   GraphType = SENSTYPE;
   InitPlot();
```

```
   SetMaxCount(10);
   SetBorderArea(18,5,11,23);

/*temp   HomeMenuBars();*/

   BarAlign = MIDDLE;
   BarPattern1 = PATTERN1;

   MainLimit = .0005;
   SubLimit = 0;

   MaxY2Val = 10;

   NumXVals = NumRFTs;

   BarWidth = 0.8;

   AxisType = LINTYPE;

   SetXStuff(0.0,((float)NumXVals+1.0),1.0,(NumXVals+1));
   SetYStuff(0.0,1.0,0.1,10);
   CopyEnviron(&MainScale);
   CopyEnviron(&MainHome);
   SetLabels(FL1_LABELS,y1labels,TRUE,TRUE,0.0,10,0.1);
   CopyEnviron(&SubScale);
   CopyEnviron(&SubHome);
   sprintf(xlabels[0],"\0");
   for (i=1; i<NumRFTs+1; i++)
      sprintf(xlabels[i],"%d",RFTArrayPtr[i-1]);
   sprintf(xlabels[NumRFTs+1],"\0");
   sprintf(XTitle,"RFT (seconds)");
   sprintf(Y1Title,"P(I)");
   ZoomHome();
   TicOut = 0;

   BuildSens();

   HomeMenuBars();

   DrawGraph();

   return;
}

int DrawDistType()
{
   int i,j;
   float min,max,inc;

   GraphType = DISTTYPE;
   InitPlot();
   SetMaxCount(10);
   SetBorderArea(18,5,11,23);
```

```
/*temp   HomeMenuBars();*/

   BarAlign = LEFT;
   BarPattern1 = PATTERN2;

   MainLimit = .5;
   SubLimit = 0;

   MaxY2Val = 10;

   NumXVals = FREQ_BINS;

   BarWidth = 1.0/((float)FREQ_BINS);

   AxisType = LINTYPE;

   SetXStuff(0.0,1.0,BarWidth,FREQ_BINS);
   SetYStuff(0.0,100.0,10.0,10);
   CopyEnviron(&MainScale);
   CopyEnviron(&MainHome);
   SetLabels(FL1_LABELS,xlabels,TRUE,TRUE,0.0,FREQ_BINS,BarWidth);

   /* blank every other label */
   for (i=1; i<FREQ_BINS; i=i+2)
      sprintf(xlabels[i]," ");
   /* HARDWIRED FOR FREQ_BINS = 20! */

   SetLabels(INT_LABELS,y1labels,TRUE,TRUE,0.0,10,10.0);
   CopyEnviron(&SubScale);
   CopyEnviron(&SubHome);
   sprintf(XTitle,"P(I)");
   sprintf(Y1Title,"      %%");
   ZoomHome();
   TicOut = 0;

   BuildDist();

   HomeMenuBars();

   DrawGraph();

   return;
}

int BuildGraph()
{
   switch (GraphType)
   {
      case VULNTYPE:
         BuildVuln();
         break;
      case SENSTYPE:
         BuildSens();
         break;
```

```
      case DISTTYPE:
         BuildDist();
         break;
   }
   return;
}

int BuildVuln()
{
   int i;
   char sec_string[20];

   /* generate plot data */
   for (i=0; i<NumXVals; i++)
   {
      XValue[i] = i+1;
      Y1Value[i] = 1.0 - get_path_ptr(theRFT,i)->cum_prod;
      Y2Value[i] = MaxOf(get_path_ptr(theRFT,i)->cum_sum-
                         RFTArrayPtr[theRFT],0);
   }

   if (RFTArrayPtr[theRFT] == 1)
      sprintf(sec_string,"1 second");
   else
      sprintf(sec_string,"%d seconds",RFTArrayPtr[theRFT]);
   if (NumXVals == 1)
      sprintf(plot_title,
             "Vulnerability of Worst Path/  RFT = %s  ",
              sec_string);
   else
      sprintf(plot_title,
              "Vulnerability of %d Worst Paths/  RFT = %s  ",
              NumXVals,sec_string);

   return;
}

int BuildSens()
{
   int i;
   char sec_string[20],temp_string[40];

   /* generate plot data */
   for (i=0; i<NumXVals; i++)
   {
      XValue[i] = i+1;
      Y1Value[i] = 1.0 - get_path_ptr(i,0)->cum_prod;
   }

   if (NumRFTs == 1)
   {
      if (RFTArrayPtr[theRFT] == 1)
         sprintf(sec_string,"1 second");
      else
```

```
         sprintf(sec_string,"%d seconds",RFTArrayPtr[0]);
      sprintf(temp_string,"1 RFT: %s",sec_string);
   }
   else
   {
      if (RFTArrayPtr[NumRFTs-1] == 1)
         sprintf(sec_string,"1 second");
      else
         sprintf(sec_string,"%d seconds",RFTArrayPtr[NumRFTs-1]);
      sprintf(temp_string,"%d RFTs from %d to %s",NumRFTs,RFTArrayPtr[0],
            sec_string);
   }
   sprintf(plot_title,"Sensitivity of Worst Case P(I) to RFT/  %s  ",
          temp_string);

   return;
}

int BuildDist()
{
   int i;
   float maxpaths;
   char sec_string[20];

   maxpaths = get_valid_paths(theRFT);
   /* generate plot data */
   for (i=0; i<NumXVals; i++)
   {
      XValue[i] = i*BarWidth;
      Y1Value[i] = 100 * get_frequency_ptr(theRFT)[(FREQ_BINS-1)-i]/maxpaths;
   }

   if (RFTArrayPtr[theRFT] == 1)
      sprintf(sec_string,"1 second");
   else
      sprintf(sec_string,"%d seconds",RFTArrayPtr[theRFT]);
   sprintf(plot_title,
          "Distribution of P(I)/      %.0f Paths for RFT = %s      ",
          maxpaths,sec_string);

   return;
}

int ResponseService()
{
   if (SetRFT())
   {
      BuildGraph();
      RedrawGraphAndTitle();
   }
   return(0);
}

int ScrollUpService()
```

```
{
   theRFT++;
   if (theRFT >= NumRFTs)
      theRFT = 0;
   BuildGraph();
   RedrawGraphAndTitle();
   return(0);
}

int ScrollDownService()
{
   theRFT--;
   if (theRFT < 0)
      theRFT = NumRFTs-1;
   BuildGraph();
   RedrawGraphAndTitle();
   return(0);
}

int ZoomHomeService()
{
   ZoomHome();
   DrawGraph();
   return(0);
}

int ZoomHome()
{
   int popped,label1,label2;

   ZoomDepth = 0;
   DownOption = TRUE;
   popped=FALSE;
   while (PopTwo(&MainScale,&SubScale) == OK)popped=TRUE;
   if (popped)
   {
      switch (GraphType)
      {
         case VULNTYPE:
            label1 = FL2_LABELS;
            label2 = INT_LABELS;
            break;
         case SENSTYPE:
            label1 = FL2_LABELS;
            break;
         case DISTTYPE:
            label1 = INT_LABELS;
            break;
      }
      if (AxisType == LINTYPE)
         LabelTwo(label1,label2,y1labels,y2labels,&MainScale,&SubScale);
      else
         LabelTwo(LOG_LABELS,LOG_LABELS,y1labels,y2labels,
                  &MainScale,&SubScale);
```

```
   )
   DeactivateMenu(ZoomMenuBar,ZoomHomeMenu);
   DeactivateMenu(ZoomMenuBar,ZoomOutMenu);
   ActivateMenu(ZoomMenuBar,ZoomInMenu);
   return;
}

int ZoomOutService()
{
   int label1,label2;

   if (ZoomDepth>0)
   {
      PopTwo(&MainScale,&SubScale);
      ZoomDepth--;
      switch (GraphType)
      {
         case VULNTYPE:
            label1 = FL4_LABELS;
            label2 = FL2_LABELS;
            break;
         case SENSTYPE:
            label1 = FL4_LABELS;
            break;
         case DISTTYPE:
            label1 = FL1_LABELS;
            break;
      }
      LabelTwo(label1,label2,y1labels,y2labels,&MainScale,&SubScale);
   }
   DownOption = TRUE;
   ActivateMenu(ZoomMenuBar,ZoomInMenu);
   if (ZoomDepth == 0)
   {
      switch (GraphType)
      {
         case VULNTYPE:
            label1 = FL2_LABELS;
            label2 = INT_LABELS;
            break;
         case SENSTYPE:
            label1 = FL2_LABELS;
            break;
         case DISTTYPE:
            label1 = INT_LABELS;
            break;
      }
      LabelTwo(label1,label2,y1labels,y2labels,
               &MainScale,&SubScale);
   }
   if (AxisType == LOGTYPE)
      LabelTwo(LOG_LABELS,LOG_LABELS,y1labels,y2labels,
               &MainScale,&SubScale);
   if (ZoomDepth == 0)
```

```
   {
      DeactivateMenu(ZoomMenuBar,ZoomHomeMenu);
      DeactivateMenu(ZoomMenuBar,ZoomOutMenu);
   }
   DrawGraph();
   return(0);
}

int ZoomInService()
{
   float min,max,inc;
   int label1,label2;
   int count;

   if (DownOption)
   {
      PushTwo(&MainScale,&SubScale);
      SetEnviron(&MainScale);
      if (zoom())
      {
         if (AxisType == LOGTYPE)
         {
            SetEnviron(&MainScale);
            SetYStuff((GetYMin()+zmin),(GetYMin()+zmax),
                      1.0,(zmax-zmin));
            CopyEnviron(&MainScale);
            SetEnviron(&SubScale);
            SetYStuff((GetYMin()+zmin),(GetYMin()+zmax),
                      1.0,(zmax-zmin));
            CopyEnviron(&SubScale);
         }
         else
         {
            SetEnviron(&MainScale);
            count = SetScale(GetYMin()+GetYInc()*zmin,
                             GetYMin()+GetYInc()*zmax,
                             LINTYPE,&min,&max,&inc);
            SetYStuff(min,max,inc,count);
            CopyEnviron(&MainScale);
            SetEnviron(&SubScale);
            SetCount(GetYMin()+GetYInc()*zmin,
                     GetYMin()+GetYInc()*zmax,
                     LINTYPE,count,&min,&max,&inc);
            SetYStuff(min,max,inc,count);
            CopyEnviron(&SubScale);
         }
         switch (GraphType)
         {
            case VULNTYPE:
               label1 = FL4_LABELS;
               label2 = FL2_LABELS;
               break;
            case SENSTYPE:
               label1 = FL4_LABELS;
```

```
            break;
         case DISTTYPE:
            label1 = FL1_LABELS;
            break;
      }
      if (AxisType == LINTYPE)
         LabelTwo(label1,label2,y1labels,y2labels,&MainScale,&SubScale);
      else
         LabelTwo(LOG_LABELS,LOG_LABELS,y1labels,y2labels,
                  &MainScale,&SubScale);
      ZoomDepth++;
      if ((ZoomDepth<5)&&(MainScale.YInc>MainLimit)&&
          (SubScale.YInc>SubLimit)&&(MainScale.YCount>1))
         DownOption = TRUE;
      else
         DownOption = FALSE;
      DrawGraph();
   }
   else
      PopTwo(&MainScale,&SubScale);
}
if (ZoomDepth > 0)
{
   ActivateMenu(ZoomMenuBar,ZoomHomeMenu);
   ActivateMenu(ZoomMenuBar,ZoomOutMenu);
}
if (!DownOption)
   DeactivateMenu(ZoomMenuBar,ZoomInMenu);
return(0);
}

int ZoomQuitService()
{
   return(1);
}

int ScaleService()
{
   int label1,label2;
   float max,MinCycle,MaxCycle;
   int i,XMin,XMax,YMin,YMax,XCenter,YCenter,RestoreScreen,CharHeight,
       Yposn,Line,start,Done,Choice,OldChoice,Key,OldType;
   char work[200];
   rect MessageR,RestoreR,BarR;
   image *RestoreImage;

   RestoreScreen = TRUE;

   OldType = AxisType;

   XMin = GetPortField(c_portRect+c_xmin);
   XMax = GetPortField(c_portRect+c_xmax);
   YMin = GetPortField(c_portRect+c_ymin);
   YMax = GetPortField(c_portRect+c_ymax);
```

```
   XCenter = (XMax+XMin)/2;
   YCenter = (YMax+YMin)/2;
   SetRect(&MessageR,XCenter-80,YCenter-17,
                     XCenter+80,YCenter+22);
   DupRect(&MessageR,&RestoreR);
   RestoreImage = (image *)malloc(ImageSize(&MessageR));
   if (RestoreImage == NULL)
      RestoreScreen = FALSE;
   else
      ReadImage(&RestoreR,RestoreImage);
   EraseRect(&MessageR);
/*   InsetRect(&MessageR,2,1);*/
   PenMode(Replace);
   PenPattern(1);
   FrameRect(&MessageR);
   MoveTo(XCenter-70,YCenter+9);
   LineTo(XCenter+70,YCenter+9);

   /*temp - CharHeight hardwired to 9 */
   CharHeight = 9;

   TextAlign(alignCenter,alignMiddle);
   MoveTo(XCenter,YCenter+15);
   DrawString("Graph Scale");
   TextAlign(alignLeft,alignBaseline);
   MoveTo(XCenter-40,YCenter-2);
   DrawString("Linear");
   MoveTo(XCenter-40,YCenter-12);
   DrawString("Logarithmic");

   MoveTo(XCenter-60,YCenter-AxisType*(CharHeight+1)-2);
   TextMode(Invert);
   DrawString("\20");

   Done = FALSE;
   Choice = AxisType;
   SetRect(&BarR,XCenter-69,YCenter-Choice*(CharHeight+1)-4,
           XCenter+69,YCenter-Choice*(CharHeight+1)+5);
   InvertRect(&BarR);
   OldChoice = Choice;
   do
   {
      if (Choice != OldChoice)
      {
         SetRect(&BarR,XCenter-69,YCenter-OldChoice*(CharHeight+1)-4,
                 XCenter+69,YCenter-OldChoice*(CharHeight+1)+5);
         InvertRect(&BarR);
         SetRect(&BarR,XCenter-69,YCenter-Choice*(CharHeight+1)-4,
                 XCenter+69,YCenter-Choice*(CharHeight+1)+5);
         InvertRect(&BarR);
         OldChoice = Choice;
      }
      Key = GetKey();
      switch (Key)
```

```
    {
        case ESC:
            Done = TRUE;
            Choice = OldType;
            break;
        case RETURN:
            MoveTo(XCenter-60,YCenter-AxisType*(CharHeight+1)-2);
            DrawString("\20");
            MoveTo(XCenter-60,YCenter-Choice*(CharHeight+1)-2);
            DrawString("\20");
            Done = TRUE;
            break;
        case PAGEUP:
        case HOME:
            Choice = 0;
            break;
        case PAGEDOWN:
        case END:
            Choice = 1;
            break;
        case LEFTARROW:
        case UPARROW:
            Choice--;
            if (Choice < 0)
                Choice = 0;
            break;
        case RIGHTARROW:
        case DOWNARROW:
            Choice++;
            if (Choice > 1)
                Choice = 1;
            break;
        default:
            bell();
            break;
    }
} while (!Done);

if (Choice != OldType)
{
    if (ZoomDepth != 0)
        ZoomHome();

    if (AxisType == LINTYPE)
    {
        AxisType = LOGTYPE;
        TicOut = 2;
        if (GraphType == DISTTYPE)
        {
            MinCycle = -2.0;
            MaxCycle = 2.0;
        }
        else
        {
```

```
            MinCycle = -4.0;
            MaxCycle = 0.0;
         }
         SetLog(&MainHome,&MainScale,MinCycle,MaxCycle,1.0,4);
         max = floor(log10((double)MaxY2Val) + 1.0);
         SetLog(&SubHome,&SubScale,max-4.0,max,1.0,4);
         LabelTwo(LOG_LABELS,LOG_LABELS,y1labels,y2labels,
                  &MainScale,&SubScale);
      }
      else
      {
         AxisType = LINTYPE;
         TicOut = 0;
         SetLin(&MainHome,&MainScale);
         SetLin(&SubHome,&SubScale);
         switch (GraphType)
         {
            case VULNTYPE:
               label1 = FL2_LABELS;
               label2 = INT_LABELS;
               break;
            case SENSTYPE:
               label1 = FL2_LABELS;
               break;
            case DISTTYPE:
               label1 = INT_LABELS;
               break;
         }
         LabelTwo(label1,label2,y1labels,y2labels,&MainScale,&SubScale);
      }
      DrawGraph();
      if (RestoreImage != NULL)
         free(RestoreImage);
   }
   else
      if (RestoreImage != NULL)
      {
         WriteImage(&RestoreR,RestoreImage);
         free(RestoreImage);
      }
      else
         DrawGraph();

   return(0);
}

int PrintService()
{
   PrintScreen(FALSE);
   DumpKeyBuffer();
   return(0);
}

int PrintLogoService()
```

```
{
   PrintScreen(TRUE);
   DumpKeyBuffer();
   return(0);
}

int QuitService()
{
   return(1);
}

int Legend()
{
   int x,y,Axis;
   rect LegendR;

   Axis = GetYType();
   SetAxis(YAXIS,LINTYPE);

   x = ScaledX((GetXMin()-20*(GetXMax()-GetXMin())/100.0))-8;
   y = ScaledY((GetYMin()+0.4*(GetYMax()-GetYMin())))-1;
   SetRect(&LegendR,x,y-16,x+16,y);
   if (GrafixBoard == EGA)
      PenColor(HIBLUE);
   FillRect(&LegendR,PATTERN1);
   FrameRect(&LegendR);

   x = ScaledX((GetXMax()+20*(GetXMax()-GetXMin())/100.0))-8;
   y = ScaledY((GetYMin()+0.4*(GetYMax()-GetYMin())))-1;
   SetRect(&LegendR,x,y-16,x+16,y);
   if (GrafixBoard == EGA)
      PenColor(HIGREEN);
   FillRect(&LegendR,PATTERN2);
   FrameRect(&LegendR);

   SetAxis(YAXIS,Axis);
   return;
}

/* ************************ SAVIINIT.C source file ************************** */

#include <wfd.h>
#include <savidefs.h>

/* ------------------------------------------------------------------------ */

savi_init()

{
   /* initialize the windowing package */
   init_wfc();
   init_wfd();
   init_display();
   init_windows();
```

```
display_intro();

/* initialize the menus */
savi_menu_init();
analyze_menu_init();

/* initialize the file data */
savi_file_init();

/* set default parameters */
set_num_copies(1);
set_current_output("stdprn");

/* ensure that all support files exist in current directory */
ensure_file_existence(PE_CATALOG_FILE_NAME);
ensure_file_existence(DELAY_CATALOG_FILE_NAME);
ensure_file_existence(NONDET_CATALOG_FILE_NAME);
ensure_file_existence(THREAT_CATALOG_FILE_NAME);
ensure_file_existence(STARTUP_ASD_FILE_NAME);
if (!file_version_ok(STARTUP_ASD_FILE_NAME))
{
   cls();
   popup_incorrect_version_window(STARTUP_ASD_FILE_NAME);
   user_acknowledge_wait();
   remove_incorrect_version_window();
   display_quit();
   exit(0);
}
ensure_file_existence(GRAPHIC_FONT_NAME);

/* allocate memory for all support catalogs and current ASD */
allocate_PE_cat();
allocate_delay_cat();
allocate_nondet_cat();
allocate_threat_cat();
allocate_ASD();

/* load all support files */
display_now_loading();

retrieve_current_PE_catalog(PE_CATALOG_FILE_NAME);
retrieve_current_delay_catalog(DELAY_CATALOG_FILE_NAME);
retrieve_current_nondet_catalog(NONDET_CATALOG_FILE_NAME);
retrieve_current_threat_catalog(THREAT_CATALOG_FILE_NAME);
retrieve_current_ASD(STARTUP_ASD_FILE_NAME);

/* initialize metawindows */
/* Note: graphic font file must be present */
/* in current directory                    */
init_meta();

remove_intro();
complete_tone();
display_current_ASD();
```

```
   return;
}

/* ********************** end SAVIINIT.C source file ********************** */

/* ************************ SAVIMENU.C source file ************************ */

#include <wfd.h>
#include <savidefs.h>

DFORMPTR top_menu;
DFORMPTR file_sub_menu;
DFORMPTR modify_sub_menu;
DFORMPTR area_sub_menu;
DFORMPTR element_sub_menu;
DFORMPTR define_sub_menu;
DFORMPTR component_sub_menu;
DFORMPTR distance_sub_menu;
DFORMPTR threat_sub_menu;
DFORMPTR print_sub_menu;
DFORMPTR quit_sub_menu;

int savi_done;

/* ------------------------------------------------------------------------ */

/* savi_menu handles the top menu functions allowing the user to select any  */
/* function within the menu tree. An attempt to quit from the top of the     */
/* menu rings the bell                                                       */

savi_menu()

{
   /* dump the keyboard buffer before  */
   /* beginning menu handling function */
   dump_key_buffer();

   savi_done = FALSE;

   while (!savi_done)
   {
      rd_menu(top_menu);

      /* if the user attempts to quit from the top */
      /* of the menu, ring the bell                */
      if (!savi_done) bell();
   }

   return;
}

/* ------------------------------------------------------------------------ */
```

```
savi_menu_init()

{
   cl_att = LNORMAL;

   /* Define the top level menu form */
   top_menu = def_frm(0, 0, 1, 80, LFIELDI, BDR_0P);

   /* Define the sub-menu forms */
   file_sub_menu = def_frm(0, 0, 1, 80, LFIELDI, BDR_0P);
   modify_sub_menu = def_frm(0, 0, 1, 80, LFIELDI, BDR_0P);
   area_sub_menu = def_frm(0, 0, 1, 80, LFIELDI, BDR_0P);
   element_sub_menu = def_frm(0, 0, 1, 80, LFIELDI, BDR_0P);
   threat_sub_menu = def_frm(0, 0, 1, 80, LFIELDI, BDR_0P);
   define_sub_menu = def_frm(0, 0, 1, 80, LFIELDI, BDR_0P);
   component_sub_menu = def_frm(0, 0, 1, 80, LFIELDI, BDR_0P);
   distance_sub_menu = def_frm(0, 0, 1, 80, LFIELDI, BDR_0P);
   print_sub_menu = def_frm(0, 0, 1, 80, LFIELDI, BDR_0P);
   quit_sub_menu = def_frm(0, 0, 1, 80, LFIELDI, BDR_0P);

   /* Define the functions and sub-menus for the top level menu */
   def_mfld(0, 1, "File", "Retrieve      Save      Directory      Quit",
            file_sub_menu, NULLFP, top_menu);
   def_mfld(0, 11, "Modify","Title      Area      Element      Quit",
            modify_sub_menu, NULLFP, top_menu);
   def_mfld(0, 23, "Threat", "Attributes      Objective      \
Tactics      Quit",
            threat_sub_menu, NULLFP, top_menu);
   def_mfld(0, 35, "Define", "Component      Distance      \
Response      Quit",
            define_sub_menu, NULLFP, top_menu);
   def_mfld(0, 47, "Analyze", "Results      Print      Quit",
            NULLP, savi_analyze, top_menu);
   def_mfld(0, 60, "Print", "Diagram      Tables      Quit",
            print_sub_menu, NULLFP, top_menu);
   def_mfld(0, 71, "Quit", "Quit SAVI", quit_sub_menu,
            NULLFP, top_menu);

   /* Define the functions for the file sub-menu */
   def_mfld(0, 1, "Retrieve", "Replace the Current ASD with an ASD \
from a Specified Disk File", NULLP, savi_file_retrieve, file_sub_menu);
   def_mfld(0, 15, "Save", "Save the Current ASD to a Specified \
Disk File", NULLP, savi_file_save,file_sub_menu);
   def_mfld(0, 25, "Directory", "Set the Data File Directory",
            NULLP, savi_file_directory,file_sub_menu);
   def_mfld(0, 40, "Quit", "Return to the Main Menu",
            NULLP, savi_file_quit, file_sub_menu);

   /* Define the functions for the modify sub-menu */
   def_mfld(0, 1, "Title", "Modify the Title of the Current ASD",
            NULLP, savi_modify_title, modify_sub_menu);
   def_mfld(0, 12, "Area", "Labels      Traversal      Quit",
            area_sub_menu, NULLFP, modify_sub_menu);
   def_mfld(0, 22, "Element", "Presence      Jumps      Types      Quit",
```

```
            element_sub_menu, NULLFP, modify_sub_menu);
   def_mfld(0, 35, "Quit", "Return to the Main Menu",
            NULLP, savi_modify_quit, modify_sub_menu);

   /* Define the functions for the area sub-menu */
   def_mfld(0, 1, "Labels", "Modify the Area Labels",
            NULLP, savi_modify_area_labels, area_sub_menu);
   def_mfld(0, 13, "Traversal", "Modify the Area Traversal Settings",
            NULLP, savi_modify_area_traversal, area_sub_menu);
   def_mfld(0, 28, "Quit", "Return to the Modify Menu",
            NULLP, savi_modify_area_quit, area_sub_menu);

   /* Define the functions for the element sub-menu */
   def_mfld(0, 1, "Presence", "Modify the Presence of the Protection \
Elements",
            NULLP, savi_modify_element_presence, element_sub_menu);
   def_mfld(0, 15, "Jumps", "Modify the Output Areas of the Protection \
Elements",
            NULLP, savi_modify_element_jumps, element_sub_menu);
   def_mfld(0, 26, "Types", "Modify the Types of the Protection \
Elements",
            NULLP, savi_modify_element_types, element_sub_menu);
   def_mfld(0, 37, "Quit", "Return to the Modify Menu",
            NULLP, savi_modify_element_quit, element_sub_menu);

   /* Define the functions for the define sub-menu */
   def_mfld(0, 1, "Component", "Settings      Copy      Quit",
            component_sub_menu, NULLFP, define_sub_menu);
   def_mfld(0, 16, "Distance", "Areas      Elements      Quit",
            distance_sub_menu, NULLFP, define_sub_menu);
   def_mfld(0, 30, "Response", "Define the Response Force Time Range",
            NULLP, savi_define_response, define_sub_menu);
   def_mfld(0, 44, "Quit", "Return to the Main Menu",
            NULLP, savi_define_quit, define_sub_menu);

   /* Define the functions for the component sub-menu */
   def_mfld(0, 1, "Settings", "Define the Protection Elements' \
Component Settings",
            NULLP, savi_define_component_settings, component_sub_menu);
   def_mfld(0, 15, "Copy", "Copy Protection Element \
Component Settings",
            NULLP, savi_define_component_copy, component_sub_menu);
   def_mfld(0, 25, "Quit", "Return to the Define Menu",
            NULLP, savi_define_component_quit, component_sub_menu);

   /* Define the functions for the distance sub-menu */
   def_mfld(0, 1, "Areas", "Define the Distance Across Each Area",
            NULLP, savi_define_distance_areas, distance_sub_menu);
   def_mfld(0, 12, "Elements", "Define the Distance Across Each \
Protection Element",
            NULLP, savi_define_distance_elements, distance_sub_menu);
   def_mfld(0, 26, "Quit", "Return to the Define Menu",
            NULLP, savi_define_distance_quit, distance_sub_menu);
```

```
   /* Define the functions for the threat sub-menu */
   def_mfld(0, 1, "Attributes", "Select the Threat Attributes",
            NULLP, savi_threat_attributes, threat_sub_menu);
   def_mfld(0, 17, "Objective", "Select the Threat Objective",
            NULLP, savi_threat_objective, threat_sub_menu);
   def_mfld(0, 32, "Tactics", "Select the Threat Tactics",
            NULLP, savi_threat_strategy, threat_sub_menu);
   def_mfld(0, 45, "Quit", "Return to the Main Menu",
            NULLP, savi_threat_quit, threat_sub_menu);

   /* Define the functions for the print sub-menu */
   def_mfld(0, 1, "Diagram", "Print the Current Adversary Sequence Diagram",
            NULLP, savi_print_diagram, print_sub_menu);
   def_mfld(0, 14, "Tables", "Print the Current ASD Settings in Tabular Form",
            NULLP, savi_print_tables, print_sub_menu);
   def_mfld(0, 26, "Quit", "Return to the Main Menu",
            NULLP, savi_print_quit, print_sub_menu);

   /* Define the functions for the quit sub-menu */
   def_mfld(0, 1, "No", "Do Not Quit SAVI",
            NULLP, savi_quit_no, quit_sub_menu);
   def_mfld(0, 9, "Yes", "Quit SAVI",
            NULLP, savi_quit_yes, quit_sub_menu);

     return;
}

/* ------------------------------------------------------------------------ */

set_savi_done()

{
   savi_done = TRUE;

   return;
}

/* ********************** end SAVIMENU.C source file ********************** */

/* ************************ SAVIMISC.C source file ************************ */

#include <wfd.h>
#include <conio.h>
#include <savidefs.h>
#include <dos.h>
#include <time.h>

/* ------------------------------------------------------------------------ */

/* get_user_choice allows user to move a highlight bar vertically between    */
/* num_row rows beginning at min_row of the window that wn_ptr is pointing   */
/* to starting at the given choice number (0 to num_rows-1).                 */
/* When the user presses the return key the choice number of the currently   */
/* highlighted row is returned.  (ie: the choice number of the first row     */
```

```
/* (min_row) is zero (0))                                                    */
/* The return_code value is set according to the type of return.  If the     */
/* user presses the return key to exit, the return_code is set to zero (0)   */
/* If the user presses Escape, Home, End, PgUp or PgDn the negative of the   */
/* scan code is returned.  If the user presses the Left or Up arrow keys     */
/* when the highlight bar is at the top of the range (min_row) then the      */
/* negative of the Up arrow key scan code is returned.  Similarly if the     */
/* user presses the Right or Down arrow keys when the highlight bar is at    */
/* the bottom of the range the negative of the Down arrow key scan           */
/* code is returned. If the user presses any alphabetic key, the ASCII code  */
/* for that key in uppercase is set in the return_code.  If the user presses */
/* ?, # or SPACE-BAR the corresponding ASCII code is set in the return_code. */
/* If any other key is pressed the bell is sounded.                          */

get_user_choice(min_row, num_rows, start_choice, wn_ptr, return_code_ptr)
int min_row, num_rows, start_choice;
WINDOW *wn_ptr;
int *return_code_ptr;

{
   int key, temp_key, current_row, max_row, start_row, done;

   done = FALSE;
   *return_code_ptr = 0;
   max_row = min_row + num_rows - 1;
   start_row = min_row + start_choice;

   /* highlight the start_row within the given window */
   current_row = start_row;
   highlight_row(current_row, wn_ptr);

   /* process key press */
   while (!done)
   {
      /* dump the keyboard input buffer */
      dump_key_buffer();

      /* wait for a key press */
      key = ki();

      /* process key */
      /* Note: ki() returns the negative of the key scan code  */
      /* if an extended code key is pressed (ie: UP, DOWN etc) */
      /* and returns the ASCII code directly otherwise         */
      switch (-key)
      {
         /* LEFT and UP arrows decrement the row */
         /* not including wrap around            */

         /* LEFT arrow */
         case K_LEFT   : ;

         /* UP arrow */
         case K_UP     : if (current_row > min_row)
```

```
                {
                   unhighlight_row(current_row, wn_ptr);
                   current_row--;
                   highlight_row(current_row, wn_ptr);
                }
                else
                /* return the negative of the Up arrow scan code */
                {
                   *return_code_ptr = -K_UP;
                   done = TRUE;
                }

                break;


/* RIGHT and DOWN arrows increment the row */
/* not including wrap around               */

/* RIGHT arrow */
case K_RIGHT  : ;

/* DOWN arrow */
case K_DN     : if (current_row < max_row)
                {
                   unhighlight_row(current_row, wn_ptr);
                   current_row++;
                   highlight_row(current_row, wn_ptr);
                }
                else
                /* return the negative of the Down arrow scan code */
                {
                   *return_code_ptr = -K_DN;
                   done = TRUE;
                }

                break;

/* PgUp, PgDN, Home or End */
/* return the negative of the scan code */
case K_PGUP   : ;
case K_PGDN   : ;
case K_HOME   : ;
case K_END    : *return_code_ptr = key;
                done = TRUE;
                break;


/* ESCAPE key */
/* return the negative of the escape key */
case -K_ESC   : *return_code_ptr = -K_ESC;
                done = TRUE;

                break;
```

```
         /* ENTER (Return) key */
         case -K_ENTER : done = TRUE;
                         break;

         default       : /* if the user has pressed an alphabetic key */
                         /* return the ASCII code for that letter in  */
                         /* uppercase                                 */
                         temp_key = toupper(key);
                         if (temp_key >= 'A' && temp_key <= 'Z')
                         {
                            *return_code_ptr = temp_key;
                            done = TRUE;
                         }
                         /* if the user presses a ?, # or SPACE */
                         /* return the corresponding key        */
                         else if ( (key == '?') ||
                                   (key == '#') ||
                                   (key == ' ') )
                         {
                            *return_code_ptr = key;
                            done = TRUE;
                         }
                         else
                            bell();

                         break;
      }
   }

   unhighlight_row(current_row, wn_ptr);

   /* the current row number must be reduced by the */
   /* minimum row number to reflect a choice number */
   return(current_row - min_row);
}

/* ------------------------------------------------------------------------ */

/* get_end_num_in_range returns the highest number within a set of numbers   */
/* from 0 to total_numbers-1 which will fit within the given range from      */
/* start_num                                                                 */

get_end_num_in_range(start_num, total_numbers, range)
int start_num, total_numbers, range;

{
   if ( (total_numbers - start_num) >= range)
      return(start_num + range - 1);
   else
      return(start_num + (total_numbers % range) - 1);
}

/* ------------------------------------------------------------------------ */
```

```
/* trunc_float returns the result of truncating the given value of x to the  */
/* given number of digits after the decimal point                            */
/* Note: num_digits must be greater than or equal to ZERO (0)                */

float trunc_float(x, num_digits)
float x;
int num_digits;

{
   double int_part;
   int count;

   /* shift left the given number of decimal digits */
   for (count = 0; count < num_digits; count++) x *= 10.0;

   /* truncate all remaining fractional part */
   modf(x, &int_part);
   x = (float)int_part;

   /* shift back right the given number of decimal digits */
   for (count = 0; count < num_digits; count++) x /= 10.0;

   return(x);
}

/* ------------------------------------------------------------------------- */

#define PRINTER_INTERRUPT 23
#define PRINTER_STATUS_SERVICE 2
#define ON_LINE_AND_NOT_BUSY 144

/* ------------------------------------------------------------------------- */

/* printer_ready returns TRUE only if the given printer (0-1) is on-line and */
/* not busy. Otherwise FALSE is returned.                                    */

printer_ready(printer_number)
int printer_number;

{
   union REGS regs;

   regs.h.ah = PRINTER_STATUS_SERVICE;
   regs.x.dx = printer_number;

   int86(PRINTER_INTERRUPT,&regs,&regs);

   return(regs.h.ah == ON_LINE_AND_NOT_BUSY);
}

/* ------------------------------------------------------------------------- */

#define DISK_INTERRUPT 19
#define DISK_VERIFY_SERVICE 4
```

```
#define BAD_ADDRESS_MARK 2
#define NUMBER_OF_SECTORS 1
#define DISK_SIDE 0
#define TRACK_NUMBER 10
#define SECTOR_NUMBER 1
#define MAX_TRIES 3

/* ------------------------------------------------------------------------ */

/* check_drive returns:                                                     */
/* DRIVE_OK if the drive associated with the given drive letter is ready    */
/* (ie: disk is in place and readable).                                     */
/* DRIVE_NOT_READY if the drive does not respond after MAX_TRIES are        */
/* attempted.                                                               */
/* DISK_NOT_READABLE if the disk in the drive is not readable               */
/* (ie: not formatted)                                                      */
/* Any drive letter other than A or B causes drive_ready to return DRIVE_OK */

check_drive(drive_letter)
char drive_letter;

{
   union REGS regs;
   int count, drive_number;

   if ( (drive_letter != 'A') && (drive_letter != 'B') )
      return(DRIVE_OK);

   drive_number = drive_letter - 'A';
   count = 0;

   while (count < MAX_TRIES)
   {
      /* attempt to verify a sector on the given drive */
      regs.h.ah = DISK_VERIFY_SERVICE;
      regs.h.al = NUMBER_OF_SECTORS;
      regs.h.dh = DISK_SIDE;
      regs.h.dl = drive_number;
      regs.h.ch = TRACK_NUMBER;
      regs.h.cl = SECTOR_NUMBER;

      int86(DISK_INTERRUPT,&regs,&regs);

      /* if the status byte is zero, the drive is ready */
      if (regs.h.ah == 0)
      {
         return(DRIVE_OK);
      }

      /* otherwise, assume a time-out has occurred. Try again. */

      count++;
   }
```

```
   /* if a bad address mark is found, the disk is not readable */
   if (regs.h.ah == BAD_ADDRESS_MARK)
   {
      return(DISK_NOT_READABLE);
   }

   /* drive has timed out MAX_TRIES times.  Drive is not ready. */
   return(DRIVE_NOT_READY);
}

/* ------------------------------------------------------------------------ */

#define DOS_EXTENDED_FUNCTION 33
#define DISK_FREE_SPACE_SERVICE 54
#define REPORT_CURRENT_DRIVE 25

/* ------------------------------------------------------------------------ */

/* free_space_on_drive returns the number of bytes available on the given    */
/* drive.  Note: the drive must be ready before calling this function.       */

float free_space_on_drive(drive_letter)
char drive_letter;

{
   union REGS regs;
   int drive_number;

   /* note that drive number 0 is the current default drive */
   /* A is 1, B is 2 etc.                                   */
   drive_number = drive_letter - 'A' + 1;

   regs.h.ah = DISK_FREE_SPACE_SERVICE;
   regs.h.dl = drive_number;

   int86(DOS_EXTENDED_FUNCTION,&regs,&regs);

   return((float)regs.x.ax * (float)regs.x.bx * (float)regs.x.cx);
}

/* ------------------------------------------------------------------------ */

/* get_current_drive_letter returns the letter of the current drive          */

get_current_drive_letter()

{
   union REGS regs;
   int current_drive;

   /* get the current drive number */
   regs.h.ah = REPORT_CURRENT_DRIVE;

   int86(DOS_EXTENDED_FUNCTION,&regs,&regs);
```

```
   current_drive = regs.h.al;

   return(current_drive + 'A');
}

/* ------------------------------------------------------------------------ */

/* user_acknowledge_wait waits for the user to press a key to acknowledge    */
/* some point in the execution of the program. Usually used to ensure that   */
/* the user has read a message window before it is removed.  The only keys   */
/* which are recognized are <ESC>, <RETURN> and <SPACE>                      */

user_acknowledge_wait()

{
   int done, key;

   /* wait for an <ESC>, <RETURN> or <SPACE> key press */

   done = FALSE;

   while (!done)
   {
      /* dump the keyboard input buffer */
      dump_key_buffer();

      /* wait for a key press */
      key = ki();

      /* process key */
      switch (key)
      {
         case K_ESC   : ;
         case K_ENTER : ;
         case K_SPACE : done = TRUE;
                        break;
         default      : break;
      }
   }

}

/* ------------------------------------------------------------------------ */

/* invert_det_nondet returns the result of inverting the given value to the  */
/* complement (ie: converts detection probabilities into non-detection and   */
/* vice versa)                                                               */

float invert_det_nondet(val)
float val;

{
   return(1.0 - val);
```

```
}

/* ------------------------------------------------------------------------ */

/* partition_seconds sets mins to the number of minutes in the given total   */
/* seconds and sets secs to the remaining number of seconds                   */

partition_seconds(total_seconds,mins,secs)
int total_seconds, *mins, *secs;

{
   *mins = total_seconds/60;
   *secs = total_seconds%60;

   return;
}

/* ------------------------------------------------------------------------ */

/* unpartition_seconds sets total_seconds to the total number of seconds in  */
/* the given minutes (mins) and seconds (secs) variables                      */

unpartition_seconds(total_seconds,mins,secs)
int *total_seconds, mins, secs;

{
   *total_seconds = 60 * mins + secs;

   return;
}

/* ------------------------------------------------------------------------ */

/* pause_sec pauses the given number of seconds                               */

pause_sec(secs)
int secs;

{
   long start;

   start = time(NULL);
   while (time(NULL)-start < secs);

   return;
}

/* ------------------------------------------------------------------------ */

savi_stop()

{
   display_quit();
   dump_key_buffer();
```

```
   return;
}

/* ------------------------------------------------------------------------ */

/* cardinal_number returns a pointer to a string containing the cardinal     */
/* number for the given number (ie: 1 returns "First ")                        */

char *cardinal_number(num)
int num;

{
   static char *cards[19] =
      {
         "First ",
         "Second ",
         "Third ",
         "Fourth ",
         "Fifth ",
         "Sixth ",
         "Seventh ",
         "Eighth ",
         "Ninth ",
         "Tenth ",
         "Eleventh ",
         "Twelfth ",
         "Thirteenth ",
         "Fourteenth ",
         "Fifteenth ",
         "Sixteenth ",
         "Seventeenth ",
         "Eighteenth ",
         "Nineteenth "
      };

   return(cards[num-1]);
}

/* ------------------------------------------------------------------------ */

/* right_just right justifies the characters within the given string with    */
/* leading blanks. str_dim is the dimension of the string str including the  */
/* terminating null byte.                                                    */

right_just(str,str_dim)
char str[];
int str_dim;

{
   int i, num_pad;

   /* calculate the number of pad characters */
   /* to add to the front of the string      */
```

```
   num_pad = str_dim - strlen(str) - 1;

   /* return if the string is already right justified */
   if (num_pad == 0) return;

   /* now slide all characters to right end of string */
   for (i = (str_dim - num_pad -1); i >= 0; i--)
   {
      str[i + num_pad] = str[i];
      str[i] = ' ';
   }

   return;
}

/* ------------------------------------------------------------------------ */

/* first_non_blank_char returns the first non-blank character in the given  */
/* string                                                                   */

first_non_blank_char(string)
char *string;

{
   int i;

   i = 0;
   while (string[i] == ' ') i++;

   return(string[i]);
}

/* ------------------------------------------------------------------------ */

/* last_char returns the last character in the given string                 */

last_char(string)
char *string;

{
   return(string[strlen(string)-1]);
}

/* ------------------------------------------------------------------------ */

/* dump_key_buffer dumps all characters from the keyboard input buffer      */

dump_key_buffer()

{
   while (kbhit()) getch();

   return;
}
```

```
/* ------------------------------------------------------------------------ */

/* search_key_buf returns TRUE if an instance of the given key ASCII code is */
/* found anywhere within the keyboard buffer. Returns FALSE otherwise. The   */
/* keyboard buffer is emptied up to the given key and is emptied completely  */
/* if the key is not found                                                   */

search_key_buf(key)
int key;

{
   int ch;

   while (kbhit())
   {
      if ((ch = getch()) == key) return(TRUE);
   }

   return(FALSE);
}

/* ------------------------------------------------------------------------ */

/* picdigit_only is the picture control function for the picture control     */
/* character '9' within the Windows for Data form field entry system         */
/* This function returns a 1 if the given key points to a valid ASCII code   */
/* for the '9' picture control character. In this case digits (0-9) are the  */
/* only valid characters, all other key codes cause function to return a 0   */

picdigit_only(key)
int *key;

{
   if ((*key >= '0' && *key <= '9') || (*key == ' '))
      return(1);
   else
      return(0);
}

/* ------------------------------------------------------------------------ */

/* picone is the picture control function for the picture control            */
/* character '1' within the Windows for Data form field entry system         */
/* This function returns a 1 if the given key points to a valid ASCII code   */
/* for the '1' picture control character. In this case digits (0-1) are the  */
/* only valid characters, all other key codes cause function to return a 0   */

picone(key)
int *key;

{
   if ((*key >= '0' && *key <= '1') || (*key == ' '))
      return(1);
```

```
   else
      return(0);
}

/* ------------------------------------------------------------------------ */

/* picfive is the picture control function for the picture control          */
/* character '5' within the Windows for Data form field entry system        */
/* This function returns a 1 if the given key points to a valid ASCII code  */
/* for the '5' picture control character. In this case digits (0-5) are the */
/* only valid characters, all other key codes cause function to return a 0  */

picfive(key)
int *key;

{
   if ((*key >= '0' && *key <= '5') || (*key == ' '))
      return(1);
   else
      return(0);
}

/* ------------------------------------------------------------------------ */

/* picfile_name is the picture control function for the picture control     */
/* character 'F' within the Windows for Data form field entry system        */
/* This function returns a 1 if the given key points to a valid ASCII code  */
/* for the 'F' picture control character. In this case any ASCII code is    */
/* valid except for ?*,.<>/\|+=[]";: due to the nature of these characters  */
/* in DOS file names.  A 0 is returned for invalid characters. The character */
/* which key points to is also converted to upper case if it is alphabetic.  */

picfile_name(key)
int *key;

{
   switch (*key)
   {
      case '?' : ;
      case '*' : ;
      case ',' : ;
      case '.' : ;
      case '<' : ;
      case '>' : ;
      case '/' : ;
      case '\\': ;
      case '|' : ;
      case '+' : ;
      case '=' : ;
      case '[' : ;
      case ']' : ;
      case '"' : ;
      case ';' : ;
      case ' ' : ;
```

```
      case ':' : return(0);
                 break;
      default  : *key = toupper(*key);
                 return(1);
                 break;
   }
}

/* ------------------------------------------------------------------------ */

/* click_speaker clicks the PC speaker                                      */

click_speaker()

{
   pulse_speaker_on();
   speaker_off();

   return;
}

/* ------------------------------------------------------------------------ */

/* error_tone plays a tone indicative of an error                           */

error_tone()

{
   set_tone_freq(55.0);
   speaker_on();
   wait_tics(7);
   speaker_off();

   return;
}

/* ------------------------------------------------------------------------ */

/* complete_tone plays a warble to indicate that analysis is complete       */

complete_tone()

{
   set_tone_freq(880.0);
   speaker_on();
   wait_tics(1);
   set_tone_freq(523.3);
   wait_tics(1);

   set_tone_freq(880.0);
   wait_tics(1);
   set_tone_freq(523.3);
   wait_tics(1);
```

```
   set_tone_freq(880.0);
   wait_tics(1);
   set_tone_freq(523.3);
   wait_tics(1);

   set_tone_freq(880.0);
   wait_tics(1);
   set_tone_freq(523.3);
   wait_tics(1);

   speaker_off();

  return;
}

/* ---------------------------------------------------------------------- */

#define timer_port 67
#define count_port 66
#define count_follows 182

/* ---------------------------------------------------------------------- */

/* set_tone_freq sets the speaker frequency to the given value            */
/* Note: freq must be between 50 and 15000 Hz                             */

set_tone_freq(freq)
float freq;

{
   int count;

   /* limit frequency to range from 50 to 15000 Hz */
   if (freq < 50.0)
      freq = 100.0;
   else if (freq > 15000.0)
      freq = 15000.0;

   count = (int)(1193180.0 / freq);

   /* set the given frequency */
   outp(timer_port,count_follows);
   outp(count_port,count % 256);
   outp(count_port,count / 256);

   return;
}

/* ---------------------------------------------------------------------- */

#define PPI 97

/* ---------------------------------------------------------------------- */
```

```
/* pulse_speaker_on directly pulses the PC speaker                          */

pulse_speaker_on()

{
   outp(PPI,inp(PPI) | 2);

   return;
}

/* ------------------------------------------------------------------------ */

/* speaker_on turns the PC speaker on, enabling control by the programmable  */
/* peripheral interface's (PPI) current frequency                           */

speaker_on()

{
   outp(PPI,inp(PPI) | 3);

   return;
}

/* ------------------------------------------------------------------------ */

/* speaker_off turns the PC speaker off                                     */

speaker_off()

{
   outp(PPI,inp(PPI) & 65532);

   return;
}

/* ------------------------------------------------------------------------ */

/* wait_tics waits the given number of clock tics (18.2 per second)         */

wait_tics(tics)
int tics;

{
   long int current, stop;

   get_tics(&current);

   stop = current + tics;

   while (current < stop) get_tics(&current);

   return;
}
```

```
/* ---------------------------------------------------------------------- */

#define BIOS_TIME_INTERRUPT 26
#define GET_CLOCK_COUNT 0

/* ---------------------------------------------------------------------- */

/* get_tics returns the current number of elapsed clock ticks (18.2 per sec) */
/* in the given tics variable                                                */

get_tics(tics)
long int *tics;

{
   union REGS regs;

   regs.h.ah = GET_CLOCK_COUNT;

   int86(BIOS_TIME_INTERRUPT,&regs,&regs);

   *tics = (unsigned long)regs.x.cx << 16 | regs.x.dx;

   return;
}

/* ********************** end SAVIMISC.C source file ********************** */

/* ************************ SAVIMODI.C source file ************************ */

#include <wfd.h>
#include <savidefs.h>

#define ASD_LABEL_WINDOW_TOP 16
#define ASD_LABEL_WINDOW_COL 25

/* ASD_LABEL_WINDOW_WIDTH must be wide enough to contain the */
/* ASD label window border, margins and maximum ASD label    */
#define ASD_LABEL_WINDOW_WIDTH 52
#define ASD_LABEL_WINDOW_HEIGHT 5

/* ---------------------------------------------------------------------- */

/* savi_modify_title allows the user to modify the current ASD title.      */
/* The current title is displayed in a popup window above the title window */
/* and the user may edit the label. Edit functions are provided by Windows */
/* for Data.                                                               */

savi_modify_title()

{
   DFIELDPTR field_ptr;
   char new_ASD_label[MAX_ASD_LABEL_LEN];

   /* copy the current ASD label into the new label */
```

```
   /* so that the def_fld function can display the  */
   /* current label for editing                     */
   strcpy(new_ASD_label,get_ASD_label());

   /* set the input form size */
   mod_wn(ASD_LABEL_WINDOW_TOP, ASD_LABEL_WINDOW_COL,
          ASD_LABEL_WINDOW_HEIGHT, ASD_LABEL_WINDOW_WIDTH,
          STDWNP);

   /* set the input form border */
   sw_border(BDR_LNP,STDWNP);

   /* create the input form */
   def_txt(0,0,
           "          Adversary Sequence Diagram Title          ",
           LCYAN);
   def_txt(1,0,
           " DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD ",
           LBLUE);

   /* define the input form field */
   field_ptr = def_fld(0,0,"",def_pic('X',MAX_ASD_LABEL_LEN-1),
                       F_STRING,new_ASD_label);

   /* set the input field location within the window */
   /* independent of the prompt                      */
   sf_location(2,1,field_ptr);

   /* accept input from user */
   if (rd_flds() != ABORT_FORM)
      set_current_ASD_saved(FALSE);

   /* set the new ASD label in the current ASD */
   set_ASD_label(new_ASD_label);

   /* update the ASD label window */
   display_ASD_label(get_ASD_label());

   dump_key_buffer();

   return(0);
}

/* ------------------------------------------------------------------------ */

/* savi_modify_element_presence allows the user to modify the presence of    */
/* protection elements in the current ASD by inserting or deleting any PE's  */
/* which are defined (present or not).                                       */
/* The task PE is not reachable since it cannot be deleted from the ASD      */

savi_modify_element_presence()

{
   int layer, PE_num, PE_temp, key, done, window_on;
```

```
PROT_ELEM *PE_ptr;

layer = 0;
PE_num = 0;

done = FALSE;
window_on = TRUE;

/* display all non-present PEs in the current ASD to allow  */
/* user the option of re-inserting non-present PEs as well  */
/* as deleting present PEs                                  */
/* Note: all present PEs must be currently on the screen    */
display_all_non_present_PEs();

/* highlight the first PE and display the label window */
popup_PE_label_window(layer,PE_num);
display_PE_label(layer,PE_num);
highlight_PE(layer,PE_num);

/* process key press */
while (!done)
{
   /* dump the keyboard input buffer */
   dump_key_buffer();

   /* wait for a key press */
   key = ki();

   /* process key */
   /* Note: ki() returns the negative of the key scan code  */
   /* if an extended code key is pressed (ie: UP, DOWN etc) */
   /* and returns the ASCII code directly otherwise         */
   switch (-key)
   {
      /* UP arrow */
      /* highlight PE above current PE */
      /* do not wrap around top to bottom */
      /* if new PE is null then move to last non-null PE */
      /* on the new layer                                */
      case K_UP    : unhighlight_PE(layer,PE_num);
                     if (layer > 0)
                        {
                           layer--;
                           if (get_ASD_PE_ptr(layer,PE_num)->type == 0)
                              PE_num = last_non_null_PE_num_on_layer(layer);
                        }
                     else
                        bell();
                     if (window_on)
                     {
                        popup_PE_label_window(layer,PE_num);
                        display_PE_label(layer,PE_num);
                     }
                     highlight_PE(layer,PE_num);
```

```
                break;

/* DOWN arrow */
/* highlight PE below current PE */
/* do not wrap around bottom to top*/
/* if new PE is null then move to last non-null PE */
/* on the new layer                                */
case K_DN    : unhighlight_PE(layer,PE_num);
               if (layer < MAX_LAYERS_PER_ASD-1)
                  {
                     layer++;
                     if (get_ASD_PE_ptr(layer,PE_num)->type == 0)
                        PE_num = last_non_null_PE_num_on_layer(layer);
                  }
               else
                  bell();
               if (window_on)
               {
                  popup_PE_label_window(layer,PE_num);
                  display_PE_label(layer,PE_num);
               }
               highlight_PE(layer,PE_num);
               break;

/* LEFT arrow */
/* highlight PE to the left of current PE */
/* wrap around to last non-null PE on right */
case K_LEFT  : unhighlight_PE(layer,PE_num);
               if (PE_num > 0)
                  PE_num--;
               else
                  PE_num = last_non_null_PE_num_on_layer(layer);
               if (window_on)
               {
                  popup_PE_label_window(layer,PE_num);
                  display_PE_label(layer,PE_num);
               }
               highlight_PE(layer,PE_num);
               break;

/* RIGHT arrow */
/* highlight PE to the right of current PE */
/* wrap around to first PE on left */
case K_RIGHT : unhighlight_PE(layer,PE_num);
               if (PE_num < MAX_PE_PER_LAYER-1 &&
                   get_ASD_PE_ptr(layer,PE_num+1)->type != 0)
                  PE_num++;
               else
                  PE_num = 0;
               if (window_on)
               {
                  popup_PE_label_window(layer,PE_num);
                  display_PE_label(layer,PE_num);
               }
```

```
                highlight_PE(layer,PE_num);
                break;

/* HOME key */
/* highlight the furthest left PE */
case K_HOME   : if (PE_num == 0) break;
                unhighlight_PE(layer,PE_num);
                PE_num = 0;
                if (window_on)
                {
                   popup_PE_label_window(layer,PE_num);
                   display_PE_label(layer,PE_num);
                }
                highlight_PE(layer,PE_num);
                break;

/* END key */
/* highlight the furthest right non_null PE */
case K_END    : PE_temp = last_non_null_PE_num_on_layer(layer);
                if (PE_num == PE_temp) break;
                unhighlight_PE(layer,PE_num);
                PE_num = PE_temp;
                if (window_on)
                {
                   popup_PE_label_window(layer,PE_num);
                   display_PE_label(layer,PE_num);
                }
                highlight_PE(layer,PE_num);
                break;

/* PAGE UP key */
/* highlight the PE on the top layer */
case K_PGUP   : unhighlight_PE(layer,PE_num);
                if (layer > 0)
                   {
                      layer = 0;
                      if (get_ASD_PE_ptr(layer,PE_num)->type == 0)
                         PE_num = last_non_null_PE_num_on_layer(layer);
                   }
                else
                   bell();
                if (window_on)
                {
                   popup_PE_label_window(layer,PE_num);
                   display_PE_label(layer,PE_num);
                }
                highlight_PE(layer,PE_num);
                break;

/* PAGE DOWN key */
/* highlight the PE on the bottom layer */
case K_PGDN   : unhighlight_PE(layer,PE_num);
                if (layer < MAX_LAYERS_PER_ASD-1)
                   {
```

```
                        layer = MAX_LAYERS_PER_ASD-1;
                        if (get_ASD_PE_ptr(layer,PE_num)->type == 0)
                           PE_num = last_non_null_PE_num_on_layer(layer);
                     }
                  else
                     bell();
                  if (window_on)
                  {
                     popup_PE_label_window(layer,PE_num);
                     display_PE_label(layer,PE_num);
                  }
                  highlight_PE(layer,PE_num);
                  break;

/* INSERT key */
/* insert the currently highlighted PE into the current ASD */
/* redisplay to indicate that it has been inserted          */
/* also erase bypass bar for current layer if this is the   */
/* first PE to be inserted on this layer (ie: bypass is     */
/* currently TRUE)                                          */
case K_INS    : PE_ptr = get_ASD_PE_ptr(layer,PE_num);
                  if (PE_ptr->present) break;

                  /* do not allow HEL to be inserted if current */
                  /* threat does not have a helicopter          */
                  if ( (get_PE_cat_spec_ptr(PE_ptr->type)->helicopter) &&
                       (get_threat_transport_mode(get_current_threat())
                            != HELICOPTER) )
                  {
                     display_helicopter_insert_error();
                     user_acknowledge_wait();
                     remove_helicopter_insert_error_window();
                     break;
                  }

                  set_vulnerability_current(FALSE);

                  PE_ptr->present = TRUE;
                  display_PE(PE_ptr,layer,PE_num);
                  highlight_PE(layer,PE_num);

                  /* if the bypass was set prior to inserting */
                  /* this PE then remove the bypass           */
                  if (get_ASD_layer_bypass(layer))
                  {
                     /* remove the PE label window to ensure */
                     /* that it does not obscure the bypass  */
                     remove_PE_label_window();

                     /* turn the bypass off */
                     set_ASD_layer_bypass(layer,FALSE);
                     erase_layer_bypass(layer);
                  }
```

```
                if (window_on)
                {
                   popup_PE_label_window(layer,PE_num);
                   display_PE_label(layer,PE_num);
                }
                break;

/* DELETE key */
/* delete the currently highlighted PE from the current ASD */
/* redisplay to indicate that it has been deleted           */
case K_DEL    : PE_ptr = get_ASD_PE_ptr(layer,PE_num);
                if (!(PE_ptr->present)) break;

                set_vulnerability_current(FALSE);

                PE_ptr->present = FALSE;
                display_PE(PE_ptr,layer,PE_num);
                highlight_PE(layer,PE_num);

                /* if no PEs are present after deleting this */
                /* PE then set the bypass                    */
                if (!(PEs_present(layer,0,MAX_PE_PER_LAYER-1)))
                {
                   /* remove the PE label window to ensure */
                   /* that it does not obscure the bypass  */
                   remove_PE_label_window();

                   /* turn the bypass on */
                   set_ASD_layer_bypass(layer,TRUE);
                   display_layer_bypass(layer);

                }

                if (window_on)
                {
                   popup_PE_label_window(layer,PE_num);
                   display_PE_label(layer,PE_num);
                }
                break;

/* SUPER key '*' */
/* test for INS or DEL immediately following */
case -'*'     : key = ki();

                if (-key == K_INS)
                /* insert all non-null PEs on current layer  */
                /* Note: do not insert HEL PE if the current */
                /* threat does not have a helicopter         */
                {
                   insert_non_null_PE_layer(layer);

                   set_vulnerability_current(FALSE);

                   /* remove the PE label window to ensure */
```

```
                    /* that it does not obscure the layer   */
                    remove_PE_label_window();

                    display_non_null_PE_layer(layer,0);

                    if (window_on)
                    {
                       popup_PE_label_window(layer,PE_num);
                       display_PE_label(layer,PE_num);
                    }

                    /* re-highlight the current PE */
                    highlight_PE(layer,PE_num);

                    break;
                 }

                 if (-key == K_DEL)
                 /* delete all non-null PEs on current layer */
                 {
                    delete_non_null_PE_layer(layer);

                    set_vulnerability_current(FALSE);

                    /* remove the PE label window to ensure */
                    /* that it does not obscure the layer   */
                    remove_PE_label_window();

                    display_non_null_PE_layer(layer,0);

                    if (window_on)
                    {
                       popup_PE_label_window(layer,PE_num);
                       display_PE_label(layer,PE_num);
                    }

                    /* re-highlight the current PE */
                    highlight_PE(layer,PE_num);

                    break;
                 }
                 bell();
                 break;

/* ENTER (Return) key */
/* toggle the present state of the highlighted PE */
case -K_ENTER : PE_ptr = get_ASD_PE_ptr(layer,PE_num);

                 PE_ptr->present = !PE_ptr->present;

                 /* determine if the PE was inserted or deleted */
                 if (!PE_ptr->present)
                 {
                    /* PE was deleted */
```

```
            /* if no PEs are present after deleting this */
            /* PE then set the bypass                    */
            if (!(PEs_present(layer,0,MAX_PE_PER_LAYER-1)))
            {
               /* remove the PE label window to ensure */
               /* that it does not obscure the bypass  */
               remove_PE_label_window();

               /* turn the bypass on */
               set_ASD_layer_bypass(layer,TRUE);
               display_layer_bypass(layer);
            }
         }
         else
         {
            /* PE was inserted */

            /* do not allow HEL to be inserted if current */
            /* threat does not have a helicopter          */
            if ( (get_PE_cat_spec_ptr(PE_ptr->type)->
                     helicopter) &&
                 (get_threat_transport_mode(
                     get_current_threat()) != HELICOPTER) )
            {
               PE_ptr->present = FALSE;
               display_helicopter_insert_error();
               user_acknowledge_wait();
               remove_helicopter_insert_error_window();
               break;
            }

            /* if the bypass was set prior to inserting */
            /* this PE then remove the bypass           */
            if (get_ASD_layer_bypass(layer))
            {
               /* remove the PE label window to ensure */
               /* that it does not obscure the bypass  */
               remove_PE_label_window();

               /* turn the bypass off */
               set_ASD_layer_bypass(layer,FALSE);
               erase_layer_bypass(layer);
            }
         }

         display_PE(PE_ptr,layer,PE_num);
         highlight_PE(layer,PE_num);

         if (window_on)
         {
            popup_PE_label_window(layer,PE_num);
            display_PE_label(layer,PE_num);
         }
```

```
                        set_vulnerability_current(FALSE);

                        break;

          /* ESCAPE key */
          case -K_ESC   : done = TRUE;
                        break;

         /* SPACE bar */
         /* toggle the window display */
         case -K_SPACE : if (popup_window_on_screen())
                        {
                           window_on = FALSE;
                           remove_PE_label_window();
                        }
                        else
                        {
                           window_on = TRUE;
                           popup_PE_label_window(layer,PE_num);
                           display_PE_label(layer,PE_num);
                        }

                        break;

          default       : bell();
                        break;

      }
   }

   /* unhighlight the current PE and remove the PE label window */
   unhighlight_PE(layer,PE_num);
   remove_PE_label_window();

   /* change ASD display back to only show present PEs */
   erase_all_non_present_PEs();

   dump_key_buffer();

   return(0);
}

/* ------------------------------------------------------------------------ */

/* savi_modify_element_jumps allows the user to modify the output segment    */
/* connection of any PE which is defined (present or not) in the current ASD */
/* on all layers which can jump forward (toward target)                      */

savi_modify_element_jumps()

{
   int layer, PE_num, PE_temp, key, done, window_on;
   PROT_ELEM *PE_ptr;
```

```
layer = 0;
PE_num = 0;
done = FALSE;
window_on = TRUE;

/* display all non-present PEs in the current ASD to allow  */
/* user the option of modifying the jump for non-present    */
/* PEs as well as present PEs                               */
/* Note: all present PEs must be currently on the screen    */
display_all_non_present_PEs();

/* highlight the first PE and display the label window */
popup_PE_label_window(layer,PE_num);
display_PE_label(layer,PE_num);
highlight_PE(layer,PE_num);

/* process key press */
while (!done)
{
   /* dump the keyboard input buffer */
   dump_key_buffer();

   /* wait for a key press */
   key = ki();

   /* process key */
   /* Note: ki() returns the negative of the key scan code  */
   /* if an extended code key is pressed (ie: UP, DOWN etc) */
   /* and returns the ASCII code directly otherwise         */
   switch (-key)
   {
      /* UP arrow */
      /* highlight PE above current PE */
      /* do not wrap around top to bottom */
      /* if new PE is null then move to last non-null PE */
      /* on the new layer                                */
      case K_UP    : unhighlight_PE(layer,PE_num);
                     if (layer > 0)
                        {
                           layer--;
                           if (get_ASD_PE_ptr(layer,PE_num)->type == 0)
                              PE_num = last_non_null_PE_num_on_layer(layer);
                        }
                     else
                        bell();
                     if (window_on)
                     {
                        popup_PE_label_window(layer,PE_num);
                        display_PE_label(layer,PE_num);
                     }
                     highlight_PE(layer,PE_num);
                     break;

      /* DOWN arrow */
```

```
/* highlight PE below current PE */
/* do not wrap around bottom to top */
/* if new PE is null then move to last non-null PE */
/* on the new layer                                */
case K_DN    : unhighlight_PE(layer,PE_num);
               if (layer < MAX_LAYERS_PER_ASD-2)
                  {
                     layer++;
                     if (get_ASD_PE_ptr(layer,PE_num)->type == 0)
                        PE_num = last_non_null_PE_num_on_layer(layer);
                  }
               else
                  bell();
               if (window_on)
               {
                  popup_PE_label_window(layer,PE_num);
                  display_PE_label(layer,PE_num);
               }
               highlight_PE(layer,PE_num);
               break;

/* LEFT arrow */
/* highlight PE to the left of current PE */
/* wrap around to last non-null PE on right */
case K_LEFT  : unhighlight_PE(layer,PE_num);
               if (PE_num > 0)
                  PE_num--;
               else
                  PE_num = last_non_null_PE_num_on_layer(layer);
               if (window_on)
               {
                  popup_PE_label_window(layer,PE_num);
                  display_PE_label(layer,PE_num);
               }
               highlight_PE(layer,PE_num);
               break;

/* RIGHT arrow */
/* highlight PE to the right of current PE */
/* wrap around to first PE on left */
case K_RIGHT : unhighlight_PE(layer,PE_num);
               if (PE_num < MAX_PE_PER_LAYER-1 &&
                   get_ASD_PE_ptr(layer,PE_num+1)->type != 0)
                  PE_num++;
               else
                  PE_num = 0;
               if (window_on)
               {
                  popup_PE_label_window(layer,PE_num);
                  display_PE_label(layer,PE_num);
               }
               highlight_PE(layer,PE_num);
               break;
```

```
/* HOME key */
/* highlight the furthest left PE */
case K_HOME  : if (PE_num == 0) break;
               unhighlight_PE(layer,PE_num);
               PE_num = 0;
               if (window_on)
               {
                  popup_PE_label_window(layer,PE_num);
                  display_PE_label(layer,PE_num);
               }
               highlight_PE(layer,PE_num);
               break;

/* END key */
/* highlight the furthest right non_null PE */
case K_END   : PE_temp = last_non_null_PE_num_on_layer(layer);
               if (PE_num == PE_temp) break;
               unhighlight_PE(layer,PE_num);
               PE_num = PE_temp;
               if (window_on)
               {
                  popup_PE_label_window(layer,PE_num);
                  display_PE_label(layer,PE_num);
               }
               highlight_PE(layer,PE_num);
               break;

/* PAGE UP key */
/* highlight the PE on the top layer */
case K_PGUP  : unhighlight_PE(layer,PE_num);
               if (layer > 0)
                  {
                     layer = 0;
                     if (get_ASD_PE_ptr(layer,PE_num)->type == 0)
                        PE_num = last_non_null_PE_num_on_layer(layer);
                  }
               else
                  bell();
               if (window_on)
               {
                  popup_PE_label_window(layer,PE_num);
                  display_PE_label(layer,PE_num);
               }
               highlight_PE(layer,PE_num);
               break;

/* PAGE DOWN key */
/* highlight the PE on the top layer */
case K_PGDN  : unhighlight_PE(layer,PE_num);
               if (layer < MAX_LAYERS_PER_ASD-2)
                  {
                     layer = MAX_LAYERS_PER_ASD-2;
                     if (get_ASD_PE_ptr(layer,PE_num)->type == 0)
                        PE_num = last_non_null_PE_num_on_layer(layer);
```

```
                        }
                     else
                        bell();
                     if (window_on)
                     {
                        popup_PE_label_window(layer,PE_num);
                        display_PE_label(layer,PE_num);
                     }
                     highlight_PE(layer,PE_num);
                     break;

      /* ENTER (Return) key */
      /* allow user to change the highlighted PE's Jump */
      case -K_ENTER : if (!window_on)
                      {
                         popup_PE_label_window(layer,PE_num);
                         display_PE_label(layer,PE_num);
                      }

                      select_PE_jump(layer,PE_num);

                      if (!window_on)
                        remove_PE_label_window();

                      break;

      /* ESCAPE key */
      case -K_ESC   : done = TRUE;
                      break;

     /* SPACE bar */
     /* toggle the window display */
     case -K_SPACE : if (popup_window_on_screen())
                     {
                        window_on = FALSE;
                        remove_PE_label_window();
                     }
                     else
                     {
                        window_on = TRUE;
                        popup_PE_label_window(layer,PE_num);
                        display_PE_label(layer,PE_num);
                     }

                     break;

      default      : bell();
                     break;

   }
}

/* unhighlight the current PE and remove the PE label window */
unhighlight_PE(layer,PE_num);
```

```
   remove_PE_label_window();

   /* change ASD display back to only show present PEs */
   erase_all_non_present_PEs();

   dump_key_buffer();

   return(0);
}

/* ------------------------------------------------------------------------ */

/* savi_modify_element_types allows the user to modify the PE type of any PE */
/* which is defined (present or not) in the current ASD.                     */

savi_modify_element_types()

{
   int layer, PE_num, PE_temp, key, done, window_on;
   PROT_ELEM *PE_ptr;

   layer = 0;
   PE_num = 0;
   done = FALSE;
   window_on = TRUE;

   /* display all non-present PEs in the current ASD to allow  */
   /* user the option of modifying the type for non-present    */
   /* PEs as well as present PEs                               */
   /* Note: all present PEs must be currently on the screen    */
   display_all_non_present_PEs();

   /* highlight the first PE and display the label window */
   popup_PE_label_window(layer,PE_num);
   display_PE_label(layer,PE_num);
   highlight_PE(layer,PE_num);

   /* process key press */
   while (!done)
   {
      /* dump the keyboard input buffer */
      dump_key_buffer();

      /* wait for a key press */
      key = ki();

      /* process key */
      /* Note: ki() returns the negative of the key scan code  */
      /* if an extended code key is pressed (ie: UP, DOWN etc) */
      /* and returns the ASCII code directly otherwise         */
      switch (-key)
      {
         /* UP arrow */
         /* highlight PE above current PE */
```

```
/* do not wrap around top to bottom */
/* if new PE is null then move to last non-null PE */
/* on the new layer                                */
case K_UP    : unhighlight_PE(layer,PE_num);
               if (layer > 0)
                  {
                     layer--;
                     if (get_ASD_PE_ptr(layer,PE_num)->type == 0)
                        PE_num = last_non_null_PE_num_on_layer(layer);
                  }
               else
                  bell();
               if (window_on)
               {
                  popup_PE_label_window(layer,PE_num);
                  display_PE_label(layer,PE_num);
               }
               highlight_PE(layer,PE_num);
               break;

/* DOWN arrow */
/* highlight PE below current PE */
/* do not wrap around bottom to top*/
/* if new PE is null then move to last non-null PE */
/* on the new layer                                */
case K_DN    : unhighlight_PE(layer,PE_num);
               if (layer < MAX_LAYERS_PER_ASD-1)
                  {
                     layer++;
                     if (get_ASD_PE_ptr(layer,PE_num)->type == 0)
                        PE_num = last_non_null_PE_num_on_layer(layer);
                  }
               else
                  bell();
               if (window_on)
               {
                  popup_PE_label_window(layer,PE_num);
                  display_PE_label(layer,PE_num);
               }
               highlight_PE(layer,PE_num);
               break;

/* LEFT arrow */
/* highlight PE to the left of current PE */
/* wrap around to last non-null PE on right */
case K_LEFT  : unhighlight_PE(layer,PE_num);
               if (PE_num > 0)
                  PE_num--;
               else
                  PE_num = last_non_null_PE_num_on_layer(layer);
               if (window_on)
               {
                  popup_PE_label_window(layer,PE_num);
                  display_PE_label(layer,PE_num);
```

```
                }
                highlight_PE(layer,PE_num);
                break;

/* RIGHT arrow */
/* highlight PE to the right of current PE */
/* wrap around to first PE on left */
case K_RIGHT : unhighlight_PE(layer,PE_num);
                if (PE_num < MAX_PE_PER_LAYER-1 &&
                    get_ASD_PE_ptr(layer,PE_num+1)->type != 0)
                   PE_num++;
                else
                   PE_num = 0;
                if (window_on)
                {
                   popup_PE_label_window(layer,PE_num);
                   display_PE_label(layer,PE_num);
                }
                highlight_PE(layer,PE_num);
                break;

/* HOME key */
/* highlight the furthest left PE */
case K_HOME  : if (PE_num == 0) break;
                unhighlight_PE(layer,PE_num);
                PE_num = 0;
                if (window_on)
                {
                   popup_PE_label_window(layer,PE_num);
                   display_PE_label(layer,PE_num);
                }
                highlight_PE(layer,PE_num);
                break;

/* END key */
/* highlight the furthest right non_null PE */
case K_END   : PE_temp = last_non_null_PE_num_on_layer(layer);
                if (PE_num == PE_temp) break;
                unhighlight_PE(layer,PE_num);
                PE_num = PE_temp;
                if (window_on)
                {
                   popup_PE_label_window(layer,PE_num);
                   display_PE_label(layer,PE_num);
                }
                highlight_PE(layer,PE_num);
                break;

/* PAGE UP key */
/* highlight the PE on the top layer */
case K_PGUP  : unhighlight_PE(layer,PE_num);
                if (layer > 0)
                   {
                      layer = 0;
```

```
                        if (get_ASD_PE_ptr(layer,PE_num)->type == 0)
                           PE_num = last_non_null_PE_num_on_layer(layer);
                     }
                  else
                     bell();
                  if (window_on)
                  {
                     popup_PE_label_window(layer,PE_num);
                     display_PE_label(layer,PE_num);
                  }
                  highlight_PE(layer,PE_num);
                  break;

 /* PAGE DOWN key */
 /* highlight the PE on the top layer */
 case K_PGDN  : unhighlight_PE(layer,PE_num);
                  if (layer < MAX_LAYERS_PER_ASD-1)
                     {
                        layer = MAX_LAYERS_PER_ASD-1;
                        if (get_ASD_PE_ptr(layer,PE_num)->type == 0)
                           PE_num = last_non_null_PE_num_on_layer(layer);
                     }
                  else
                     bell();
                  if (window_on)
                  {
                     popup_PE_label_window(layer,PE_num);
                     display_PE_label(layer,PE_num);
                  }
                  highlight_PE(layer,PE_num);
                  break;

 /* ENTER <Return> key */
 /* allow user to change the highlighted PE's Type */
 case -K_ENTER : if (!window_on)
                  {
                     popup_PE_label_window(layer,PE_num);
                     display_PE_label(layer,PE_num);
                  }

                  select_PE_type(layer,PE_num);

                  if (!window_on)
                    remove_PE_label_window();

                  break;

 /* ESCAPE key */
 case -K_ESC   : done = TRUE;
                  break;

/* SPACE bar */
/* toggle the window display */
case -K_SPACE : if (popup_window_on_screen())
```

```
                        {
                           window_on = FALSE;
                           remove_PE_label_window();
                        }
                        else
                        {
                           window_on = TRUE;
                           popup_PE_label_window(layer,PE_num);
                           display_PE_label(layer,PE_num);
                        }

                        break;

          default      : bell();
                         break;

      }
   }

   /* unhighlight the current PE and remove the PE label window */
   unhighlight_PE(layer,PE_num);
   remove_PE_label_window();

   /* change ASD display back to only show present PEs */
   erase_all_non_present_PEs();

   dump_key_buffer();

   return(0);
}

/* ------------------------------------------------------------------------ */

/* insert_non_null_PE_layer sets all non-null PEs on the given layer to be   */
/* present in the current ASD                                                */

insert_non_null_PE_layer(layer)
int layer;

{
   int PE_num;
   PROT_ELEM *PE_ptr;

   for (PE_num = 0; PE_num < MAX_PE_PER_LAYER; PE_num++)
   {
      PE_ptr = get_ASD_PE_ptr(layer,PE_num);
      if (PE_ptr->type != 0)
      {
         /* do not insert any HEL PEs unless the current */
         /* threat has a helicopter                      */
         if ( (!get_PE_cat_spec_ptr(PE_ptr->type)->helicopter) ||
              (get_threat_transport_mode(get_current_threat()) == HELICOPTER) )
            PE_ptr->present = TRUE;
      }
```

```
   }

   /* unset the bypass for this layer */
   set_ASD_layer_bypass(layer,FALSE);

   return;
}

/* ------------------------------------------------------------------------ */

/* delete_non_null_PE_layer sets all non-null PEs on the given layer to be  */
/* not present in the current ASD                                           */

delete_non_null_PE_layer(layer)
int layer;

{
   int PE_num;
   PROT_ELEM *PE_ptr;

   for (PE_num = 0; PE_num < MAX_PE_PER_LAYER; PE_num++)
   {
      PE_ptr = get_ASD_PE_ptr(layer,PE_num);
      if (PE_ptr->type != 0) PE_ptr->present = FALSE;
   }

   /* set the bypass for this layer */
   set_ASD_layer_bypass(layer,TRUE);

   return;
}

/* ------------------------------------------------------------------------ */

/* select_PE_jump allows the user to select a new PE jump for the PE defined */
/* by the given layer and PE number.  The user can select the new PE jump by */
/* pressing <Enter> or <Return> or the original jump can be restored by      */
/* pressing Escape <Esc>.                                                    */

select_PE_jump(layer,PE_num)
int layer, PE_num;

{
   int key, start_output_area, current_output_area,
       first_valid_output_area, done, input_char;
   PROT_ELEM *PE_ptr;

   PE_ptr = get_ASD_PE_ptr(layer,PE_num);
   start_output_area = PE_ptr->output_area;
   current_output_area = start_output_area;

   /* the first valid output area for the given PE     */
   /* is the next area below (ie: jump back not valid) */
   first_valid_output_area = layer + 1;
```

```
done = FALSE;

/* highlight the current PE output_area within the PE label window */
highlight_PE_output_area();

/* process key press */
while (!done)
{
   /* dump the keyboard input buffer */
   dump_key_buffer();

   /* wait for a key press */
   key = ki();

   /* process key */
   /* Note: ki() returns the negative of the key scan code  */
   /* if an extended code key is pressed (ie: UP, DOWN etc) */
   /* and returns the ASCII code directly otherwise         */
   switch (-key)
   {
      /* LEFT and UP arrows decrement the output area */
      /* including wrap around                        */
      case K_LEFT   : ;
      case K_UP     : if (current_output_area > first_valid_output_area)
                         current_output_area--;
                      else
                         current_output_area = MAX_LAYERS_PER_ASD;

                      PE_ptr->output_area = current_output_area;

                      /* redisplay the PE label to reflect */
                      /* the new PE output_area            */
                      display_PE_label(layer,PE_num);
                      highlight_PE_output_area();

                      /* redisplay the PE itself to reflect */
                      /* the new PE output_area             */
                      display_PE(PE_ptr,layer,PE_num);
                      highlight_PE(layer,PE_num);

                      break;


      /* RIGHT and DOWN arrows increment the output_area */
      /* including wrap around                           */
      case K_RIGHT  : ;
      case K_DN     : if (current_output_area < MAX_LAYERS_PER_ASD)
                         current_output_area++;
                      else
                         current_output_area = first_valid_output_area;

                      PE_ptr->output_area = current_output_area;
```

```
                /* redisplay the PE label to reflect */
                /* the new PE output_area            */
                display_PE_label(layer,PE_num);
                highlight_PE_output_area();

                /* redisplay the PE itself to reflect */
                /* the new PE output area             */
                display_PE(PE_ptr,layer,PE_num);
                highlight_PE(layer,PE_num);

                break;

/* HOME and PAGE UP keys set output area to minimum */
case K_HOME   : ;
case K_PGUP   : current_output_area = first_valid_output_area;

                PE_ptr->output_area = current_output_area;

                /* redisplay the PE label to reflect */
                /* the new PE output_area            */
                display_PE_label(layer,PE_num);
                highlight_PE_output_area();

                /* redisplay the PE itself to reflect */
                /* the new PE output area             */
                display_PE(PE_ptr,layer,PE_num);
                highlight_PE(layer,PE_num);

                break;

/* END and PAGE DOWN keys set output area to maximum */
case K_END    : ;
case K_PGDN   : current_output_area = MAX_LAYERS_PER_ASD;

                PE_ptr->output_area = current_output_area;

                /* redisplay the PE label to reflect */
                /* the new PE output_area            */
                display_PE_label(layer,PE_num);
                highlight_PE_output_area();

                /* redisplay the PE itself to reflect */
                /* the new PE output_area             */
                display_PE(PE_ptr,layer,PE_num);
                highlight_PE(layer,PE_num);

                break;

/* ESCAPE key */
/* set the output_area back to the state prior to */
/* selecting this option                          */
case -K_ESC   : PE_ptr->output_area = start_output_area;

                /* redisplay the PE label to reflect */
```

```
                        /* the original PE output_area       */
                        display_PE_label(layer,PE_num);

                        /* redisplay the PE itself to reflect */
                        /* the original PE output_area        */
                        display_PE(PE_ptr,layer,PE_num);
                        highlight_PE(layer,PE_num);

                        done = TRUE;
                        break;

        /* ENTER (Return) key */
        case -K_ENTER : display_PE_label(layer,PE_num);
                        done = TRUE;

                        if (current_output_area != start_output_area)
                        {
                           set_vulnerability_current(FALSE);
                        }
                        break;

        /* check for the letters corresponding to the desired output area */
        default       : input_char = toupper(key);
                        if ( (input_char >= 'A'+ first_valid_output_area-2) &&
                             (input_char <= 'D') )
                        {
                           PE_ptr->output_area = input_char - 'A' + 2;

                           /* redisplay the PE label to reflect */
                           /* the new PE output_area            */
                           display_PE_label(layer,PE_num);

                           /* redisplay the PE itself to reflect */
                           /* the new PE output_area             */
                           display_PE(PE_ptr,layer,PE_num);
                           highlight_PE(layer,PE_num);

                           done = TRUE;
                        }
                        else
                           bell();
                        break;
      }
   }

   return;
}

/* ------------------------------------------------------------------------ */

/* select_PE_type allows the user to select a new PE type for the PE defined */
/* by the given layer and PE number.  The user can select the new PE type by */
/* pressing (Enter) or (Return) or the original type can be restored by      */
/* pressing Escape (Esc). If the user selects a new PE type, the PE settings */
```

```
/* are cleared                                                          */

select_PE_type(layer,PE_num)
int layer, PE_num;

{
   int key, start_type, current_type, done;
   PROT_ELEM *PE_ptr;

   PE_ptr = get_ASD_PE_ptr(layer,PE_num);
   start_type = PE_ptr->type;
   current_type = start_type;

   done = FALSE;

   /* highlight the current PE type within the PE label window */
   highlight_PE_label();

   /* process key press */
   while (!done)
   {
      /* dump the keyboard input buffer */
      dump_key_buffer();

      /* wait for a key press */
      key = ki();

      /* process key */
      /* Note: ki() returns the negative of the key scan code  */
      /* if an extended code key is pressed (ie: UP, DOWN etc) */
      /* and returns the ASCII code directly otherwise         */
      switch (-key)
      {
         /* LEFT and UP arrows decrement the type */
         /* including wrap around                 */
         case K_LEFT   : ;
         case K_UP     : if (current_type > 1)
                            current_type--;
                         else
                            current_type = LAST_DEFINED_PE_TYPE;

                         PE_ptr->type = current_type;

                         /* redisplay the PE label to reflect */
                         /* the new PE type                   */
                         display_PE_label(layer,PE_num);
                         highlight_PE_label();

                         /* redisplay the PE itself to reflect */
                         /* the new PE type acronym            */
                         display_PE(PE_ptr,layer,PE_num);
                         highlight_PE(layer,PE_num);

                         break;
```

```
/* RIGHT and DOWN arrows increment the type */
/* including wrap around                    */
case K_RIGHT  : ;
case K_DN     : if (current_type < LAST_DEFINED_PE_TYPE)
                   current_type++;
                else
                   current_type = 1;

                PE_ptr->type = current_type;

                /* redisplay the PE label to reflect */
                /* the new PE type                   */
                display_PE_label(layer,PE_num);
                highlight_PE_label();

                /* redisplay the PE itself to reflect */
                /* the new PE type acronym            */
                display_PE(PE_ptr,layer,PE_num);
                highlight_PE(layer,PE_num);

                break;

/* HOME and PAGE UP keys set type to minimum */
case K_HOME   : ;
case K_PGUP   : current_type = 1;

                PE_ptr->type = current_type;

                /* redisplay the PE label to reflect */
                /* the new PE type                   */
                display_PE_label(layer,PE_num);
                highlight_PE_label();

                /* redisplay the PE itself to reflect */
                /* the new PE type acronym            */
                display_PE(PE_ptr,layer,PE_num);
                highlight_PE(layer,PE_num);

                break;

/* END and PAGE DOWN keys set type to maximum */
case K_END    : ;
case K_PGDN   : current_type = LAST_DEFINED_PE_TYPE;

                PE_ptr->type = current_type;

                /* redisplay the PE label to reflect */
                /* the new PE type                   */
                display_PE_label(layer,PE_num);
                highlight_PE_label();

                /* redisplay the PE itself to reflect */
                /* the new PE type acronym            */
```

```
                        display_PE(PE_ptr,layer,PE_num);
                        highlight_PE(layer,PE_num);

                        break;

        /* ESCAPE key */
        /* set the type back to the state prior to selecting */
        /* this option                                       */
        case -K_ESC   : PE_ptr->type = start_type;
                        /* redisplay the PE label to reflect */
                        /* the original PE type              */
                        display_PE_label(layer,PE_num);

                        /* redisplay the PE itself to reflect */
                        /* the original PE type acronym       */
                        display_PE(PE_ptr,layer,PE_num);
                        highlight_PE(layer,PE_num);

                        done = TRUE;
                        break;

        /* ENTER <Return> key */
        case -K_ENTER : display_PE_label(layer,PE_num);
                        done = TRUE;
                        /* if a new PE type has been selected */
                        /* then clear the PE settings         */
                        if (current_type != start_type)
                        {
                           clear_PE_settings(layer,PE_num);
                           set_vulnerability_current(FALSE);
                        }
                        break;

        default       : bell();
                        break;
    }
}

/* if the new PE type is HEL then */
/* delete this PE from the ASD if */
/* the current threat does not    */
/* have a helicopter              */
if ( (get_PE_cat_spec_ptr(PE_ptr->type)->helicopter) &&
     (get_threat_transport_mode(get_current_threat()) != HELICOPTER) )
{
   PE_ptr->present = FALSE;

   /* if the HEL PE was the only present PE on this layer */
   /* then set and display a bypass indicator             */
   if (number_of_PEs_present(layer) == 0)
   {
      /* remove the PE label window to ensure */
      /* that it does not obscure the bypass  */
      remove_PE_label_window();
```

```
         /* turn the bypass on */
         set_asd_layer_bypass(layer,TRUE);
         display_layer_bypass(layer);

         /* put the PE label window back */
         popup_PE_label_window(layer,PE_num);
      }

      /* redisplay the PE label and PE */
      /* to update the presence state  */
      display_PE_label(layer,PE_num);
      display_PE(PE_ptr,layer,PE_num);
      highlight_PE(layer,PE_num);
   }

   return;
}

/* ------------------------------------------------------------------------ */

savi_modify_element_quit()

{
   /* return a 1 to jump up one menu level */
   return(1);
}

/* ------------------------------------------------------------------------ */

savi_modify_quit()

{
   /* return a 1 to jump up one menu level */
   return(1);
}

/* ********************** end SAVIMODI.C source file ********************** */

/* ************************ SAVIPOP1.C source file ************************ */

#include <wfd.h>
#include <savidefs.h>
#include <stdlib.h>
#include <string.h>

#define MAIN_COMP_WINDOW_TOP 4
#define MAIN_COMP_WINDOW_COL 20
#define MAIN_COMP_WINDOW_HEIGHT 11
#define MAIN_COMP_WINDOW_WIDTH 50

#define COMP_TYPE_WINDOW_TOP 6
#define COMP_TYPE_WINDOW_COL 22
#define COMP_TYPE_WINDOW_HEIGHT 11
```

```
#define COMP_TYPE_WINDOW_WIDTH 50

#define COMP_CHOICE_WINDOW_TOP 8
#define COMP_CHOICE_WINDOW_COL 24
#define COMP_CHOICE_WINDOW_HEIGHT 11
#define COMP_CHOICE_WINDOW_WIDTH 50

#define PANIC_BAR_WINDOW_TOP 10
#define PANIC_BAR_WINDOW_COL 26
#define PANIC_BAR_WINDOW_HEIGHT 11
#define PANIC_BAR_WINDOW_WIDTH 50

#define CLEAR_ALL_WINDOW_TOP 8
#define CLEAR_ALL_WINDOW_COL 26
#define CLEAR_ALL_WINDOW_HEIGHT 8
#define CLEAR_ALL_WINDOW_WIDTH 41

#define CLEAR_PE_WINDOW_TOP 8
#define CLEAR_PE_WINDOW_COL 26
#define CLEAR_PE_WINDOW_HEIGHT 8
#define CLEAR_PE_WINDOW_WIDTH 41

extern WINDOW popup_wn;
extern WINDOW popup_wn_1, popup_wn_2, popup_wn_3;

extern char temp_string_1[];

/* ------------------------------------------------------------------------ */

/* get_main_component_window_ptr returns a pointer to the window in which    */
/* the main component selections begin                                       */

WINDOW *get_main_component_window_ptr()

{
   return(&popup_wn);
}

/* ------------------------------------------------------------------------ */

/* get_comp_type_select_window_ptr returns a pointer to the window in        */
/* which the component type selections are made                              */

WINDOW *get_comp_type_select_window_ptr()

{
   return(&popup_wn_1);
}

/* ------------------------------------------------------------------------ */

/* get_comp_choices_window_ptr returns a pointer to the window in which the  */
/* component choice selections are made                                      */
```

```
WINDOW *get_comp_choices_window_ptr()

{
   return(&popup_wn_2);
}

/* ------------------------------------------------------------------------ */

/* get_threat_attributes_window_ptr returns a pointer to the window in which */
/* the threat attributes are selected                                        */

WINDOW *get_threat_attributes_window_ptr()

{
   return(&popup_wn);
}

/* ------------------------------------------------------------------------ */

/* highlight_row highlights the given row_number within the window that      */
/* wn_ptr points to.                                                         */
/* Note: the given window must be currently on the screen                    */

highlight_row(row_number, wn_ptr)
int row_number;
WINDOW *wn_ptr;

{
   mv_cs(row_number,0,wn_ptr);
   v_natt(LHIGHLITE,ROW,wn_ptr);

   return;
}

/* ------------------------------------------------------------------------ */

/* unhighlight_row unhighlights the given row_number within the window that  */
/* wn_ptr points to.                                                         */
/* Note: the given window must be currently on the screen                    */

unhighlight_row(row_number, wn_ptr)
int row_number;
WINDOW *wn_ptr;

{
   mv_cs(row_number,0,wn_ptr);
   v_natt(LWHITE,ROW,wn_ptr);

   return;
}

/* ------------------------------------------------------------------------ */

/* popup_main_component_window displays the main component selection window  */
```

```
/* for the PE at the given layer and PE_num                                  */

popup_main_component_window(layer,PE_num)
int layer, PE_num;

{
   PROT_ELEM *PE_ptr;

   PE_ptr = get_ASD_PE_ptr(layer,PE_num);

   /* place window on screen */
   mod_wn(MAIN_COMP_WINDOW_TOP,
          MAIN_COMP_WINDOW_COL,
          MAIN_COMP_WINDOW_HEIGHT,
          MAIN_COMP_WINDOW_WIDTH,
          &popup_wn);

   sw_att(LCYAN,&popup_wn);
   set_wn(&popup_wn);

   mv_cs(1,0,&popup_wn);
   v_st("                   between                    ",&popup_wn);
   mv_cs(3,0,&popup_wn);
   v_st("                     and                      ",&popup_wn);

   sw_att(LBLUE,&popup_wn);
   mv_cs(5,0,&popup_wn);
   v_st("DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD",&popup_wn);
   mv_cs(8,0,&popup_wn);
   v_st("DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD",&popup_wn);

   /* display the PE label */
   sw_att(LWHITE,&popup_wn);
   v_plst(0,CENTER_TXT,
          get_PE_cat_spec_ptr(PE_ptr->type)->label,
          &popup_wn);

   /* display the input area label */
   v_plst(2,CENTER_TXT,
          get_full_ASD_area_label(PE_ptr->input_area),
          &popup_wn);

   /* display the output area label */
   v_plst(4,CENTER_TXT,
          get_full_ASD_area_label(PE_ptr->output_area),
          &popup_wn);

   mv_cs(6,0,&popup_wn);
   v_st("   Select Delay Components        ",&popup_wn);
   mv_cs(7,0,&popup_wn);
   v_st("   Select Detection Components    ",&popup_wn);

   return;
}
```

```
/* ------------------------------------------------------------------------ */

/* mark_main_component_window marks the delay and detection component       */
/* selection rows appropriately to indicate if any CLEAR (?) or DIRECT (#)  */
/* component choices have been made for the given PE in the current ASD     */

mark_main_component_window(layer, PE_num)
int layer, PE_num;

{
   if (any_PE_comp_type_choices_are_val(layer, PE_num, DELAY, CLEAR))
      v_plst(6,0,"?",&popup_wn);
   else
      v_plst(6,0," ",&popup_wn);

   if (any_PE_comp_type_choices_are_val(layer, PE_num, DELAY, DIRECT))
      v_plst(6,1,"#",&popup_wn);
   else
      v_plst(6,1," ",&popup_wn);

   if (any_PE_comp_type_choices_are_val(layer, PE_num, NONDET, CLEAR))
      v_plst(7,0,"?",&popup_wn);
   else
      v_plst(7,0," ",&popup_wn);

   if (any_PE_comp_type_choices_are_val(layer, PE_num, NONDET, DIRECT))
      v_plst(7,1,"#",&popup_wn);
   else
      v_plst(7,1," ",&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* remove_main_component_window removes the main component selection window */

remove_main_component_window()

{
   if (window_on_screen(&popup_wn))
      unset_wn(&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* popup_comp_type_select_window displays the component type selection      */
/* window for the given component type (DELAY or NONDET)                    */

popup_comp_type_select_window(comp_type)
int comp_type;
```

```
{
   /* place window on screen */
   mod_wn(COMP_TYPE_WINDOW_TOP,
          COMP_TYPE_WINDOW_COL,
          COMP_TYPE_WINDOW_HEIGHT,
          COMP_TYPE_WINDOW_WIDTH,
          &popup_wn_1);

   sw_att(LWHITE,&popup_wn_1);
   set_wn(&popup_wn_1);

   if (comp_type == DELAY)
   {
      mv_cs(0,0,&popup_wn_1);
      v_st("                DELAY COMPONENTS                ",&popup_wn_1);
   }
   else
   {
      mv_cs(0,0,&popup_wn_1);
      v_st("              DETECTION COMPONENTS              ",&popup_wn_1);
   }

   sw_att(LBLUE,&popup_wn_1);
   mv_cs(1,0,&popup_wn_1);
   v_st("DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD",&popup_wn_1);

   return;
}

/* ---------------------------------------------------------------------- */

/* display_more_indicator displays the graphics in the given window to       */
/* indicate that more rows of data are available in the given direction if   */
/* state is TRUE.  Otherwise, the more display area is cleared.               */
/* The more indicator is displayed on the given row within the given window  */
/* Note: direct must be set to MORE_UP or MORE_DN and the given window must  */
/* already be on the screen                                                   */

display_more_indicator(direct, state, row_num, wn_ptr)
int direct, state, row_num;
WINDOW *wn_ptr;

{
   sw_att(LCYAN,wn_ptr);

   if (direct == MORE_UP)
   {
      if (state)
         v_plst(row_num,CENTER_TXT,
DDDDDDDDDDDDDDDDD",
                  wn_ptr);
      else
         v_plst(row_num,CENTER_TXT,
```

```
                "DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD",
                wn_ptr);
   }
   else
   {
      if (state)
         v_plst(row_num,CENTER_TXT,
                "DDDDDDDDDDDDDDDDDD  MORE  DDDDDDDDDDDDDDDDDD",
                wn_ptr);
      else
         v_plst(row_num,CENTER_TXT,
                "DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD",
                wn_ptr);
   }

   return;
}

/* ------------------------------------------------------------------------ */

/* display_comp_type_selections displays the component type selections in    */
/* the comp_type window for the PE at the given layer and PE_num with the    */
/* given comp type.  The component labels from start_comp to end_comp        */
/* numbers are displayed. Also, the graphics which indicate more components  */
/* above or below the current window are displayed if appropriate by         */
/* comparing start_comp to zero and end_comp to last_comp which is the       */
/* number of the last component in the current list                          */
/* Note: the comp_type window must already be on the screen                  */

display_comp_type_selections(comp_type,layer,PE_num,
                             start_comp,end_comp,last_comp)
int comp_type, layer, PE_num, start_comp, end_comp, last_comp;


{
   int row_num, comp_num, last_row,
       done, delay_comp_type, nondet_comp_type, PE_type;
   char mark_char, letter_char;
   PROT_ELEM *PE_ptr;
   PE_SPECS *PE_spec_ptr;

   PE_ptr = get_ASD_PE_ptr(layer,PE_num);
   PE_type = PE_ptr->type;
   PE_spec_ptr = get_PE_cat_spec_ptr(PE_type);

   sw_att(LWHITE,&popup_wn_1);
   row_num = 3;
   comp_num = start_comp;
   done = FALSE;

   while (!done)
   {
      /* mark clear components with a ? and direct settings with a # */
      switch (get_comp_choice_num(comp_type, comp_num, layer, PE_num))
```

```
{
   case CLEAR  : mark_char = '?';
                 break;

   case DIRECT : mark_char = '#';
                 break;

       default : mark_char = ' ';
                 break;
}

/* prefix component types with capital letters */
letter_char = 'A' + comp_num;

if (comp_type == DELAY)
{
   if (comp_num > end_comp)
      done = TRUE;
   else
   {
      delay_comp_type = PE_spec_ptr->del_comp_list[comp_num].comptype;

      /* create the delay component type string */
      switch (PE_spec_ptr->del_comp_list[comp_num].location)
      {
      case  INPUT_SEG : sprintf(temp_string_1,"%c %c) INPUT %-34s ",
                        mark_char,
                        letter_char,
                        get_delay_comp_spec_ptr(delay_comp_type)->label);
                        break;

      case OUTPUT_SEG : sprintf(temp_string_1,"%c %c) OUTPUT %-34s",
                        mark_char,
                        letter_char,
                        get_delay_comp_spec_ptr(delay_comp_type)->label);
                        break;

      case EITHER_SEG : sprintf(temp_string_1,"%c %c) %-34s       ",
                        mark_char,
                        letter_char,
                        get_delay_comp_spec_ptr(delay_comp_type)->label);
                        break;
      }

      /* output delay component type to window */
      v_plst(row_num,0,temp_string_1,&popup_wn_1);

      comp_num++;
      row_num++;

   }
}
else
{
```

```
      if (comp_num > end_comp)
         done = TRUE;
      else
      {
         nondet_comp_type = PE_spec_ptr->nondet_comp_list[comp_num].
                                          comptype;

         /* create the detection component type string */
         switch (PE_spec_ptr->nondet_comp_list[comp_num].location)
         {
         case  INPUT_SEG : sprintf(temp_string_1,"%c %c) INPUT %-34s ",
                           mark_char,
                           letter_char,
                           get_nondet_comp_spec_ptr(nondet_comp_type)->
                                          label);
                           break;

         case OUTPUT_SEG : sprintf(temp_string_1,"%c %c) OUTPUT %-34s",
                           mark_char,
                           letter_char,
                           get_nondet_comp_spec_ptr(nondet_comp_type)->
                                          label);
                           break;

         case EITHER_SEG : sprintf(temp_string_1,"%c %c) %-34s       ",
                           mark_char,
                           letter_char,
                           get_nondet_comp_spec_ptr(nondet_comp_type)->
                                          label);
                           break;
         }

         /* output detection component type to window */
         v_plst(row_num,0,temp_string_1,&popup_wn_1);

         comp_num++;
         row_num++;
      }

   }

}

/* clear the remaining rows of the window if any exist */
last_row = COMP_TYPE_WINDOW_HEIGHT - 4;

while (row_num <= last_row)
{
   v_plst(row_num,0,"                                                    ",
          &popup_wn_1);
   row_num++;
}

/* display the correct state 'more' indicators */
```

```
   display_more_indicator(MORE_UP,(start_comp > 0),2,&popup_wn_1);
   display_more_indicator(MORE_DN,(end_comp < last_comp),last_row + 1,
                          &popup_wn_1);

   return;
}

/* ------------------------------------------------------------------- */

/* display_no_comp_type_selections displays a message in the component type  */
/* selection window indicating that no components of the given type exist in */
/* the definition of a particular PE                                         */

display_no_comp_type_selections(comp_type)
int comp_type;

{
   sw_att(LWHITE,&popup_wn_1);

   if (comp_type == DELAY)
   {
      v_plst(4,0,"     No Delay Components Exist Within the     ",
             &popup_wn_1);
      v_plst(5,0,"    Definition of this Protection Element     ",
             &popup_wn_1);
   }
   else
   {
      v_plst(4,0,"   No Detection Components Exist Within the   ",
             &popup_wn_1);
      v_plst(5,0,"    Definition of this Protection Element     ",
             &popup_wn_1);
   }

   return;
}

/* ------------------------------------------------------------------- */

/* remove_comp_type_select_window removes the component type selection       */
/* window                                                                    */

remove_comp_type_select_window()

{
   if (window_on_screen(&popup_wn_1))
      unset_wn(&popup_wn_1);

   return;
}

/* ------------------------------------------------------------------- */

/* popup_comp_choices_window displays the component choices window for the   */
```

```
/* given component type (DELAY or NONDET) for the given component (comp_num) */
/* within the appropriate component list for the given PE_type                */

popup_comp_choices_window(comp_type,PE_type,comp_num)
int comp_type, PE_type, comp_num;

{
   int delay_comp_type, nondet_comp_type;
   PE_SPECS *PE_spec_ptr;

   /* place window on screen */
   mod_wn(COMP_CHOICE_WINDOW_TOP,
          COMP_CHOICE_WINDOW_COL,
          COMP_CHOICE_WINDOW_HEIGHT,
          COMP_CHOICE_WINDOW_WIDTH,
          &popup_wn_2);

   PE_spec_ptr = get_PE_cat_spec_ptr(PE_type);

   sw_att(LWHITE,&popup_wn_2);
   set_wn(&popup_wn_2);

   if (comp_type == DELAY)
   {
      delay_comp_type = PE_spec_ptr->del_comp_list[comp_num].comptype;

      /* create the delay component type string */
      switch (PE_spec_ptr->del_comp_list[comp_num].location)
      {
            case  INPUT_SEG : sprintf(temp_string_1,"INPUT %s",
                              get_delay_comp_spec_ptr(delay_comp_type)->label);
                              break;

            case OUTPUT_SEG : sprintf(temp_string_1,"OUTPUT %s",
                              get_delay_comp_spec_ptr(delay_comp_type)->label);
                              break;

            case EITHER_SEG : sprintf(temp_string_1,"%s",
                              get_delay_comp_spec_ptr(delay_comp_type)->label);
                              break;
      }
   }
   else
   {
      nondet_comp_type = PE_spec_ptr->nondet_comp_list[comp_num].comptype;

      /* create the detection component type string */
      switch (PE_spec_ptr->nondet_comp_list[comp_num].location)
      {
          case  INPUT_SEG : sprintf(temp_string_1,"INPUT %s",
                            get_nondet_comp_spec_ptr(nondet_comp_type)->label);
                            break;

          case OUTPUT_SEG : sprintf(temp_string_1,"OUTPUT %s",
```

```
                         get_nondet_comp_spec_ptr(nondet_comp_type)->label);
                         break;

      case EITHER_SEG : sprintf(temp_string_1,"%s",
                         get_nondet_comp_spec_ptr(nondet_comp_type)->label);
                         break;
      }
   }

   /* output component name to window */
   v_plst(0,CENTER_TXT,temp_string_1,&popup_wn_2);

   sw_att(LBLUE,&popup_wn_2);
   mv_cs(1,0,&popup_wn_2);
   v_st("DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD",&popup_wn_2);

   return;
}

/* ------------------------------------------------------------------------ */

display_comp_choices(comp_type,PE_type,comp_num,
                     start_choice,end_choice,last_choice,
                     layer, PE_num)
int comp_type, PE_type, comp_num, start_choice, end_choice, last_choice,
    layer,PE_num;

{
   PROT_ELEM *PE_ptr;
   int row_num, choice_num, last_row, done;
   char letter_char;
   COMP_SPECS *comp_spec_ptr;
   float val;

   PE_ptr = get_ASD_PE_ptr(layer,PE_num);

   if (comp_type == DELAY)
      comp_spec_ptr = get_delay_comp_spec_ptr(get_PE_cat_spec_ptr(PE_type)->
                                              del_comp_list[comp_num].
                                              comptype);
   else
      comp_spec_ptr = get_nondet_comp_spec_ptr(get_PE_cat_spec_ptr(PE_type)->
                                               nondet_comp_list[comp_num].
                                               comptype);

   sw_att(LWHITE,&popup_wn_2);
   row_num = 3;
   choice_num = start_choice;
   done = FALSE;

   while (!done)
   {
      if (choice_num > end_choice)
         done = TRUE;
```

```
else
{
   if (choice_num == 0)
   {
      sprintf(temp_string_1,"  %-44s",
             "CLEAR SETTING (?)");
   }
   else if (choice_num == last_choice)
   {
      if (comp_type == DELAY)
      {
         val = PE_ptr->del_comp_settings[comp_num].direct_setting;
         sprintf(temp_string_1,"  %-37s   %4.0f",
                "DIRECT SETTING (#)",val);
      }
      else
      {
         val = PE_ptr->nondet_comp_settings[comp_num].direct_setting;
         sprintf(temp_string_1,"  %-37s  %5.3f",
                "DIRECT SETTING (#)",invert_det_nondet(val));
      }
   }
   else
   {
      letter_char = 'a' + choice_num - 1;
      val = comp_spec_ptr->threat_values_table[choice_num-1]
                                              [get_current_threat()];
      if (comp_type == DELAY  )
      {
         if (val == INFINITY)
            sprintf(temp_string_1,"  %c) %-34s    N/A",
               letter_char,
               comp_spec_ptr->comp_choice_labels[choice_num-1]);
         else
            sprintf(temp_string_1,"  %c) %-34s   %4.0f",
               letter_char,
               comp_spec_ptr->comp_choice_labels[choice_num-1],
               val);
      }
      else
      {
         if (val == 0.0)
            sprintf(temp_string_1,"  %c) %-34s    N/A",
               letter_char,
               comp_spec_ptr->comp_choice_labels[choice_num-1]);
         else
            sprintf(temp_string_1,"  %c) %-34s  %5.3f",
               letter_char,
               comp_spec_ptr->comp_choice_labels[choice_num-1],
               invert_det_nondet(val));
      }
   }

   v_plst(row_num,0,temp_string_1,&popup_wn_2);
```

```
            choice_num++;
            row_num++;
         }
      }

      /* clear the remaining rows of the window if any exist */
      last_row = COMP_CHOICE_WINDOW_HEIGHT - 4;

      while (row_num <= last_row)
      {
         v_plst(row_num,0,"                                                    ",
                &popup_wn_2);
         row_num++;
      }

      /* display the correct state 'more' indicators */
      display_more_indicator(MORE_UP,(start_choice > 0),2,&popup_wn_2);
      display_more_indicator(MORE_DN,(end_choice < last_choice),last_row + 1,
                                     &popup_wn_2);

      return;
}

/* ------------------------------------------------------------------------ */

/* mark_component_choice marks the given choice row in the component choices */
/* window                                                                    */

mark_component_choice(choice_setting, start_choice, end_choice)
int choice_setting, start_choice, end_choice;

{
      /* if the current choice setting is not within the range of choices */
      /* which are on the screen then there is no need to mark the choice */
      if ((choice_setting < start_choice) || (choice_setting > end_choice))
         return;
      else
      {
         sw_att(LWHITE,&popup_wn_2);
         mv_cs(3+choice_setting-start_choice,0,&popup_wn_2);
         v_st("",&popup_wn_2);
      }

      return;
}

/* ------------------------------------------------------------------------ */

/* remove_comp_choices_window removes the component choices window           */

remove_comp_choices_window()

{
```

```
   if (window_on_screen(&popup_wn_2))
      unset_wn(&popup_wn_2);

   return;
}

/* ------------------------------------------------------------------------ */

/* popup_panic_bar_setting_window displays a pop-up window for selecting the */
/* panic bar setting for a delay component                                    */

popup_panic_bar_setting_window()

{
   /* place panic_bar window on screen */
   mod_wn(PANIC_BAR_WINDOW_TOP,
          PANIC_BAR_WINDOW_COL,
          PANIC_BAR_WINDOW_HEIGHT,
          PANIC_BAR_WINDOW_WIDTH,
          &popup_wn_3);

   sw_att(LWHITE,&popup_wn_3);
   set_wn(&popup_wn_3);

   /* display the window title */
   v_plst(0,CENTER_TXT,"PANIC BAR SETTING",&popup_wn_3);
   v_plst(2,CENTER_TXT,"Does This Delay Component Have a Panic Bar",
          &popup_wn_3);
   v_plst(3,CENTER_TXT,"D or D",
          &popup_wn_3);
   v_plst(4,CENTER_TXT,"Allow Free Passage on Exit ?",
          &popup_wn_3);

   sw_att(LBLUE,&popup_wn_3);
   v_plst(1,0,"DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD",
          &popup_wn_3);

   sw_att(LWHITE,&popup_wn_3);
   v_plst(6,0,"  NO - Passage Restricted on Exit              ",&popup_wn_3);
   v_plst(7,0,"  YES - Free Passage Allowed on Exit           ",&popup_wn_3);

   sw_att(LCYAN,&popup_wn_3);
   v_plst(5,0,"DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD",
          &popup_wn_3);
   v_plst(8,0,"DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD",
          &popup_wn_3);

   return;
}

/* ------------------------------------------------------------------------ */

/* highlight_panic_bar_setting highlights the given panic bar setting in the */
/* panic bar setting window                                                   */
```

```
/* panic_bar must be TRUE or FALSE                                          */

highlight_panic_bar_setting(panic_bar)
int panic_bar;

{
   if (panic_bar)
      highlight_row(7,&popup_wn_3);
   else
      highlight_row(6,&popup_wn_3);

   return;
}

/* ------------------------------------------------------------------------ */

/* unhighlight_panic_bar_setting unhighlights the given panic bar setting in */
/* the panic bar setting window                                             */
/* panic_bar must be TRUE or FALSE                                          */

unhighlight_panic_bar_setting(panic_bar)
int panic_bar;

{
   if (panic_bar)
      unhighlight_row(7,&popup_wn_3);
   else
      unhighlight_row(6,&popup_wn_3);

   return;
}

/* ------------------------------------------------------------------------ */

/* mark_panic_bar marks the current panic_bar setting in the panic_bar      */
/* setting window                                                           */
/* panic_bar must be TRUE or FALSE                                          */

mark_panic_bar(panic_bar)
int panic_bar;

{
   sw_att(LWHITE,&popup_wn_3);
   if (panic_bar)
      v_plst(7,0,"",&popup_wn_3);
   else
      v_plst(6,0,"",&popup_wn_3);

   return;
}

/* ------------------------------------------------------------------------ */

/* unmark_panic_bar unmarks the current panic_bar setting in the panic_bar  */
```

```
/* setting window                                                          */
/* panic_bar must be TRUE or FALSE                                         */

unmark_panic_bar(panic_bar)
int panic_bar;

{
   sw_att(LWHITE,&popup_wn_3);
   if (panic_bar)
      v_plst(7,0," ",&popup_wn_3);
   else
      v_plst(6,0," ",&popup_wn_3);

   return;
}

/* ----------------------------------------------------------------------- */

/* remove_panic_bar_setting_window removes the panic_bar_setting window    */

remove_panic_bar_setting_window()

{
   if (window_on_screen(&popup_wn_3))
      unset_wn(&popup_wn_3);

   return;
}

/* ----------------------------------------------------------------------- */

/* popup_clear_PE_setting_window displays a popup window to allow the user */
/* to confirm whether to clear all settings within the given PE in the     */
/* current ASD                                                             */

popup_clear_PE_setting_window(layer,PE_num)
int layer, PE_num;

{
   PROT_ELEM *PE_ptr;

   PE_ptr = get_ASD_PE_ptr(layer,PE_num);

   /* place window on screen */
   mod_wn(CLEAR_PE_WINDOW_TOP,
          CLEAR_PE_WINDOW_COL,
          CLEAR_PE_WINDOW_HEIGHT,
          CLEAR_PE_WINDOW_WIDTH,
          &popup_wn);

   sw_att(LBLUE,&popup_wn);
   set_wn(&popup_wn);

   /* display the window text */
```

```
   v_plst(2,0,"          DDDDDDDDDDDDDDD          ",&popup_wn);
   v_plst(3,0,"          ZDDDD?   ZDDDDD?         ",&popup_wn);
   v_plst(4,0,"          3    3   3     3         ",&popup_wn);
   v_plst(5,0,"          @DDDDY   @DDDDDY         ",&popup_wn);

   sw_att(LWHITE,&popup_wn);
   /* if this PE is the Target Task PE then use 'the' to indicate */
   /* that only one Target Task PE exists in any ASD              */
   if (layer == MAX_LAYERS_PER_ASD)
      v_plst(0,1,"Clear All Component Settings in the",&popup_wn);
   else
      v_plst(0,0,"Clear All Component Settings in this",&popup_wn);
   sprintf(temp_string_1,"%s ?",get_PE_cat_spec_ptr(PE_ptr->type)->label);
   v_plst(1,CENTER_TXT,temp_string_1,&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* highlight_clear_PE_setting highlights the options YES or NO based on the  */
/* given state. If state is TRUE the YES option is highlighted within the    */
/* clear_PE_setting window, otherwise the NO option is highlighted.          */

highlight_clear_PE_setting(state)
int state;

{
   /* if state is TRUE then unhighlight NO and highlight YES */
   if (state)
   {
      sw_att(LWHITE,&popup_wn);
      v_plst(4,11," NO ",&popup_wn);
      sw_att(LHIGHLITE,&popup_wn);
      v_plst(4,20," YES ",&popup_wn);
   }
   /* otherwise unhighlight YES and highlight NO */
   else
   {
      sw_att(LWHITE,&popup_wn);
      v_plst(4,20," YES ",&popup_wn);
      sw_att(LHIGHLITE,&popup_wn);
      v_plst(4,11," NO ",&popup_wn);
   }

   return;
}

/* ------------------------------------------------------------------------ */

/* remove_clear_PE_setting_window removes the clear_PE_setting window        *

remove_clear_PE_setting_window()
```

```
{
   if (window_on_screen(&popup_wn))
      unset_wn(&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* popup_clear_all_setting_window displays a popup window to allow the user  */
/* to confirm whether to clear all PE settings in the current ASD            */

popup_clear_all_setting_window()

{
   /* place window on screen */
   mod_wn(CLEAR_ALL_WINDOW_TOP,
          CLEAR_ALL_WINDOW_COL,
          CLEAR_ALL_WINDOW_HEIGHT,
          CLEAR_ALL_WINDOW_WIDTH,
          &popup_wn);

   sw_att(LBLUE,&popup_wn);
   set_wn(&popup_wn);

   /* display the window text */
   v_plst(2,0,"          DDDDDDDDDDDDDDD          ",&popup_wn);
   v_plst(3,0,"          ZDDDD?   ZDDDDD?         ",&popup_wn);
   v_plst(4,0,"          3    3   3     3         ",&popup_wn);
   v_plst(5,0,"          @DDDDY   @DDDDDY         ",&popup_wn);

   sw_att(LWHITE,&popup_wn);
   v_plst(0,4,"Clear ALL Component Settings",&popup_wn);
   v_plst(1,8,"in the Current ASD ?",&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* highlight_clear_all_setting highlights the options YES or NO based on the */
/* given state. If state is TRUE the YES option is highlighted within the    */
/* clear_all_setting window, otherwise the NO option is highlighted.         */

highlight_clear_all_setting(state)
int state;

{
   /* if state is TRUE then unhighlight NO and highlight YES */
   if (state)
   {
      sw_att(LWHITE,&popup_wn);
      v_plst(4,11," NO ",&popup_wn);
      sw_att(LHIGHLITE,&popup_wn);
```

```
      v_plst(4,20," YES ",&popup_wn);
   }
   /* otherwise unhighlight YES and highlight NO */
   else
   {
      sw_att(LWHITE,&popup_wn);
      v_plst(4,20," YES ",&popup_wn);
      sw_att(LHIGHLITE,&popup_wn);
      v_plst(4,11," NO ",&popup_wn);
   }

   return;
}

/* ------------------------------------------------------------------------ */

/* remove_clear_all_setting_window removes the clear_all_setting window      */

remove_clear_all_setting_window()

{
   if (window_on_screen(&popup_wn))
      unset_wn(&popup_wn);

   return;
}

/* ********************** end SAVIPOP1.C source file ********************** */

/* ************************ SAVIPOP2.C source file ************************ */

#include <wfd.h>
#include <savidefs.h>
#include <stdlib.h>
#include <string.h>

#define AREA_TRANSIT_INPUT_WINDOW_TOP 9
#define AREA_TRANSIT_INPUT_WINDOW_COL 28
#define AREA_TRANSIT_INPUT_WINDOW_HEIGHT 10
#define AREA_TRANSIT_INPUT_WINDOW_WIDTH 38

#define AREA_LABEL_WINDOW_TOP 9
#define AREA_LABEL_WINDOW_COL 32
#define AREA_LABEL_WINDOW_HEIGHT 6
#define AREA_LABEL_WINDOW_WIDTH 29

#define AREA_TRAVERSAL_WINDOW_TOP 7
#define AREA_TRAVERSAL_WINDOW_COL 22
#define AREA_TRAVERSAL_WINDOW_HEIGHT 9
#define AREA_TRAVERSAL_WINDOW_WIDTH 50

#define THREAT_WINDOW_TOP 7
#define THREAT_WINDOW_COL 22
#define THREAT_WINDOW_HEIGHT 9
```

```
#define THREAT_WINDOW_WIDTH 50

#define THREAT_OBJECTIVE_WINDOW_TOP 8
#define THREAT_OBJECTIVE_WINDOW_COL 22
#define THREAT_OBJECTIVE_WINDOW_HEIGHT 8
#define THREAT_OBJECTIVE_WINDOW_WIDTH 50

#define THREAT_STRATEGY_WINDOW_TOP 8
#define THREAT_STRATEGY_WINDOW_COL 22
#define THREAT_STRATEGY_WINDOW_HEIGHT 8
#define THREAT_STRATEGY_WINDOW_WIDTH 50

#define RFT_RANGE_WINDOW_TOP 3
#define RFT_RANGE_WINDOW_COL 33
#define RFT_RANGE_WINDOW_HEIGHT 5
#define RFT_RANGE_WINDOW_WIDTH 27

#define RFT_VALUES_WINDOW_TOP 8
#define RFT_VALUES_WINDOW_COL 33
#define RFT_VALUES_WINDOW_HEIGHT 6
#define RFT_VALUES_WINDOW_WIDTH 27

#define RFT_ERROR_WINDOW_TOP 19
#define RFT_ERROR_WINDOW_COL 26
#define RFT_ERROR_WINDOW_HEIGHT 4
#define RFT_ERROR_WINDOW_WIDTH 41

#define STRANDED_PE_WINDOW_TOP 8
#define STRANDED_PE_WINDOW_COL 25
#define STRANDED_PE_WINDOW_HEIGHT 7
#define STRANDED_PE_WINDOW_WIDTH 43

#define CURRENT_RFT_WINDOW_TOP 6
#define CURRENT_RFT_WINDOW_COL 32
#define CURRENT_RFT_WINDOW_HEIGHT 7
#define CURRENT_RFT_WINDOW_WIDTH 31

#define TURBO_PROGRESS_WINDOW_TOP 9
#define TURBO_PROGRESS_WINDOW_COL 26
#define TURBO_PROGRESS_WINDOW_HEIGHT 6
#define TURBO_PROGRESS_WINDOW_WIDTH 41

#define INSERT_ERROR_WINDOW_TOP 13
#define INSERT_ERROR_WINDOW_COL 21
#define INSERT_ERROR_WINDOW_HEIGHT 4
#define INSERT_ERROR_WINDOW_WIDTH 52

#define SAVE_ASD_OFFER_WINDOW_TOP 7
#define SAVE_ASD_OFFER_WINDOW_COL 31
#define SAVE_ASD_OFFER_WINDOW_HEIGHT 10
#define SAVE_ASD_OFFER_WINDOW_WIDTH 31

#define PE_TRANSIT_INPUT_WINDOW_HEIGHT 14
```

```
#define PE_COPY_WINDOW_HEIGHT 9

extern WINDOW popup_wn, popup_wn_1, popup_wn_2;

extern char temp_string_1[];

int turbo_progress_updated, last_percent_displayed;

/* ------------------------------------------------------------------------ */

/* popup_threat_objective_window displays a popup window to allow the user   */
/* to select the current threat objective: sabotage (ENTRY only analysis)    */
/* or theft (EXIT analysis)                                                  */

popup_threat_objective_window()

{
   /* place window on screen */
   mod_wn(THREAT_OBJECTIVE_WINDOW_TOP,
          THREAT_OBJECTIVE_WINDOW_COL,
          THREAT_OBJECTIVE_WINDOW_HEIGHT,
          THREAT_OBJECTIVE_WINDOW_WIDTH,
          &popup_wn);

   sw_att(LCYAN,&popup_wn);
   set_wn(&popup_wn);

   /* display the window text */
   v_plst(0,CENTER_TXT,"Threat Objective",&popup_wn);

   sw_att(LBLUE,&popup_wn);
   v_plst(1,0,"DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD",
          &popup_wn);

   sw_att(LWHITE,&popup_wn);
   v_plst(3,0,"  ENTRY - Sabotage or Hands-On Theft Analysis ",&popup_wn);
   v_plst(4,0,"  ENTRY/EXIT - Containment Theft Analysis     ",&popup_wn);

   sw_att(LCYAN,&popup_wn);
   v_plst(2,0,"DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD",
          &popup_wn);
   v_plst(5,0,"DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD",
          &popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* highlight_threat_objective highlights the SABOTAGE or THEFT objective     */
/* based on the given direction. If direction is ENTRY then the SABOTAGE     */
/* objective is highlighted, otherwise the THEFT objective is highlighted.   */

highlight_threat_objective(direction)
```

```
int direction;

{
   if (direction == ENTRY)
      highlight_row(3,&popup_wn);
   else
      highlight_row(4,&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* unhighlight_threat_objective unhighlights the SABOTAGE or THEFT objective */
/* based on the given direction. If direction is ENTRY then the SABOTAGE     */
/* objective is unhighlighted, otherwise the THEFT objective                 */

unhighlight_threat_objective(direction)
int direction;

{
   if (direction == ENTRY)
      unhighlight_row(3,&popup_wn);
   else
      unhighlight_row(4,&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* remove_threat_objective_window removes the threat objective window        */

remove_threat_objective_window()

{
   if (window_on_screen(&popup_wn))
      unset_wn(&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* mark_objective marks the current threat objective in the selection window */
/* direction must be ENTRY or EXIT                                           */

mark_objective(direction)
int direction;

{
   sw_att(LWHITE,&popup_wn);
   if (direction == ENTRY)
      v_plst(3,0,"",&popup_wn);
```

```
   else
      v_plst(4,0,"",&popup_wn);

   return;
}

/* ----------------------------------------------------------------------- */

/* unmark_objective unmarks the given threat objective in the selection    */
/* window - direction must be ENTRY or EXIT                                */

unmark_objective(direction)
int direction;

{
   sw_att(LWHITE,&popup_wn);
   if (direction == ENTRY)
      v_plst(3,0," ",&popup_wn);
   else
      v_plst(4,0," ",&popup_wn);

   return;
}

/* ----------------------------------------------------------------------- *

/* popup_threat_strategy_window displays a popup window to allow the user  */
/* to select the current threat strategy: FORCE or MIXED                   */

popup_threat_strategy_window()

{
   /* place window on screen */
   mod_wn(THREAT_STRATEGY_WINDOW_TOP,
          THREAT_STRATEGY_WINDOW_COL,
          THREAT_STRATEGY_WINDOW_HEIGHT,
          THREAT_STRATEGY_WINDOW_WIDTH,
          &popup_wn);

   sw_att(LCYAN,&popup_wn);
   set_wn(&popup_wn);

   /* display the window text */
   v_plst(0,CENTER_TXT,"Threat Tactics",&popup_wn);

   sw_att(LBLUE,&popup_wn);
   v_plst(1,0,"DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD",
          &popup_wn);

   sw_att(LWHITE,&popup_wn);
   v_plst(3,0,"  MIXED - Force & Deceit Intrusion               ",&popup_wn);
   v_plst(4,0,"  FORCE - Force Intrusion Only                   ",&popup_wn);

   sw_att(LCYAN,&popup_wn);
```

```
   v_plst(2,0,"DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD",
          &popup_wn);
   v_plst(5,0,"DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD",
          &popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* highlight_threat_strategy highlights the FORCE or MIXED strategy based   */
/* on the given strategy.                                                   */

highlight_threat_strategy(strategy)
int strategy;

{
   if (strategy == FORCE)
      highlight_row(4,&popup_wn);
   else
      highlight_row(3,&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* unhighlight_threat_strategy unhighlights the FORCE or MIXED strategy     */
/* based on the given strategy.                                             */

unhighlight_threat_strategy(strategy)
int strategy;

{
   if (strategy == FORCE)
      unhighlight_row(4,&popup_wn);
   else
      unhighlight_row(3,&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* remove_threat_strategy_window removes the threat strategy window         */

remove_threat_strategy_window()

{
   if (window_on_screen(&popup_wn))
      unset_wn(&popup_wn);

   return;
}
```

```
/* ------------------------------------------------------------------------ */

/* mark_strategy marks the current threat strategy in the selection window  */
/* strategy must be FORCE or MIXED                                          */

mark_strategy(strategy)
int strategy;

{
   sw_att(LWHITE,&popup_wn);
   if (strategy == FORCE)
      v_plst(4,0,"",&popup_wn);
   else
      v_plst(3,0,"",&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* unmark_strategy unmarks the given threat strategy in the selection window */
/* strategy must be FORCE or MIXED                                          */

unmark_strategy(strategy)
int strategy;

{
   sw_att(LWHITE,&popup_wn);
   if (strategy == FORCE)
      v_plst(4,0," ",&popup_wn);
   else
      v_plst(3,0," ",&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* popup_PE_transit_input_window displays a pop-up window for inputting the  */
/* transit distance for the PE specified by layer and PE_num in the current  */
/* ASD ensuring that the PE box is not obscured by the pop-up window         */
/* If the transit input window is already on the screen it is moved if       */
/* necessary or left alone if it will not obscure the given PE. If it is not */
/* on the screen the window is placed on the screen so as not to obscure the */
/* given PE.                                                                 */

popup_PE_transit_input_window(layer,PE_num)
int layer, PE_num;

{
   int popup_col;

   /* determine which side the transit input window should be on */
```

```
if (PE_on_left(layer,PE_num) ||
    (PE_base_row(layer,PE_num)+1) >
    (POPUP_TOP+PE_TRANSIT_INPUT_WINDOW_HEIGHT-3)
   )
   popup_col = POPUP_RIGHT_COL;
else
   popup_col = POPUP_LEFT_COL;

/* if the transit window is already on the screen in the  */
/* correct place so as not to obscure the given PE then   */
/* return, else remove it from the screen to prepare for  */
/* moving                                                 */
if (window_on_screen(&popup_wn))
   {
      /* set window dimensions to FULL to check left column */
      dim_wn(FULL,&popup_wn);

      /* check if window is already in correct place */
      if (popup_wn.cb == popup_col)
         {
            /* window is already in correct place   */
            /* set window dimensions back to INSIDE */
            /* before returning                     */
            dim_wn(INSIDE,&popup_wn);
            return;
         }
      else
         {
            /* window will obscure PE if not moved  */
            /* set window dimensions back to INSIDE */
            /* before removing window               */
            dim_wn(INSIDE,&popup_wn);
            remove_PE_transit_input_window();
         }
   }

/* place transit input window on screen in correct place */
mod_wn(POPUP_TOP,popup_col,
       PE_TRANSIT_INPUT_WINDOW_HEIGHT,
       POPUP_WIDTH,
       &popup_wn);

sw_att(LCYAN,&popup_wn);
set_wn(&popup_wn);

/* display the transit window titles */
v_plst(0,0,
       "         Distance Across          ",
       &popup_wn);
v_plst(2,0,
       "              between             ",
       &popup_wn);
v_plst(4,0,
       "                 and              ",
```

```
        &popup_wn);
    v_plst(9,0,
        "            Transit Time            ",
        &popup_wn);

    sw_att(LBLUE,&popup_wn);
    v_plst(6,0,
        "DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD",
        &popup_wn);

    v_plst(8,0,
        "          DDDDDDDDDDDDDDDD          ",
        &popup_wn);

    return;
}

/* ------------------------------------------------------------------------ */

/* display_PE_transit_val displays the transit distance setting for the PE   */
/* specified by layer and PE_num within the current ASD                      */

display_PE_transit_val(layer,PE_num)
int layer,PE_num;

{
    PROT_ELEM *PE_ptr;
    int dist, transit_time, transit_min, transit_sec;

    PE_ptr = get_ASD_PE_ptr(layer,PE_num);
    dist = PE_ptr->transit_dist;
    transit_time = (int)element_transit_time(layer,PE_num);
    partition_seconds(transit_time,&transit_min,&transit_sec);

    /* clear the PE label field */
    mv_cs(1,0,&popup_wn);
    v_qch(' ',POPUP_WIDTH-4,&popup_wn);

    /* display the PE label */
    sw_att(LWHITE,&popup_wn);
    v_plst(1,CENTER_TXT,
        get_PE_cat_spec_ptr(PE_ptr->type)->label,
        &popup_wn);

    /* clear the input area field */
    mv_cs(3,0,&popup_wn);
    v_qch(' ',POPUP_WIDTH-4,&popup_wn);

    /* display the input area label */
    v_plst(3,CENTER_TXT,
        get_full_ASD_area_label(PE_ptr->input_area),
        &popup_wn);

    /* clear the output area field */
```

```
   mv_cs(5,0,&popup_wn);
   v_qch(' ',POPUP_WIDTH-4,&popup_wn);

   /* display the output area label */
   v_plst(5,CENTER_TXT,
          get_full_ASD_area_label(PE_ptr->output_area),
          &popup_wn);

   /* display the current transit distance value */
   sprintf(temp_string_1,"%4d",dist);
   v_plst(7,11,temp_string_1,&popup_wn);

   /* display the meters label correctly for 1 meter */
   sw_att(LCYAN,&popup_wn);
   if (dist == 1)
      v_plst(7,16,
             "meter ",
             &popup_wn);
   else
      v_plst(7,16,
             "meters",
             &popup_wn);

   /* display the current transit time across this PE */
   sw_att(LWHITE,&popup_wn);
   sprintf(temp_string_1,"%2d",transit_min);
   v_plst(10,12,temp_string_1,&popup_wn);
   sprintf(temp_string_1,"%2d",transit_sec);
   v_plst(11,12,temp_string_1,&popup_wn);

   sw_att(LCYAN,&popup_wn);
   if (transit_min == 1)
      v_plst(10,15,"minute ",&popup_wn);
   else
      v_plst(10,15,"minutes",&popup_wn);

   if (transit_sec == 1)
      v_plst(11,15,"second ",&popup_wn);
   else
      v_plst(11,15,"seconds",&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* remove_PE_transit_input_window removes the transit input pop-up window    */

remove_PE_transit_input_window()

{
   if (window_on_screen(&popup_wn))
      unset_wn(&popup_wn);
```

```
   return;
}

/* ------------------------------------------------------------------------ */

/* popup_area_transit_input_window displays a pop-up window for inputting    */
/* the transit distance for any area in the current ASD.                      */

popup_area_transit_input_window()

{
   /* if the area transit window is already on the screen then return  */
   if (window_on_screen(&popup_wn))
   {
      return;
   }

   /* place transit input window on screen */
   mod_wn(AREA_TRANSIT_INPUT_WINDOW_TOP,
          AREA_TRANSIT_INPUT_WINDOW_COL,
          AREA_TRANSIT_INPUT_WINDOW_HEIGHT,
          AREA_TRANSIT_INPUT_WINDOW_WIDTH,
          &popup_wn);

   sw_att(LCYAN,&popup_wn);
   set_wn(&popup_wn);

   /* display the transit window titles */
   v_plst(0,0,
          "          Distance Across          ",
          &popup_wn);
   v_plst(5,0,
          "            Transit Time           ",
          &popup_wn);


   sw_att(LBLUE,&popup_wn);
   v_plst(2,0,
          "DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD",
          &popup_wn);
   v_plst(4,0,
          "          DDDDDDDDDDDDDDDD          ",
          &popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* display_area_transit_val displays the transit distance setting for the    */
/* given area

display_area_transit_val(area_num)
int area_num;
```

```
{
   int dist, transit_time, transit_min, transit_sec;

   dist = get_ASD_area_dist(area_num);
   transit_time = (int)area_transit_time(area_num);
   partition_seconds(transit_time,&transit_min,&transit_sec);

   /* clear the area label field */
   mv_cs(1,0,&popup_wn);
   v_gch(' ',AREA_TRANSIT_INPUT_WINDOW_WIDTH-4,&popup_wn);

   /* display the area label */
   sw_att(LWHITE,&popup_wn);
   v_plst(1,CENTER_TXT,
          get_full_ASD_area_label(area_num),
          &popup_wn);

   /* display the current transit distance value */
   sprintf(temp_string_1,"%4d",dist);
   v_plst(3,11,temp_string_1,&popup_wn);

   /* display the meters label correctly for 1 meter */
   sw_att(LCYAN,&popup_wn);
   if (dist == 1)
      v_plst(3,16,
             "meter ",
             &popup_wn);
   else
      v_plst(3,16,
             "meters",
             &popup_wn);

   /* display the current transit time across this area */
   sw_att(LWHITE,&popup_wn);
   sprintf(temp_string_1,"%2d",transit_min);
   v_plst(6,12,temp_string_1,&popup_wn);
   sprintf(temp_string_1,"%2d",transit_sec);
   v_plst(7,12,temp_string_1,&popup_wn);

   sw_att(LCYAN,&popup_wn);
   if (transit_min == 1)
      v_plst(6,15,"minute ",&popup_wn);
   else
      v_plst(6,15,"minutes",&popup_wn);

   if (transit_sec == 1)
      v_plst(7,15,"second ",&popup_wn);
   else
      v_plst(7,15,"seconds",&popup_wn);

   return;
}
```

```
/* ------------------------------------------------------------------------ */

/* remove_area_transit_input_window removes the area transit input window    */

remove_area_transit_input_window()

{
   if (window_on_screen(&popup_wn))
      unset_wn(&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* popup_area_traversal_window displays a pop-up window for selecting the    */
/* traversal setting for the given area                                      */
/* If the window is already on the screen, then the area label is updated    */
/* without redisplaying the window                                           */

popup_area_traversal_window(area_num)
int area_num;

{
   if (!window_on_screen(&popup_wn))
   {
      /* place traversal window on screen */
      mod_wn(AREA_TRAVERSAL_WINDOW_TOP,
             AREA_TRAVERSAL_WINDOW_COL,
             AREA_TRAVERSAL_WINDOW_HEIGHT,
             AREA_TRAVERSAL_WINDOW_WIDTH,
             &popup_wn);

      sw_att(LCYAN,&popup_wn);
      set_wn(&popup_wn);

      /* display the window title */
      v_plst(0,CENTER_TXT,"Traversal Setting",&popup_wn);

      sw_att(LBLUE,&popup_wn);
      v_plst(2,0,"DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD",
             &popup_wn);

      sw_att(LWHITE,&popup_wn);
      v_plst(4,0,"  INTERIOR - Foot Traversal Only                ",&popup_wn);
      v_plst(5,0,"  EXTERIOR - Vehicle Traversal Possible         ",&popup_wn);

      sw_att(LCYAN,&popup_wn);
      v_plst(3,0,"DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD",
             &popup_wn);
      v_plst(6,0,"DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD",
             &popup_wn);
   }
```

```
   sw_att(LWHITE,&popup_wn);

   /* clear the area label field */
   mv_cs(1,0,&popup_wn);
   v_ach(' ',AREA_TRAVERSAL_WINDOW_WIDTH-4,&popup_wn);

   /* display the area label */
   v_plst(1,CENTER_TXT,
          get_full_ASD_area_label(area_num),
          &popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* highlight_traversal_setting highlights the given traversal setting in the */
/* traversal setting window                                                  */
/* traversal_setting must be INTERIOR or EXTERIOR                            */

highlight_traversal_setting(traversal_setting)
int traversal_setting;

{
   if (traversal_setting == INTERIOR)
      highlight_row(4,&popup_wn);
   else
      highlight_row(5,&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* unhighlight_traversal_setting highlights the given traversal setting in   */
/* the traversal setting window                                              */
/* traversal_setting must be INTERIOR or EXTERIOR                            */

unhighlight_traversal_setting(traversal_setting)
int traversal_setting;

{
   if (traversal_setting == INTERIOR)
      unhighlight_row(4,&popup_wn);
   else
      unhighlight_row(5,&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* mark_traversal marks the current traversal setting in the area traversal  */
/* setting window                                                            */
```

```
/* traversal_setting must be INTERIOR or EXTERIOR                          */

mark_traversal(traversal_setting)
int traversal_setting;

{
   sw_att(LWHITE,&popup_wn);
   if (traversal_setting == INTERIOR)
      v_plst(4,0,"",&popup_wn);
   else
      v_plst(5,0,"",&popup_wn);

   return;
}

/* ---------------------------------------------------------------------- */

/* unmark_traversal unmarks the given traversal setting in the traversal  */
/* setting window                                                         */
/* traversal_setting must be INTERIOR or EXTERIOR                         */

unmark_traversal(traversal_setting)
int traversal_setting;

{
   sw_att(LWHITE,&popup_wn);
   if (traversal_setting == INTERIOR)
      v_plst(4,0," ",&popup_wn);
   else
      v_plst(5,0," ",&popup_wn);

   return;
}

/* ---------------------------------------------------------------------- */

/* remove_area_traversal_window removes the area traversal setting window */

remove_area_traversal_window()

{
   if (window_on_screen(&popup_wn))
      unset_wn(&popup_wn);

   return;
}

/* ---------------------------------------------------------------------- */

/* popup_area_label_window displays a pop-up window for editing area labels */

popup_area_label_window()

{
```

```
   /* place area label window on screen */
   mod_wn(AREA_LABEL_WINDOW_TOP,
          AREA_LABEL_WINDOW_COL,
          AREA_LABEL_WINDOW_HEIGHT,
          AREA_LABEL_WINDOW_WIDTH,
          &popup_wn);

   sw_att(LCYAN,&popup_wn);
   set_wn(&popup_wn);

   sw_att(LBLUE,&popup_wn);
   v_plst(1,0,"DDDDDDDDDDDDDDDDDDDDDDDDD",
          &popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* update_area_label_window updates the area label information in the area   */
/* label editing window for the given area                                   */

update_area_label_window(area_num)
int area_num;

{
   char area_label_line[MAX_HALF_AREA_LABEL_LEN];

   /* create the cardinal area label string */
   /* area number zero is cardinal 'first'  */
   sprintf(temp_string_1,"%sArea Label",cardinal_number(area_num));

   sw_att(LCYAN,&popup_wn);

   /* clear the cardinal area label field */
   mv_cs(0,0,&popup_wn);
   v_qch(' ',AREA_LABEL_WINDOW_WIDTH-4,&popup_wn);

   /* display the cardinal area label field */
   v_plst(0,CENTER_TXT,temp_string_1,&popup_wn);

   sw_att(LWHITE,&popup_wn);

   /* clear the first and second area label lines */
   mv_cs(2,0,&popup_wn);
   v_qch(' ',AREA_LABEL_WINDOW_WIDTH-4,&popup_wn);
   mv_cs(3,0,&popup_wn);
   v_qch(' ',AREA_LABEL_WINDOW_WIDTH-4,&popup_wn);

   /* display the first and second area label lines */
   sprintf(area_label_line,"%s",get_ASD_area_label_first_line(area_num));
   v_plst(2,6,area_label_line,&popup_wn);

   sprintf(area_label_line,"%s",get_ASD_area_label_second_line(area_num));
```

```
   v_plst(3,6,area_label_line,&popup_wn);

  return;
}

/* ----------------------------------------------------------------------- */

/* remove_area_label_window removes the area label editing window          */

remove_area_label_window()

{
   if (window_on_screen(&popup_wn))
      unset_wn(&popup_wn);

   return;
}

/* ----------------------------------------------------------------------- */

/* popup_threat_attributes_window displays the threat selection window      */

popup_threat_attributes_window()

{
   /* place window on screen */
   mod_wn(THREAT_WINDOW_TOP,
          THREAT_WINDOW_COL,
          THREAT_WINDOW_HEIGHT,
          THREAT_WINDOW_WIDTH,
          &popup_wn);

   sw_att(LCYAN,&popup_wn);
   set_wn(&popup_wn);

   v_plst(0,CENTER_TXT,
          "Threat Attributes",
          &popup_wn);

   sw_att(LBLUE,&popup_wn);
   v_plst(1,0,"DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD",&popup_wn);

   return;
}

/* ----------------------------------------------------------------------- */

display_threat_attributes(start_threat,end_threat,last_threat)
int start_threat, end_threat, last_threat;

{
   int row_num, threat_num, last_row, done;

   sw_att(LWHITE,&popup_wn);
```

```
   row_num = 3;
   threat_num = start_threat;
   done = FALSE;

   while (!done)
   {
      if (threat_num > end_threat)
         done = TRUE;
      else
      {
         sprintf(temp_string_1,"  %-44s",get_threat_label(threat_num));
         v_plst(row_num,0,temp_string_1,&popup_wn);

         threat_num++;
         row_num++;
      }
   }

   /* clear the remaining rows of the window if any exist */
   last_row = THREAT_WINDOW_HEIGHT - 4;

   while (row_num <= last_row)
   {
      v_plst(row_num,0,"                                                    ",
             &popup_wn);
      row_num++;
   }

   /* display the correct state 'more' indicators */
   display_more_indicator(MORE_UP,(start_threat > 0),2,&popup_wn);
   display_more_indicator(MORE_DN,(end_threat < last_threat),last_row + 1,
                                  &popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* mark_threat marks the given choice row in the threat attributes window    */

mark_threat(choice_setting, start_choice, end_choice)
int choice_setting, start_choice, end_choice;

{
   /* if the current choice setting is not within the range of choices */
   /* which are on the screen then there is no need to mark the choice */
   if ((choice_setting < start_choice) || (choice_setting > end_choice))
      return;
   else
   {
      sw_att(LWHITE,&popup_wn);
      mv_cs(3+choice_setting-start_choice,0,&popup_wn);
      v_st("",&popup_wn);
   }
```

```
   return;
}

/* ------------------------------------------------------------------------ */

/* remove_threat_attributes_window removes the threat selection window      */

remove_threat_attributes_window()

{
   if (window_on_screen(&popup_wn))
      unset_wn(&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* popup_rft_range_window displays a popup window to display the number of  */
/* rfts in the current rft range                                            */

popup_rft_range_window()

{
   /* place rft range window on screen */
   mod_wn(RFT_RANGE_WINDOW_TOP,
          RFT_RANGE_WINDOW_COL,
          RFT_RANGE_WINDOW_HEIGHT,
          RFT_RANGE_WINDOW_WIDTH,
          &popup_wn);

   sw_att(LCYAN,&popup_wn);
   set_wn(&popup_wn);

   /* display the rft range window text */
   v_plst(0,0,"Number of RFTs in Range",&popup_wn);

   sw_att(LBLUE,&popup_wn);
   v_plst(1,0,"DDDDDDDDDDDDDDDDDDDDD",&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* display_num_rfts displays the given number of rfts in the rft range      */
/* window  Note: the rft range window must be currently displayed           */

display_num_rfts(num_rfts)
int num_rfts;

{
   sw_att(LWHITE,&popup_wn);
```

```
   sprintf(temp_string_1,"%2d",num_rfts);
   v_plst(2,0,temp_string_1,&popup_wn);

   sw_att(LCYAN,&popup_wn);
   if (num_rfts == 1)
      v_plst(2,3,"Response Force Time ",&popup_wn);
   else
      v_plst(2,3,"Response Force Times",&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* highlight_num_rfts highlights the number of rfts in the rft range window  */
/* Note: the rft range window must be currently on the screen                 */

highlight_num_rfts()

{
   /* highlight the number of rfts within the rft range window */
   mv_cs(2,0,&popup_wn);
   v_natt(LHIGHLITE,CH,&popup_wn);
   mv_cs(2,1,&popup_wn);
   v_natt(LHIGHLITE,CH,&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* remove_rft_range_window removes the pop-up rft range window               */

remove_rft_range_window()

{
   if (window_on_screen(&popup_wn))
      unset_wn(&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* popup_rft_values_window displays a popup window to display the limits of  */
/* the current rft range                                                     */

popup_rft_values_window()

{
   int correction;

   /* correct the window height to fit the number of RFTs */
   if (get_num_rfts() == 1)
```

```
      correction = 0;
   else
      correction = 5;

   /* place rft values window on screen */
   mod_wn(RFT_VALUES_WINDOW_TOP,
          RFT_VALUES_WINDOW_COL,
          RFT_VALUES_WINDOW_HEIGHT + correction,
          RFT_VALUES_WINDOW_WIDTH,
          &popup_wn_1);

   sw_att(LWHITE,&popup_wn_1);
   set_wn(&popup_wn_1);

   /* display the rft values window */
   sw_att(LBLUE,&popup_wn_1);
   v_plst(1,0,"DDDDDDDDDDDDDDDDDDDDD",&popup_wn_1);

   return;
}

/* ------------------------------------------------------------------------ */

/* display_rft_values displays the rft range limits in the rft values window */
/* If only one rft is in the current range it is displayed alone.            */
/* Note: the rft values window must be currently displayed                   */

display_rft_values()

{
   int num_rfts, min_rft, max_rft,
       min_rft_minutes, min_rft_seconds,
       max_rft_minutes, max_rft_seconds;

   num_rfts = get_num_rfts();

   sw_att(LCYAN,&popup_wn_1);
   if (num_rfts == 1)
      v_plst(0,0,"  Response Force Time  ",&popup_wn_1);
   else
   {
      v_plst(0,0,"  Response Force Times ",&popup_wn_1);
      v_plst(2,0,"        Minimum        ",&popup_wn_1);
      v_plst(6,0,"        Maximum        ",&popup_wn_1);
      sw_att(LBLUE,&popup_wn_1);
      v_plst(5,0,"      DDDDDDDDDD       ",&popup_wn_1);
   }

   min_rft = get_ASD_rft(0);

   /* partition the current minimum ASD RFT into minutes and seconds */
   partition_seconds(min_rft,&min_rft_minutes,&min_rft_seconds);

   if (num_rfts == 1)
```

```
/* display only the minimum RFT minutes and seconds */
{
   sw_att(LWHITE,&popup_wn_1);
   sprintf(temp_string_1,"%2d",min_rft_minutes);
   v_plst(2,6,temp_string_1,&popup_wn_1);
   sprintf(temp_string_1,"%2d",min_rft_seconds);
   v_plst(3,6,temp_string_1,&popup_wn_1);

   sw_att(LCYAN,&popup_wn_1);
   if (min_rft_minutes == 1)
      v_plst(2,9,"minute ",&popup_wn_1);
   else
      v_plst(2,9,"minutes",&popup_wn_1);

   if (min_rft_seconds == 1)
      v_plst(3,9,"second ",&popup_wn_1);
   else
      v_plst(3,9,"seconds",&popup_wn_1);
}
else
/* display both the minimum and maximum RFT values */
{
   sw_att(LWHITE,&popup_wn_1);
   sprintf(temp_string_1,"%2d",min_rft_minutes);
   v_plst(3,6,temp_string_1,&popup_wn_1);
   sprintf(temp_string_1,"%2d",min_rft_seconds);
   v_plst(4,6,temp_string_1,&popup_wn_1);

   max_rft = get_ASD_rft(num_rfts-1);

   /* partition the current maximum ASD RFT into minutes and seconds */
   partition_seconds(max_rft,&max_rft_minutes,&max_rft_seconds);

   sprintf(temp_string_1,"%2d",max_rft_minutes);
   v_plst(7,6,temp_string_1,&popup_wn_1);
   sprintf(temp_string_1,"%2d",max_rft_seconds);
   v_plst(8,6,temp_string_1,&popup_wn_1);

   sw_att(LCYAN,&popup_wn_1);
   if (min_rft_minutes == 1)
      v_plst(3,9,"minute ",&popup_wn_1);
   else
      v_plst(3,9,"minutes",&popup_wn_1);

   if (min_rft_seconds == 1)
      v_plst(4,9,"second ",&popup_wn_1);
   else
      v_plst(4,9,"seconds",&popup_wn_1);

   if (max_rft_minutes == 1)
      v_plst(7,9,"minute ",&popup_wn_1);
   else
      v_plst(7,9,"minutes",&popup_wn_1);
```

```
      if (max_rft_seconds == 1)
         v_plst(8,9,"second ",&popup_wn_1);
      else
         v_plst(8,9,"seconds",&popup_wn_1);
   }

   return;
}

/* ------------------------------------------------------------------------ */

/* remove_rft_values_window removes the pop-up rft values window            */

remove_rft_values_window()

{
   if (window_on_screen(&popup_wn_1))
      unset_wn(&popup_wn_1);

   return;
}

/* ------------------------------------------------------------------------ */

/* popup_current_rft_window displays a popup window to display the given RFT */

popup_current_rft_window()

{
   /* place current rft window on screen */
   mod_wn(CURRENT_RFT_WINDOW_TOP,
          CURRENT_RFT_WINDOW_COL,
          CURRENT_RFT_WINDOW_HEIGHT,
          CURRENT_RFT_WINDOW_WIDTH,
          &popup_wn_1);

   sw_att(LWHITE,&popup_wn_1);
   set_wn(&popup_wn_1);

   /* display the current rft window text */
   v_plst(0,0,"    Response Force Time    ",&popup_wn_1);

   sw_att(LBLUE,&popup_wn_1);
   v_plst(1,0,"DDDDDDDDDDDDDDDDDDDDDDDDDDD",&popup_wn_1);

   sw_att(LCYAN,&popup_wn_1);
   v_plst(2,0,"       RFT number          ",&popup_wn_1);

   return;
}

/* ------------------------------------------------------------------------ */

/* display_current_rft displays the given rft number and value in minutes   */
```

```
/* and seconds                                                              */

display_current_rft(rft_num)
int rft_num;

{
   int rft, rft_minutes, rft_seconds;

   rft = get_ASD_rft(rft_num);

   /* partition the given RFT into minutes and seconds */
   partition_seconds(rft,&rft_minutes,&rft_seconds);

   sw_att(LHIGHLITE,&popup_wn_1);
   sprintf(temp_string_1,"%2d",rft_num+1);
   v_plst(2,18,temp_string_1,&popup_wn_1);

   sw_att(LWHITE,&popup_wn_1);
   sprintf(temp_string_1,"%2d",rft_minutes);
   v_plst(3,8,temp_string_1,&popup_wn_1);
   sprintf(temp_string_1,"%2d",rft_seconds);
   v_plst(4,8,temp_string_1,&popup_wn_1);

   sw_att(LCYAN,&popup_wn_1);
   if (rft_minutes == 1)
      v_plst(3,11,"minute ",&popup_wn_1);
   else
      v_plst(3,11,"minutes",&popup_wn_1);

   if (rft_seconds == 1)
      v_plst(4,11,"second ",&popup_wn_1);
   else
      v_plst(4,11,"seconds",&popup_wn_1);

   return;
}

/* ------------------------------------------------------------------------ */

/* remove_current_rft_window removes the pop-up current rft window          */

remove_current_rft_window()

{
   if (window_on_screen(&popup_wn_1))
      unset_wn(&popup_wn_1);

   return;
}

/* ------------------------------------------------------------------------ */

/* popup_rft_error_window displays a popup window to display the given error */
/* error must be MAX_LESS_THAN_MIN or RANGE_TOO_SMALL                       */
```

```
popup_rft_error_window(error)
int error;

{
   bell();

   /* place rft error window on screen */
   mod_wn(RFT_ERROR_WINDOW_TOP,
          RFT_ERROR_WINDOW_COL,
          RFT_ERROR_WINDOW_HEIGHT,
          RFT_ERROR_WINDOW_WIDTH,
          &popup_wn_2);

   sw_att(LRED,&popup_wn_2);
   set_wn(&popup_wn_2);

   /* display the error window text */
   v_plst(0,CENTER_TXT,"ERROR IN RESPONSE FORCE TIME RANGE",&popup_wn_2);
   sw_att(LWHITE,&popup_wn_2);
   switch (error)
   {
      case MAX_LESS_THAN_MIN : v_plst(1,CENTER_TXT,
                                      "Minimum must be less than Maximum",
                                      &popup_wn_2);
                               break;

      case RANGE_TOO_SMALL   : v_plst(1,CENTER_TXT,
                                      "Range is Too Small",
                                      &popup_wn_2);
                               break;
   }

   return;
}

/* ------------------------------------------------------------------------ */

/* remove_rft_error_window removes the pop-up rft error window              */

remove_rft_error_window()

{
   if (window_on_screen(&popup_wn_2))
      unset_wn(&popup_wn_2);

   return;
}

/* ------------------------------------------------------------------------ */

/* popup_stranded_PE_window displays a popup window to indicate that PEs are */
/* unreachable within the current ASD                                        */
```

```
popup_stranded_PE_window(start_area,end_area)
int start_area,end_area;

{
   bell();

   /* place rft error window on screen */
   mod_wn(STRANDED_PE_WINDOW_TOP,
          STRANDED_PE_WINDOW_COL,
          STRANDED_PE_WINDOW_HEIGHT,
          STRANDED_PE_WINDOW_WIDTH,
          &popup_wn);

   sw_att(LRED,&popup_wn);
   set_wn(&popup_wn);

   /* display the error window text */
   v_plst(0,0,"UNREACHABLE PROTECTION ELEMENTS PRESENT",&popup_wn);

   sw_att(LCYAN,&popup_wn);
   v_plst(1,0,"                 between                 ",&popup_wn);
   v_plst(3,0,"                   and                   ",&popup_wn);

   sw_att(LWHITE,&popup_wn);
   /* display the start area label */
   v_plst(2,CENTER_TXT,
          get_full_ASD_area_label(start_area),
          &popup_wn);
   /* display the end area label */
   v_plst(4,CENTER_TXT,
          get_full_ASD_area_label(end_area),
          &popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* remove_stranded_PE_window removes the pop-up stranded PE error window    */

remove_stranded_PE_window()

{
   if (window_on_screen(&popup_wn))
      unset_wn(&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* display_helicopter_insert_error displays a popup window to display       */
/* an error indicating that the helicopter PE cannot be inserted into the   */
/* current ASD since the current threat does not have a helicopter          */
```

```
display_helicopter_insert_error()

{
   bell();

   /* place error window on screen */
   mod_wn(INSERT_ERROR_WINDOW_TOP,
          INSERT_ERROR_WINDOW_COL,
          INSERT_ERROR_WINDOW_HEIGHT,
          INSERT_ERROR_WINDOW_WIDTH,
          &popup_wn_1);

   sw_att(LRED,&popup_wn_1);
   set_wn(&popup_wn_1);

   /* display the error window text */
   v_plst(0,CENTER_TXT,"HELICOPTER PROTECTION ELEMENT CANNOT BE INSERTED",
          &popup_wn_1);
   sw_att(LWHITE,&popup_wn_1);
   v_plst(1,CENTER_TXT,"Threat Attributes Must Include a Helicopter",
          &popup_wn_1);

   return;
}

/* ------------------------------------------------------------------------ */

/* remove_helicopter_insert_error_window removes the insert error window    */

remove_helicopter_insert_error_window()

{
   if (window_on_screen(&popup_wn_1))
      unset_wn(&popup_wn_1);

   return;
}

/* ------------------------------------------------------------------------ */

/* popup_turbo_progress_window displays a popup window to monitor the       */
/* progress of Turbo_SEApath                                                */

popup_turbo_progress_window()

{
   /* place turbo progress window on screen */
   mod_wn(TURBO_PROGRESS_WINDOW_TOP,
          TURBO_PROGRESS_WINDOW_COL,
          TURBO_PROGRESS_WINDOW_HEIGHT,
          TURBO_PROGRESS_WINDOW_WIDTH,
          &popup_wn);
```

```
   sw_att(LWHITE,&popup_wn);
   set_wn(&popup_wn);

   /* display the initial window text */
   v_plst(0,0,"  Calculating Path Vulnerabilities   ",&popup_wn);

   v_plst(2,0,"       Path Analysis Underway        ",&popup_wn);
   v_plst(3,0,"    Now Estimating Analysis Time     ",&popup_wn);

   sw_att(LBLUE,&popup_wn);
   v_plst(1,0,"DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD",&popup_wn);

   last_percent_displayed = -1;
   turbo_progress_updated = FALSE;

   return;
}

/* ------------------------------------------------------------------------ */

/* display_turbo_progress displays the progress of Turbo_SEApath analysis by */
/* showing the given percentage of paths currently completed and an estimate */
/* of the time remaining in minutes.                                         */

display_turbo_progress(percent_paths_done, min_remain)
int percent_paths_done;
float min_remain;

{
   /* do not update the display unless a new path percentage is passed */
   if (last_percent_displayed == percent_paths_done) return;

   last_percent_displayed = percent_paths_done;

   if (!turbo_progress_updated)
   {
      sw_att(LCYAN,&popup_wn);
      v_plst(2,0,"            of Path Analysis Complete",&popup_wn);
      v_plst(3,0,"            Estimated Time Remaining ",&popup_wn);
      turbo_progress_updated = TRUE;
   }

   sw_att(LWHITE,&popup_wn);
   sprintf(temp_string_1,"%3d percent",percent_paths_done);
   v_plst(2,0,temp_string_1,&popup_wn);

   if (min_remain < 1.0)
   /* time remaining is less than or equal to one minute */
   {
      v_plst(3,1," 1 minute ",&popup_wn);
   }
   else if (min_remain < 59.0)
   /* time remaining is less than 1 hour */
   /* display time remaining in minutes */
```

```
   {
      sprintf(temp_string_1,"%2.0f minutes",min_remain+1.0);
      v_plst(3,1,temp_string_1,&popup_wn);
   }
   else if (min_remain < 1440.0)
   /* time remaining is less than 1 day (1440 minutes) */
   /* display time remaining in fractional hours */
   {
      sprintf(temp_string_1,"%4.1f hours",(min_remain+1.0)/60.0);
      v_plst(3,1,temp_string_1,&popup_wn);
   }
   else
   /* time remaining is more than 1 day */
   {
      sw_att(LRED,&popup_wn);
      v_plst(3,0," Over 1 Day ",&popup_wn);
   }

   return;
}

/* ------------------------------------------------------------------------ */

/* remove_turbo_progress_window removes the turbo progress window           */

remove_turbo_progress_window()

{
   if (window_on_screen(&popup_wn))
      unset_wn(&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* popup_PE_copy_window displays a pop-up copy from or to window for the PE */
/* specified by layer and PE_num in the current ASD ensuring that the PE box */
/* is not obscured by the pop-up window                                     */
/* If the window is already on the screen it is moved if necessary or left  */
/* alone if it will not obscure the given PE. If it is not on the screen the */
/* window is placed on the screen so as not to obscure the given PE.        */
/* srce_dest specifies whether the window is to indicate the source or      */
/* destination PE in the copy operation. srce_dest must be SOURCE or        */
/* DESTINATION                                                              */

popup_PE_copy_window(layer,PE_num,srce_dest)
int layer, PE_num, srce_dest;

{
   int popup_col, window_ok;

   window_ok = FALSE;
```

```
/* determine which side the window should be on */
if (PE_on_left(layer,PE_num) ||
    (PE_base_row(layer,PE_num)+1) >
    (POPUP_TOP+PE_COPY_WINDOW_HEIGHT-3)
   )
   popup_col = POPUP_RIGHT_COL;
else
   popup_col = POPUP_LEFT_COL;

/* if the PE copy window is already on the screen in the  */
/* correct place so as not to obscure the given PE then   */
/* do nothing, else remove it from the screen to prepare  */
/* for moving                                             */
if (window_on_screen(&popup_wn))
   {
      /* set window dimensions to FULL to check left column */
      dim_wn(FULL,&popup_wn);

      /* check if window is already in correct place */
      if (popup_wn.cb == popup_col)
         {
            /* window is already in correct place   */
            /* set window dimensions back to INSIDE */
            dim_wn(INSIDE,&popup_wn);
            window_ok = TRUE;
         }
      else
         {
            /* window will obscure PE if not moved  */
            /* set window dimensions back to INSIDE */
            /* before removing window               */
            dim_wn(INSIDE,&popup_wn);
            remove_PE_label_window();
         }
   }

if (!window_ok)
{
   /* place window on screen in correct place */
   mod_wn(POPUP_TOP,popup_col,
          PE_COPY_WINDOW_HEIGHT,
          POPUP_WIDTH,
          &popup_wn);

   sw_att(LCYAN,&popup_wn);
   set_wn(&popup_wn);

   /* display the static label window titles */
   v_plst(3,0,"             between              ",&popup_wn);
   v_plst(5,0,"               and                ",&popup_wn);

   sw_att(LBLUE,&popup_wn);
   v_plst(1,0,"DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD",&popup_wn);
}
```

```
   /* display the From or To label */
   sw_att(LCYAN,&popup_wn);
   if (srce_dest == SOURCE)
   {
      v_plst(0,CENTER_TXT,
         "            Copy From             ",
         &popup_wn);
   }
   else
   {
      v_plst(0,CENTER_TXT,
         "             Copy To              ",
         &popup_wn);
   }

   /* set window attribute back */
   sw_att(LWHITE,&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* display_PE_copy displays the PE label information within the PE copy     */
/* window for the PE specified by layer and PE_num                          */

display_PE_copy(layer,PE_num)
int layer,PE_num;

{
   PROT_ELEM *PE_ptr;

   PE_ptr = get_ASD_PE_ptr(layer,PE_num);

   /* clear the PE label field */
   mv_cs(2,0,&popup_wn);
   v_qch(' ',MAX_PE_LABEL_LEN-1,&popup_wn);

   /* display the PE label */
   v_plst(2,CENTER_TXT,
          get_PE_cat_spec_ptr(PE_ptr->type)->label,
          &popup_wn);

   /* clear the input area field */
   mv_cs(4,0,&popup_wn);
   v_qch(' ',MAX_PE_LABEL_LEN-1,&popup_wn);

   /* display the input area label */
   v_plst(4,CENTER_TXT,
          get_full_ASD_area_label(PE_ptr->input_area),
          &popup_wn);

   /* clear the output area field */
```

```
   mv_cs(6,0,&popup_wn);
   v_qch(' ',MAX_PE_LABEL_LEN-1,&popup_wn);

   /* display the output area label */
   v_plst(6,CENTER_TXT,
          get_full_ASD_area_label(PE_ptr->output_area),
          &popup_wn);

   /* set window attribute back */
   sw_att(LWHITE,&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* remove_PE_copy_window removes the PE copy pop-up window                  */

remove_PE_copy_window()

{
   if (window_on_screen(&popup_wn))
      unset_wn(&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* popup_save_ASD_offer_window displays a popup window to allow the user to  */
/* save the current ASD                                                      */
/* the given messag is displayed in the window to indicate why this offer is */
/* being made (example: "Before Retrieving New ASD ?")                       */
/* Note: the message string cannot exceed 27 printable characters            */

popup_save_ASD_offer_window(messag)
char *messag;

{
   /* place window on screen */
   mod_wn(SAVE_ASD_OFFER_WINDOW_TOP,
          SAVE_ASD_OFFER_WINDOW_COL,
          SAVE_ASD_OFFER_WINDOW_HEIGHT,
          SAVE_ASD_OFFER_WINDOW_WIDTH,
          &popup_wn);

   sw_att(LCYAN,&popup_wn);
   set_wn(&popup_wn);

   /* display the window text */
   v_plst(0,0,"  Changes Have Been Made   ",&popup_wn);
   sw_att(LWHITE,&popup_wn);
   v_plst(2,0,"   Save the Current ASD    ",&popup_wn);
   v_plst(3,CENTER_TXT,messag,&popup_wn);
```

```
   sw_att(LBLUE,&popup_wn);
   v_plst(1,0,"DDDDDDDDDDDDDDDDDDDDDDDDDDD",&popup_wn);
   v_plst(4,0,"     DDDDDDDDDDDDDDD      ",&popup_wn);
   v_plst(5,0,"     ZDDDDD?   ZDDDD?     ",&popup_wn);
   v_plst(6,0,"     3     3   3    3      ",&popup_wn);
   v_plst(7,0,"     @DDDDDY   @DDDDY     ",&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* highlight_save_ASD_setting highlights the options YES or NO based on the  */
/* given state. If state is TRUE the YES option is highlighted within the    */
/* save_ASD_setting window, otherwise the NO option is highlighted.          */

highlight_save_ASD_setting(state)
int state;

{
   /* if state is TRUE then unhighlight NO and highlight YES */
   if (state)
   {
      sw_att(LWHITE,&popup_wn);
      v_plst(6,16," NO ",&popup_wn);
      sw_att(LHIGHLITE,&popup_wn);
      v_plst(6,6," YES ",&popup_wn);
   }
   /* otherwise unhighlight YES and highlight NO */
   else
   {
      sw_att(LWHITE,&popup_wn);
      v_plst(6,6," YES ",&popup_wn);
      sw_att(LHIGHLITE,&popup_wn);
      v_plst(6,16," NO ",&popup_wn);
   }

   return;
}

/* ------------------------------------------------------------------------ */

/* remove_save_ASD_offer_window removes the save_ASD_offer window            */

remove_save_ASD_offer_window()

{
   if (window_on_screen(&popup_wn))
      unset_wn(&popup_wn);

   return;
}
```

```
/* ************************ end SAVIPOP2.C source file ********************** */

/* ************************ SAVIPRIN.C source file ************************** */

#include <stdio.h>
#include <savidefs.h>

/* define page setup parameters */
#define LINES_PER_PAGE 66
#define TOP_MARGIN 4
#define BOTTOM_MARGIN 4
#define PAGE_WIDTH 136
#define MARGIN 2
#define INDENT 0

/* define printout titles */
#define TABLES_TITLE "- SAVI ASD Settings Tables -"
#define VULNERABILITY_TITLE "- SAVI Path Vulnerability Diagnostics -"
#define RECOMMENDATIONS_TITLE "- SAVI Path Improvement Recommendations -"

/* define printer command codes */
#define ESC 27
#define DC2 18
#define SI 15

int num_copies, page_num, line_count, lines_used,
    PE_type, escaped, print_head;

char *label;
char line_buf[PAGE_WIDTH+1];
FILE *stream;

/* ------------------------------------------------------------------------ */

savi_print_diagram()

{
   if (!printer_ready(0))
   {
      popup_printer_not_ready_window();
      user_acknowledge_wait();
      remove_printer_not_ready_window();

      dump_key_buffer();
      return(0);
   }

   print_current_ASD(stream);

   dump_key_buffer();

   return(0);
}
```

```
/* ------------------------------------------------------------------------ */

savi_print_tables()

{
   int i,layer,PE_index, rft_num;
   PROT_ELEM *PE_ptr;

   if (!printer_ready(0))
   {
      popup_printer_not_ready_window();
      user_acknowledge_wait();
      remove_printer_not_ready_window();

      dump_key_buffer();
      return(0);
   }

   popup_wait_window("Now Printing");

   escaped = FALSE;
   label = get_ASD_label();

   /* place printer in compressed mode */
   fprintf(stream,"%c%c",ESC,SI);

   for (i = 0 ; i < num_copies ; i++)
   {
      if (escaped) break;

      page_num = 0;
      line_count = 0;
      print_head = 0;

      if (!escaped) print_title(TABLES_TITLE);
      if (!escaped) print_threat_settings();
      if (!escaped) print_rft_range();
      if (!escaped)
      {
         request_lines(MAX_MODIFIABLE_AREAS+7,TABLES_TITLE);
         print_area_table();
      }

      if (!escaped)
      {
         for (layer = 0 ; layer < MAX_LAYERS_PER_ASD ; layer++)
         {
            if (escaped) break;
            if (get_ASD_layer_bypass(layer))
               print_bypass_information(layer);
            else
            {
               form_feed();
               print_title(TABLES_TITLE);
```

```
            request_lines(4,TABLES_TITLE);
            if (!escaped) print_layer_information(layer);
            if (!escaped) print_bar();

            for (PE_index = 0 ; PE_index < MAX_PE_PER_LAYER ; PE_index++)
            {
               PE_ptr = get_ASD_PE_ptr(layer,PE_index);
               PE_type = PE_ptr->type;
               if (PE_ptr->present)
               {
                  /* request enough lines for an entire PE table   */
                  /* before attempting to print the table.  Note   */
                  /* that each of the print routines which produce */
                  /* part of the PE table rely on this amount of   */
                  /* space being set aside before they are called  */
                  /* (these routines NEVER call request_lines)     */
                  request_lines(max(get_number_of_comps(DELAY,PE_type),1)+
                                max(get_number_of_comps(NONDET,PE_type),1)+
                                15,
                                TABLES_TITLE);

                  if (!escaped) print_PE_info(layer,PE_index);
                  if (!escaped) print_comp_table(DELAY,layer,PE_index);
                  if (!escaped) print_comp_table(NONDET,layer,PE_index);
               }
            }
         }
      }
   }
}

/* explicitly print TASK info */

if (!escaped)
{
   form_feed();
   print_title(TABLES_TITLE);
   request_lines(4,TABLES_TITLE);
   if (!escaped) print_layer_information(layer);
   if (!escaped) print_bar();

   layer = MAX_LAYERS_PER_ASD;
   PE_index = 0;

   PE_ptr = get_ASD_PE_ptr(layer,PE_index);
   PE_type = PE_ptr->type;

   request_lines(max(get_number_of_comps(DELAY,PE_type),1)+
                 max(get_number_of_comps(NONDET,PE_type),1)+
                 15,
                 TABLES_TITLE);

   if (!escaped) print_PE_info(layer,PE_index);
   if (!escaped) print_comp_table(DELAY,layer,PE_index);
```

```
      if (!escaped) print_comp_table(NONDET,layer,PE_index);
   }

   form_feed();

   /* cancel compressed mode */
   fprintf(stream,"%c",DC2);

   remove_wait_window();

   dump_key_buffer();

   return(0);
}

/* ------------------------------------------------------------------------ */

savi_analyze_print_vulnerability()

{
   int i, last_path, path_num, rft_num;

   if (!printer_ready(0))
   {
      popup_printer_not_ready_window();
      user_acknowledge_wait();
      remove_printer_not_ready_window();

      dump_key_buffer();
      return(0);
   }

   if (!select_current_rft())
   {
      dump_key_buffer();
      return(0);
   }

   rft_num = get_current_rft_num();

   popup_wait_window("Now Printing");

   escaped = FALSE;
   label = get_ASD_label();

   /* place printer in compressed mode */
   fprintf(stream,"%c%c",ESC,SI);

   for (i = 0 ; i < num_copies ; i++)
   {
      if (escaped) break;

      page_num = 0;
      line_count = 0;
```

```
      print_head = 0;

      if (!escaped) print_title(VULNERABILITY_TITLE);
      if (!escaped) print_number_of_paths();
      if (!escaped) print_threat_settings();
      if (!escaped) print_rft(rft_num);

      if (!escaped)
      {
         last_path = get_number_of_critical_paths() - 1;
         for (path_num = 0 ; path_num <= last_path; path_num++)
         {
            if (escaped) break;
            /* request enough lines for the entire diagnostics table      */
            /* before attempting to print the table.  Note that each of   */
            /* the print routines which produce part of the diagnostics   */
            /* table rely on this amount of space being set aside         */
            /* for them before they are called (these routines NEVER call */
            /* request_lines on their own).                               */
            request_lines(number_of_PEs_on_path(rft_num,path_num) -
                          get_num_bypasses() + 18,
                          VULNERABILITY_TITLE);
            if (!escaped) print_bar();
            if (!escaped) print_path_rank(path_num);
            if (!escaped) print_pi(rft_num,path_num);
            if (!escaped) print_diagnostics(rft_num, path_num);
            if (!escaped) print_path_table(rft_num, path_num);
         }
      }

      if (!escaped) form_feed();
   }

   if (escaped) form_feed();

   /* cancel compressed mode */
   fprintf(stream,"%c",DC2);

   remove_wait_window();

   dump_key_buffer();

   return(0);
}

/* ------------------------------------------------------------------------ */

savi_analyze_print_recommendations()

{
   int i, last_path, path_num, rft_num;

   if (!printer_ready(0))
   {
```

```
   popup_printer_not_ready_window();
   user_acknowledge_wait();
   remove_printer_not_ready_window();

   dump_key_buffer();
   return(0);
}

if (!select_current_rft())
{
   dump_key_buffer();
   return(0);
}

rft_num = get_current_rft_num();

popup_wait_window("Now Printing");

escaped = FALSE;
label = get_ASD_label();

/* place printer in compressed mode */
fprintf(stream,"%c%c",ESC,SI);

for (i = 0 ; i < num_copies ; i++)
{
   if (escaped) break;

   page_num = 0;
   line_count = 0;
   print_head = 0;

   if (!escaped) print_title(RECOMMENDATIONS_TITLE);
   if (!escaped) print_number_of_paths();
   if (!escaped) print_threat_settings();
   if (!escaped) print_rft(rft_num);

   if (!escaped)
   {
      last_path = get_number_of_critical_paths() - 1;
      for (path_num = 0 ; path_num <= last_path; path_num++)
      {
         if (escaped) break;
         /* request enough lines for the recommendations message       */
         /* before attempting to print the table.                      */
         request_lines(number_of_PEs_on_path(rft_num, path_num) -
                       get_num_bypasses() + 17,
                       RECOMMENDATIONS_TITLE);
         if (!escaped) print_bar();
         if (!escaped) print_path_rank(path_num);
         if (!escaped) print_pi(rft_num, path_num);
         if (!escaped) print_recommendations(rft_num, path_num);
         if (!escaped) print_path_table(rft_num, path_num);
      }
```

```
      }

      if (!escaped) form_feed();
   }

   if (escaped) form_feed();

   /* cancel compressed mode */
   fprintf(stream,"%c",DC2);

   remove_wait_window();

   dump_key_buffer();

   return(0);
}

/* ------------------------------------------------------------------------ */

/* print_title prints the given title string centered below the top margin   */
/* of the current page, prints the ASD label two lines below that and prints */
/* a blank line below the label                                             */

print_title(title_str)
char *title_str;

{
   int i;

   /* directly increment the line count since */
   /* we must be at the top of the page now   */
   line_count += 4 + TOP_MARGIN;
   page_num++;
   for (i=0 ; i<TOP_MARGIN ; i++)
      print_head += sprintf(line_buf+print_head,"\n");
   escaped_printf(TOP_MARGIN);
   center(title_str);
   escaped_printf(1);
   sprintf(line_buf+print_head,"\n");
   escaped_printf(1);
   center(label);
   sprintf(line_buf+print_head,"\n");
   escaped_printf(2);
   return;

}

/* ------------------------------------------------------------------------ */

print_rft_range()

{
   int num_rfts;
```

```
   request_lines(2,TABLES_TITLE);
   spaces(MARGIN);

   num_rfts = get_num_rfts();

   if (num_rfts == 1)
      print_head += sprintf(line_buf+print_head,
            "Response Force Time Range : 1 RFT of %d seconds\n\n",
            get_ASD_rft(0));
   else
      print_head += sprintf(line_buf+print_head,
            "Response Force Time Range : %d RFTs from %d to %d seconds\n\n",
            num_rfts, get_ASD_rft(0), get_ASD_rft(num_rfts-1));

   escaped_printf(2);

   return;
}

/* ------------------------------------------------------------------------ */

print_rft(rft_num)
int rft_num;

{
   int rft;
   int mins,secs;

   rft = get_ASD_rft(rft_num);

   /* partition the Response Force Time into minutes and seconds */
   partition_seconds(rft,&mins,&secs);
   request_lines(2,TABLES_TITLE);
   spaces(MARGIN);

   if (mins == 1)
      print_head += sprintf(line_buf+print_head,
            "Response Force Time #%d : %d minute ",rft_num+1, mins);
   else
      print_head += sprintf(line_buf+print_head,
            "Response Force Time #%d : %d minutes ",rft_num+1, mins);

   if (secs == 1)
      print_head += sprintf(line_buf+print_head,
            "and %d second (%d sec)\n\n",secs,rft);
   else
      print_head += sprintf(line_buf+print_head,
            "and %d seconds (%d sec)\n\n",secs,rft);

   escaped_printf(2);

   return;

}
```

```
/* ------------------------------------------------------------------------ */

print_threat_settings()

{
   float rft;
   int mins,secs;

   request_lines(4,TABLES_TITLE);

   spaces(MARGIN);
   print_head += sprintf(line_buf+print_head,
         "Threat Attributes : %-s\n",get_threat_label(get_current_threat()));
   escaped_printf(1);

   spaces(MARGIN);
   if (get_current_direction() == ENTRY)
      print_head += sprintf(line_buf+print_head,
         "Threat Objective : ENTRY - Sabotage or Hands-On Theft Analysis\n");
   else
      print_head += sprintf(line_buf+print_head,
         "Threat Objective : ENTRY/EXIT - Containment Theft Analysis\n");
   escaped_printf(1);

   spaces(MARGIN);
   switch (get_current_strategy())
   {
      case FORCE   : print_head += sprintf(line_buf+print_head,
                     "Threat Tactics : FORCE - Force Intrusion Only\n\n");
                     break;

      case MIXED   : print_head += sprintf(line_buf+print_head,
                     "Threat Tactics : MIXED - Force & Deceit Intrusion\n\n");
                     break;
   }
   escaped_printf(2);

   return;
}

/* ------------------------------------------------------------------------ */

/* print_number_of_paths prints the total number of paths which exist in the */
/* current ASD                                                               */

print_number_of_paths()

{
   int num_paths;

   request_lines(2,VULNERABILITY_TITLE);

   num_paths = get_valid_paths(get_current_rft_num());
```

```
   spaces(MARGIN);
   if (num_paths == 1)
      print_head += sprintf(line_buf+print_head,
                            "There is Only One Path Through this ASD.\n\n");
   else
      print_head += sprintf(line_buf+print_head,
                            "There are %d Paths Through this ASD.\n\n",
                            num_paths);
   escaped_printf(2);

   return;

}

/* ------------------------------------------------------------------------ */

int print_layer_information(layer)
int layer;

{
   spaces(MARGIN);
   print_head += sprintf(line_buf+print_head,
                         "Protection Elements Leading from %s :\n\n",
                         get_full_ASD_area_label(layer));
   escaped_printf(2);

   return;
}

/* ------------------------------------------------------------------------ */

/* print_path_rank prints the cardinal rank of the given path               */

print_path_rank(path_num)
int path_num;

{
   spaces(MARGIN);

   if (path_num == 0)
   {
      print_head += sprintf(line_buf+print_head,
                            "Most Vulnerable Path\n");
   }
   else
   {
      print_head += sprintf(line_buf+print_head,
                            "%sMost Vulnerable Path\n",
                            cardinal_number(path_num+1));
   }
   escaped_printf(1);

   return;
```

```
}

/* ------------------------------------------------------------------------ */

/* print_pi prints the probability of interruption of the given path        */

print_pi(rft_num,path_num)
int rft_num, path_num;

{
   spaces(MARGIN);

   print_head += sprintf(line_buf+print_head,
                         "Probability of Interruption : %6.4f\n\n",
                         get_prob_interruption(rft_num, path_num));
   escaped_printf(2);

   return;
}

/* ------------------------------------------------------------------------ */

print_diagnostics(rft_num, path_num)
int rft_num, path_num;

{
   PROT_ELEM *PE_ptr;
   int layer, PE_num, segment, test, rft,
       cdp_direction, direction, auth_unauth,
       first_delay, temp_eqn;
   float temp_float;

   /* trd is time remaining after detection */
   int trd, trd_minutes, trd_seconds;

   /* tri is time remaining after interruption */
   int tri, tri_minutes, tri_seconds;

   /* get the Response Force Time */
   rft = get_ASD_rft(rft_num);

   /* determine the correct data labels depending on the */
   /* timely_detection_point function                    */
   test = timely_detection_point(rft_num,path_num,
                                 &layer,&PE_num,&segment,
                                 &cdp_direction);

   /* partition the time remaining after detection */
   /* into minutes and seconds                     */
   trd = get_time_remaining(rft_num, path_num);
   partition_seconds(trd,&trd_minutes,&trd_seconds);

   tri = abs(trd - rft);
```

```
/* if the TDP is OFFSITE or the cumulative       */
/* path delay is deficient then the              */
/* path delay deficiency is the difference       */
/* between the RFT and the cumulative path delay */
/* without the delay on the input path segment   */
/* of the first PE on the path.  Also, add in    */
/* one second since a tie goes to the intruder.  */
if ( (test == TDP_OFFSITE) || (test == PATH_DELAY_DEFICIENCY) )
{
   /* determine the first PE on the path */
   get_first_present_PE_on_path(rft_num, path_num,
                                &layer, &PE_num,
                                &direction, &auth_unauth);
   PE_ptr = get_ASD_PE_ptr(layer,PE_num);

   /* calculate the delay on the input path segment of */
   /* the first PE on the path                         */
   min_delay_traversal(layer, PE_num, auth_unauth, INPUT_SEG, direction,
                       &temp_float, &temp_eqn);
   first_delay = (int)temp_float;

   tri = abs(trd - rft - first_delay);

   /* add in one second if result will be <= 5999 (99 min, 59 sec) */
   if (tri < 5999) tri += 1;
}

/* partition the time remaining after interruption */
/* into minutes and seconds                        */
partition_seconds(tri,&tri_minutes,&tri_seconds);

switch (test)
{
   case TIMELY_DETECTION_OK :
      spaces(MARGIN);
      print_head += sprintf(line_buf+print_head,
         "The Critical Detection Point is on the ");

      if (segment == PATH_OUT)
         print_head += sprintf(line_buf+print_head,
               "Output Path Segment of the ");
      else
         print_head += sprintf(line_buf+print_head,
               "Input Path Segment of the ");

      PE_ptr = get_ASD_PE_ptr(layer,PE_num);
      PE_type = PE_ptr->type;

      print_head += sprintf(line_buf+print_head,
               "%s (Protection Element #%d)\n",
               get_PE_cat_spec_ptr(PE_type)->label,
               PE_num_on_layer(layer,PE_num));
      escaped_printf(1);
      if (escaped) return;
```

```
            spaces(MARGIN);
            if (cdp_direction == ENTRY)
            {
               print_head += sprintf(line_buf+print_head,
                  "Leading from %s ",
                   get_full_ASD_area_label(PE_ptr->input_area));
               print_head += sprintf(line_buf+print_head,
                  "to %s on ENTRY toward the TARGET.\n\n",
                   get_full_ASD_area_label(PE_ptr->output_area));
            }
            else
            {
               print_head += sprintf(line_buf+print_head,
                  "Leading from %s ",
                   get_full_ASD_area_label(PE_ptr->output_area));
               print_head += sprintf(line_buf+print_head,
                  "to %s on EXIT from the TARGET.\n\n",
                   get_full_ASD_area_label(PE_ptr->input_area));
            }
            escaped_printf(2);
            if (escaped) return;

            spaces(MARGIN);
            if (trd_minutes == 1)
               print_head += sprintf(line_buf+print_head,
                     "Time Remaining after the Critical Detection Point : \
%d minute ",
                     trd_minutes);
            else
               print_head += sprintf(line_buf+print_head,
                     "Time Remaining after the Critical Detection Point : \
%d minutes ",
                     trd_minutes);

            if (trd_seconds == 1)
               print_head += sprintf(line_buf+print_head,
                     "and %d second (%d sec)\n",trd_seconds,trd);
            else
               print_head += sprintf(line_buf+print_head,
                     "and %d seconds (%d sec)\n",trd_seconds,trd);

            escaped_printf(1);
            if (escaped) return;

            spaces(MARGIN);
            if (tri_minutes == 1)
               print_head += sprintf(line_buf+print_head,
                     "Time Remaining after Interruption (TRI) : %d minute ",
                     tri_minutes);
            else
               print_head += sprintf(line_buf+print_head,
                     "Time Remaining after Interruption (TRI) : %d minutes ",
                     tri_minutes);
```

```
   if (tri_seconds == 1)
      print_head += sprintf(line_buf+print_head,
            "and %d second (%d sec)\n\n",tri_seconds,tri);
   else
      print_head += sprintf(line_buf+print_head,
            "and %d seconds (%d sec)\n\n",tri_seconds,tri);

   escaped_printf(2);
   break;

case NO_TIMELY_DETECTION :
   spaces(MARGIN);
   print_head += sprintf(line_buf+print_head,
      "The Critical Detection Point is on the ");

   if (segment == PATH_OUT)
      print_head += sprintf(line_buf+print_head,
            "Output Path Segment of the ");
   else
      print_head += sprintf(line_buf+print_head,
            "Input Path Segment of the ");

   PE_ptr = get_ASD_PE_ptr(layer,PE_num);
   PE_type = PE_ptr->type;

   print_head += sprintf(line_buf+print_head,
            "%s (Protection Element #%d)\n",
            get_PE_cat_spec_ptr(PE_type)->label,
            PE_num_on_layer(layer,PE_num));
   escaped_printf(1);
   if (escaped) return;

   spaces(MARGIN);
   if (cdp_direction == ENTRY)
   {
      print_head += sprintf(line_buf+print_head,
         "Leading from %s ",
          get_full_ASD_area_label(PE_ptr->input_area));
      print_head += sprintf(line_buf+print_head,
         "to %s on ENTRY toward the TARGET.\n\n",
          get_full_ASD_area_label(PE_ptr->output_area));
   }
   else
   {
      print_head += sprintf(line_buf+print_head,
         "Leading from %s ",
          get_full_ASD_area_label(PE_ptr->output_area));
      print_head += sprintf(line_buf+print_head,
         "to %s on EXIT from the TARGET.\n\n",
          get_full_ASD_area_label(PE_ptr->input_area));
   }
   escaped_printf(2);
   if (escaped) return;
```

```
         spaces(MARGIN);
         if (trd_minutes == 1)
            print_head += sprintf(line_buf+print_head,
                  "Time Remaining after the Critical Detection Point : \
%d minute ",
                  trd_minutes);
         else
            print_head += sprintf(line_buf+print_head,
                  "Time Remaining after the Critical Detection Point : \
%d minutes ",
                  trd_minutes);

         if (trd_seconds == 1)
            print_head += sprintf(line_buf+print_head,
                  "and %d second (%d sec)\n",trd_seconds,trd);
         else
            print_head += sprintf(line_buf+print_head,
                  "and %d seconds (%d sec)\n",trd_seconds,trd);

         escaped_printf(1);
         if (escaped) return;

         spaces(MARGIN);
         print_head += sprintf(line_buf+print_head,
         "D WARNING D No Detection Exists Up To and Including the \
Critical Detection Point\n\n");
         escaped_printf(2);
         break;

      case TDP_OFFSITE           : ;
      case PATH_DELAY_DEFICIENCY :
         spaces(MARGIN);
         print_head += sprintf(line_buf+print_head,
            "- WARNING - Cumulative Path Delay Deficiency\n");
         escaped_printf(1);
         if (escaped) return;

         spaces(MARGIN);
         print_head += sprintf(line_buf+print_head,
            "The Critical Detection Point Cannot be Calculated.\n\n");
         escaped_printf(2);
         if (escaped) return;

         spaces(MARGIN);
         if (trd_minutes == 1)
            print_head += sprintf(line_buf+print_head,
                  "Cumulative Path Delay : %d minute ",trd_minutes);
         else
            print_head += sprintf(line_buf+print_head,
                  "Cumulative Path Delay : %d minutes ",trd_minutes);

         if (trd_seconds == 1)
            print_head += sprintf(line_buf+print_head,
```

```
                "and %d second (%d sec)\n",trd_seconds,trd);
         else
            print_head += sprintf(line_buf+print_head,
                "and %d seconds (%d sec)\n",trd_seconds,trd);

         escaped_printf(1);
         if (escaped) return;

         spaces(MARGIN);
         if (tri_minutes == 1)
            print_head += sprintf(line_buf+print_head,
                "Cumulative Path Delay Deficiency : %d minute ",
                tri_minutes);
         else
            print_head += sprintf(line_buf+print_head,
                "Cumulative Path Delay Deficiency : %d minutes ",
                tri_minutes);

         if (tri_seconds == 1)
            print_head += sprintf(line_buf+print_head,
                "and %d second (%d sec)\n\n",tri_seconds,tri);
         else
            print_head += sprintf(line_buf+print_head,
                "and %d seconds (%d sec)\n\n",tri_seconds,tri);

         escaped_printf(2);
         break;
   }

   return;
}

/* ------------------------------------------------------------------------ */

print_recommendations(rft_num, path_num)
int rft_num, path_num;

{
   PROT_ELEM *PE_ptr;
   int layer, PE_num, PE_type, test, segment, tr, rft,
       del_def, del_def_minutes, del_def_seconds, direction, auth_unauth,
       first_delay, temp_eqn;
   float temp_float;

   test = timely_detection_point(rft_num,path_num,
                                 &layer,&PE_num,&segment,&direction);

   /* display the improvement recommendations */
   switch (test)
   {
      /* if the timely_detection point exists, recommend improving */
      /* detection up to and including the TDP and delay after     */
      case NO_TIMELY_DETECTION : ;
      case TIMELY_DETECTION_OK :
```

```
         spaces(MARGIN);
         print_head += sprintf(line_buf+print_head,
               "Consider Improving Detection on any Path Segment \
up to and including the Critical Detection Point on the\n");
         escaped_printf(1);
         if (escaped) return;

         spaces(MARGIN);
         if (segment == PATH_OUT)
            print_head += sprintf(line_buf+print_head,
                  "Output Path Segment of the ");
         else
            print_head += sprintf(line_buf+print_head,
                  "Input Path Segment of the ");

         PE_ptr = get_ASD_PE_ptr(layer,PE_num);
         PE_type = PE_ptr->type;

         print_head += sprintf(line_buf+print_head,
                  "%s (Protection Element #%d)\n",
                  get_PE_cat_spec_ptr(PE_type)->label,
                  PE_num_on_layer(layer,PE_num));
         escaped_printf(1);
         if (escaped) return;

         spaces(MARGIN);
         if (direction == ENTRY)
         {
            print_head += sprintf(line_buf+print_head,
               "Leading from %s ",
                get_full_ASD_area_label(PE_ptr->input_area));
            print_head += sprintf(line_buf+print_head,
               "to %s on ENTRY toward the TARGET.\n",
                get_full_ASD_area_label(PE_ptr->output_area));
         }
         else
         {
            print_head += sprintf(line_buf+print_head,
               "Leading from %s ",
                get_full_ASD_area_label(PE_ptr->output_area));
            print_head += sprintf(line_buf+print_head,
               "to %s on EXIT from the TARGET.\n",
                get_full_ASD_area_label(PE_ptr->input_area));
         }
         escaped_printf(1);
         if (escaped) return;

         spaces(MARGIN);
         print_head += sprintf(line_buf+print_head,
         "Consider Increasing Delay on any Path Segment after this point \
to the Goal.\n\n");
         escaped_printf(2);
         break;
```

```
/* if the TDP is offsite, adding delay on the input */
/* segment of the first PE will not improve the     */
/* path, since no detection prior to delay exists.  */
/* Recommend adding delay to all path               */
/* segments except the input path segment of the    */
/* first PE on the path                             */
case TDP_OFFSITE           : ;
case PATH_DELAY_DEFICIENCY : ;
   /* determine the first PE on the path */
   get_first_present_PE_on_path(rft_num, path_num,
                                &layer, &PE_num,
                                &direction, &auth_unauth);
   PE_ptr = get_ASD_PE_ptr(layer,PE_num);
   PE_type = PE_ptr->type;

   /* the path delay deficiency is the difference   */
   /* between the RFT and the cumulative path delay */
   /* without the delay on the input path segment   */
   /* of the first PE on the path.  Also, add in    */
   /* one second since a tie goes to the intruder.  */
   tr = get_time_remaining(rft_num,path_num);
   rft = get_ASD_rft(rft_num);

   /* calculate the delay on the input path segment of */
   /* the first PE on the path                         */
   min_delay_traversal(layer, PE_num, auth_unauth, INPUT_SEG,
                       direction, &temp_float, &temp_eqn);
   first_delay = (int)temp_float;

   del_def = abs(tr - rft - first_delay);

   /* add in one second if result will be <= 5999 (99 min, 59 sec) */
   if (del_def < 5999) del_def += 1;

   /* partition the delay deficiency */
   /* into minutes and seconds       */
   partition_seconds(del_def,
                     &del_def_minutes,
                     &del_def_seconds);

   spaces(MARGIN);
   if (del_def_minutes == 1)
      print_head += sprintf(line_buf+print_head,
            "Increase the Cumulative Path Delay by at least %d minute ",
            del_def_minutes);
   else
      print_head += sprintf(line_buf+print_head,
            "Increase the Cumulative Path Delay by at least %d minutes ",
            del_def_minutes);

   if (del_def_seconds == 1)
      print_head += sprintf(line_buf+print_head,
            "and %d second (%d sec)\n",del_def_seconds,del_def);
   else
```

```
            print_head += sprintf(line_buf+print_head,
                  "and %d seconds (%d sec)\n",del_def_seconds,del_def);

            escaped_printf(1);
            if (escaped) return;

            spaces(MARGIN);
            print_head += sprintf(line_buf+print_head,
            "On the Path Segments between the Goal and the\n");
            escaped_printf(1);
            if (escaped) return;

            spaces(MARGIN);
            print_head += sprintf(line_buf+print_head,
                     "Output Path Segment of the %s (Protection Element #%d)\n",
                     get_PE_cat_spec_ptr(PE_type)->label,
                     PE_num_on_layer(layer,PE_num));
            escaped_printf(1);
            if (escaped) return;

            spaces(MARGIN);
            print_head += sprintf(line_buf+print_head,
               "Leading from %s ",
                get_full_ASD_area_label(PE_ptr->input_area));
            print_head += sprintf(line_buf+print_head,
               "to %s.\n\n",
                get_full_ASD_area_label(PE_ptr->output_area));
            escaped_printf(2);
            break;
      }

      return;
}

/* ------------------------------------------------------------------------ */

/* print_path_table prints the path table for the given path for the given   */
/* rft within the rft table                                                  */

print_path_table(rft_num, path_num)
int rft_num, path_num;

{
      PROT_ELEM *PE_ptr;
      int layer, PE_num, PE_type, count, direction, auth_unauth, done, check;

      spaces(MARGIN);

      print_head += sprintf(line_buf+print_head,
                           "Protection Elements on the Path :\n",
                           get_prob_interruption(rft_num, path_num));
      escaped_printf(1);

      if (!escaped) print_path_table_header();
```

```
if (!escaped)
{
   count = 0;
   done = FALSE;

   while (!done)
   {
      check = get_PE_on_path(rft_num, path_num, count,
                             &layer, &PE_num,
                             &direction, &auth_unauth);

      if (check == NOT_FOUND)
         done = TRUE;
      else
      {
         if (escaped) break;

         if ( (layer == MAX_LAYERS_PER_ASD) ||
              (get_ASD_PE_ptr(layer,PE_num)->present) )
         {
            PE_ptr = get_ASD_PE_ptr(layer,PE_num);
            PE_type = PE_ptr->type;

            spaces(MARGIN + INDENT);

            if (auth_unauth == UNAUTH)
               print_head += sprintf(line_buf+print_head,
                        "|    FORCE    ");
            else
               print_head += sprintf(line_buf+print_head,
                        "|    DECEIT   ");

            print_head += sprintf(line_buf+print_head,
                     "| %2d %-27s",
                     PE_num_on_layer(layer,PE_num),
                     get_PE_cat_spec_ptr(PE_type)->label);

            if (direction == ENTRY)
            {
               print_head += sprintf(line_buf+print_head,
                        "| %-27s            ",
                        get_full_ASD_area_label(PE_ptr->input_area));

               print_head += sprintf(line_buf+print_head,
                        "| %-27s            |\n",
                        get_full_ASD_area_label(PE_ptr->output_area));
            }
            else
            {
               print_head += sprintf(line_buf+print_head,
                        "| %-27s            ",
                        get_full_ASD_area_label(PE_ptr->output_area));
```

```
                print_head += sprintf(line_buf+print_head,
                        "| %-27s            |\n",
                        get_full_ASD_area_label(PE_ptr->input_area));
             }

             escaped_printf(1);
          }

          count++;
       }
    }

 }

 if (!escaped) print_path_table_footer();
 if (!escaped)
 {
    print_head += sprintf(line_buf+print_head,"\n");
    escaped_printf(1);
 }

 return;
}

/* ------------------------------------------------------------------------ */

/* print_area_table prints the table of settings for all areas in the       */
/* current ASD                                                              */

print_area_table()

{
 int area;

 spaces(MARGIN);
 print_head += sprintf(line_buf+print_head,
                       "Area Data :\n");
 escaped_printf(1);

 if (!escaped) print_area_table_header();

 if (!escaped)
 {
    for (area = 1; area <= MAX_MODIFIABLE_AREAS; area++)
    {
       print_area_data(area);
    }
 }

 if (!escaped) print_area_table_footer();
 if (!escaped)
 {
    print_head += sprintf(line_buf+print_head,"\n");
    escaped_printf(1);
```

```
   }

   return;
}

/* ------------------------------------------------------------------------ */

/* print_comp_table prints the table of settings for either delay or        */
/* detection components for the given PE                                    */

print_comp_table(delay_nondet, layer, PE_num)
int delay_nondet, layer, PE_num;

{
   int comp_num, num_comps;

   spaces(MARGIN);

   if (delay_nondet == DELAY)
      print_head += sprintf(line_buf+print_head,
                            "Delay Component Settings :\n");
   else
      print_head += sprintf(line_buf+print_head,
                            "Detection Component Settings :\n");
   escaped_printf(1);

   if (!escaped) print_comp_table_header(delay_nondet);

   num_comps = get_number_of_comps(delay_nondet,
                                   get_ASD_PE_ptr(layer,PE_num)->type);

   if (num_comps == 0)
      print_no_comps(delay_nondet);
   else
      if (!escaped)
      {
         for (comp_num = 0; comp_num < num_comps; comp_num++)
         {
            print_comp_data(delay_nondet,comp_num,layer,PE_num);
         }
      }

   if (!escaped) print_comp_table_footer(delay_nondet);

   return;
}

/* ------------------------------------------------------------------------ */

int print_path_table_header()

{
   print_path_table_divider();
   spaces(MARGIN+INDENT);
```

```
   print_head += sprintf(line_buf+print_head,
         "|   Tactics   ");
   print_head += sprintf(line_buf+print_head,
         "|    Protection Element Type    |");
   print_head += sprintf(line_buf+print_head,
         "                Input Area                |");
   print_head += sprintf(line_buf+print_head,
         "               Output Area                |\n");
   escaped_printf(1);
   print_path_table_divider();

   return;

}

/* ------------------------------------------------------------------------ */

int print_comp_table_header(delay_nondet)
int delay_nondet;

{
   print_comp_table_divider(delay_nondet);
   spaces(MARGIN+INDENT);

   if (delay_nondet == DELAY)
   {
      print_head += sprintf(line_buf+print_head,
            "|                Component Type                ");
      print_head += sprintf(line_buf+print_head,
            "| Delay Setting                          seconds   ");
      print_head += sprintf(line_buf+print_head,
            "| Free Exit |\n");
   }
   else
   {
      print_head += sprintf(line_buf+print_head,
            "|                Component Type                ");
      print_head += sprintf(line_buf+print_head,
            "| Detection Setting                   probability |\n");
   }

   escaped_printf(1);
   print_comp_table_divider(delay_nondet);

   return;

}

/* ------------------------------------------------------------------------ */

int print_area_table_header()

{
   print_area_table_divider();
```

```
    spaces(MARGIN+INDENT);
    print_head += sprintf(line_buf+print_head,
        "|                                | Distance ");
    print_head += sprintf(line_buf+print_head,
        "| Transit Time | Traversal |\n");
    escaped_printf(1);

    spaces(MARGIN+INDENT);
    print_head += sprintf(line_buf+print_head,
        "|         Area Label             | (meters) ");
    print_head += sprintf(line_buf+print_head,
        "|    (sec)     |  Setting  |\n");
    escaped_printf(1);

    print_area_table_divider();

    return;
}

/* ------------------------------------------------------------------------ */

print_bar()

{
    spaces(MARGIN+INDENT);
    print_head += sprintf(line_buf+print_head,
          "---------------------------------");
    print_head += sprintf(line_buf+print_head,
          "-------------------------------------------------");
    print_head += sprintf(line_buf+print_head,
          "-------------------------------------------------\n\n");
    escaped_printf(2);

    return;
}

/* ------------------------------------------------------------------------ */

print_PE_info(layer,PE_num)
int layer, PE_num;

{
    PROT_ELEM *PE_ptr;
    int dist, time;

    PE_ptr = get_ASD_PE_ptr(layer,PE_num);
    PE_type = PE_ptr->type;

    spaces(MARGIN + INDENT);

    print_head += sprintf(line_buf+print_head,
                          "%s (Protection Element #%d) ",
                          get_PE_cat_spec_ptr(PE_type)->label,
```

```
                        PE_num_on_layer(layer,PE_num));

   print_head += sprintf(line_buf+print_head,
                        "leading from %s ",
                        get_full_ASD_area_label(PE_ptr->input_area));

   print_head += sprintf(line_buf+print_head,
                        "to %s.\n",
                        get_full_ASD_area_label(PE_ptr->output_area));
   escaped_printf(1);

   dist = PE_ptr->transit_dist;

   spaces(MARGIN + INDENT);
   if (dist == 1)
      print_head += sprintf(line_buf+print_head,
                           "Distance across this element : %d meter",
                           dist);
   else
      print_head += sprintf(line_buf+print_head,
                           "Distance across this element : %d meters",
                           dist);

   time = (int)element_transit_time(layer,PE_num);

   print_head += sprintf(line_buf+print_head,
                        "   Transit Time : %d (sec)\n\n",
                        time);
   escaped_printf(2);

   return;
}

/* ------------------------------------------------------------------------ */

print_area_data(area)
int area;

{
   spaces(MARGIN);
   print_head += sprintf(line_buf+print_head,
                        "| %-27s ",
                        get_full_ASD_area_label(area));

   print_head += sprintf(line_buf+print_head,
                        "|   %4d   ",
                        get_ASD_area_dist(area));

   print_head += sprintf(line_buf+print_head,
                        "|    %4d      ",
                        (int)area_transit_time(area));

   if (get_ASD_area_type(area) == INTERIOR)
      print_head += sprintf(line_buf+print_head,
```

```
                              "| INTERIOR  |\n");
   else
      print_head += sprintf(line_buf+print_head,
                              "| EXTERIOR  |\n");

   escaped_printf(1);

   return;
}

/* ------------------------------------------------------------------------ */

print_no_comps(delay_nondet)
int delay_nondet;

{
   spaces(MARGIN);
   if (delay_nondet == DELAY)
   {
      print_head += sprintf(line_buf+print_head,
         "|   No Delay Components Exist Within This PE   ");
      print_head += sprintf(line_buf+print_head,
         "|                                               |\n");
   }
   else
   {
      print_head += sprintf(line_buf+print_head,
         "| No Detection Components Exist Within This PE ");
      print_head += sprintf(line_buf+print_head,
         "|                                               |\n");
   }
   escaped_printf(1);

   return;
}

/* ------------------------------------------------------------------------ */

print_comp_data(delay_nondet,comp_num,layer,PE_num)
int delay_nondet,comp_num,layer,PE_num;

{
   int PE_type, delay_comp_type, nondet_comp_type, choice_num;
   char letter_char;
   float val;

   PROT_ELEM *PE_ptr;
   PE_SPECS *PE_spec_ptr;
   COMP_SPECS *comp_spec_ptr;

   PE_ptr = get_ASD_PE_ptr(layer,PE_num);
   PE_type = PE_ptr->type;
   PE_spec_ptr = get_PE_cat_spec_ptr(PE_type);
```

```
spaces(MARGIN);

if (delay_nondet == DELAY)
{
   letter_char = 'A' + comp_num;
   delay_comp_type = PE_spec_ptr->del_comp_list[comp_num].comptype;

   switch (PE_spec_ptr->del_comp_list[comp_num].location)
   {
      case  INPUT_SEG : print_head += sprintf(line_buf+print_head,
                         "| %c) INPUT %-34s  ",
                         letter_char,
                         get_delay_comp_spec_ptr(delay_comp_type)->label);
                        break;

      case OUTPUT_SEG : print_head += sprintf(line_buf+print_head,
                         "| %c) OUTPUT %-34s ",
                         letter_char,
                         get_delay_comp_spec_ptr(delay_comp_type)->label);
                        break;

      case EITHER_SEG : print_head += sprintf(line_buf+print_head,
                         "| %c) %-34s        ",
                         letter_char,
                         get_delay_comp_spec_ptr(delay_comp_type)->label);
                        break;
   }

   choice_num = get_comp_choice_num(delay_nondet,comp_num,layer,PE_num);

   switch (choice_num)
   {
      case CLEAR  : print_head += sprintf(line_buf+print_head,
                      "| CLEAR SETTING (?)                                ");
                    break;

      case DIRECT : val = PE_ptr->
                              del_comp_settings[comp_num].direct_setting;
                    print_head += sprintf(line_buf+print_head,
                      "| DIRECT SETTING (#)                    %4.0f     ",
                      val);
                    break;

      default     : letter_char = 'a' + choice_num;
                    comp_spec_ptr = get_delay_comp_spec_ptr(
                                        delay_comp_type);
                    val = comp_spec_ptr->threat_values_table[choice_num]
                                                [get_current_threat()];

                    if (val == INFINITY)
                       print_head += sprintf(line_buf+print_head,
                         "| %c) %-34s  N/A     ",
                         letter_char,
                         comp_spec_ptr->comp_choice_labels[choice_num]);
```

```
                    else
                       print_head += sprintf(line_buf+print_head,
                          "| %c) %-34s  %3.0f     ",
                          letter_char,
                          comp_spec_ptr->comp_choice_labels[choice_num],
                          val);
                    break;
      }

      if (get_delay_comp_spec_ptr(delay_comp_type)->panic_bar)
      {
         if (PE_ptr->del_comp_settings[comp_num].panic_bar)
            print_head += sprintf(line_buf+print_head,
               "|    YES    |\n");
         else
            print_head += sprintf(line_buf+print_head,
               "|    NO     |\n");
      }
      else
      {
         print_head += sprintf(line_buf+print_head,
            "|     -     |\n");
      }

      escaped_printf(1);
   }
   else
   {
      letter_char = 'A' + comp_num;
      nondet_comp_type = PE_spec_ptr->nondet_comp_list[comp_num].comptype;

      switch (PE_spec_ptr->nondet_comp_list[comp_num].location)
      {
         case  INPUT_SEG : print_head += sprintf(line_buf+print_head,
                           "| %c) INPUT %-34s  ",
                           letter_char,
                           get_nondet_comp_spec_ptr(nondet_comp_type)->label);
                          break;

         case OUTPUT_SEG : print_head += sprintf(line_buf+print_head,
                           "| %c) OUTPUT %-34s ",
                           letter_char,
                           get_nondet_comp_spec_ptr(nondet_comp_type)->label);
                          break;

         case EITHER_SEG : print_head += sprintf(line_buf+print_head,
                           "| %c) %-34s        ",
                           letter_char,
                           get_nondet_comp_spec_ptr(nondet_comp_type)->label);
                          break;
      }

      choice_num = get_comp_choice_num(delay_nondet,comp_num,layer,PE_num);
```

```
      switch (choice_num)
      {
         case CLEAR  : print_head += sprintf(line_buf+print_head,
                        "| CLEAR SETTING (?)                                  |\n");
                       escaped_printf(1);
                       break;

         case DIRECT : print_head += sprintf(line_buf+print_head,
                        "| DIRECT SETTING (#)                      %5.3f    |\n",
                        invert_det_nondet(PE_ptr->
                            nondet_comp_settings[comp_num].direct_setting));
                       escaped_printf(1);
                       break;

         default     : letter_char = 'a' + choice_num;
                       comp_spec_ptr = get_nondet_comp_spec_ptr(
                                          nondet_comp_type);
                       val = comp_spec_ptr->threat_values_table[choice_num]
                                                [get_current_threat()];

                       if (val == 0.0)
                          print_head += sprintf(line_buf+print_head,
                             "| %c) %-34s  N/A     |\n",
                             letter_char,
                             comp_spec_ptr->comp_choice_labels[choice_num]);
                       else
                          print_head += sprintf(line_buf+print_head,
                             "| %c) %-34s %5.3f    |\n",
                             letter_char,
                             comp_spec_ptr->comp_choice_labels[choice_num],
                             invert_det_nondet(val));

                       escaped_printf(1);
                       break;
      }
   }
   return;
}

/* ------------------------------------------------------------------------ */

int print_path_table_footer()

{
   print_path_table_divider();
   print_head += sprintf(line_buf+print_head,"\n");
   escaped_printf(1);

   return;
}

/* ------------------------------------------------------------------------ */

int print_area_table_footer()
```

```
{
   print_area_table_divider();
   print_head += sprintf(line_buf+print_head,"\n");
   escaped_printf(1);

   return;
}

/* ------------------------------------------------------------------------ */

int print_comp_table_footer(delay_nondet)
int delay_nondet;

{
   print_comp_table_divider(delay_nondet);
   print_head += sprintf(line_buf+print_head,"\n");
   escaped_printf(1);

   return;
}

/* ------------------------------------------------------------------------ */

int print_path_table_divider()

{
   spaces(MARGIN+INDENT);
   print_head += sprintf(line_buf+print_head,
         "+-------------");
   print_head += sprintf(line_buf+print_head,
         "+------------------------------+");
   print_head += sprintf(line_buf+print_head,
         "--------------------------------------+");
   print_head += sprintf(line_buf+print_head,
         "--------------------------------------+\n");
   escaped_printf(1);

   return;
}

/* ------------------------------------------------------------------------ */

int print_area_table_divider()

{
   spaces(MARGIN+INDENT);
   print_head += sprintf(line_buf+print_head,
         "+-----------------------------");
   print_head += sprintf(line_buf+print_head,
         "+----------+-------------");
   print_head += sprintf(line_buf+print_head,
         "+-----------+\n");
   escaped_printf(1);
```

```
   return;
}

/* ---------------------------------------------------------------------- */

int print_comp_table_divider(delay_nondet)
int delay_nondet;

{
   spaces(MARGIN+INDENT);

   if (delay_nondet == DELAY)
   {
      print_head += sprintf(line_buf+print_head,
            "+-------------------------------------------");
      print_head += sprintf(line_buf+print_head,
            "+-------------------------------------------");
      print_head += sprintf(line_buf+print_head,
            "+-----------+\n");
   }
   else
   {
      print_head += sprintf(line_buf+print_head,
            "+-------------------------------------------");
      print_head += sprintf(line_buf+print_head,
            "+-------------------------------------------+\n");
   }
   escaped_printf(1);

   return;
}

/* ---------------------------------------------------------------------- */

int print_bypass_information(layer)
int layer;

{
   int i;

   request_lines(3,TABLES_TITLE);
   spaces(MARGIN);
   print_head += sprintf(line_buf+print_head,
         "The %sProtection Layer is Bypassed.\n",
         cardinal_number(layer+1));
   escaped_printf(1);
   spaces(MARGIN);
   print_head += sprintf(line_buf+print_head,
         "%s is equivalent to ",
         get_full_ASD_area_label(layer));
   print_head += sprintf(line_buf+print_head,
         "%s.\n",
         get_full_ASD_area_label(layer+1));
```

```
   escaped_printf(1);
   print_head += sprintf(line_buf+print_head,"\n");
   escaped_printf(1);

   return;
}

/* ------------------------------------------------------------------------ */

int spaces(len)
int len;

{
   int i;

   for (i=0;i < len; i++)
      print_head += sprintf(line_buf+print_head," ");

   return;
}

/* ------------------------------------------------------------------------ */

/* center loads the given string into the line buffer with a carriage return */

center(string)
char *string;

{
   int len,i;

   len = (PAGE_WIDTH - strlen(string))/2;
   for (i=0; i<len; i++)
      spaces(1);
   print_head += sprintf(line_buf+print_head,"%s\n",string);
   return;
}

/* ------------------------------------------------------------------------ */

/* request_lines determines if there is enough space on the current page to  */
/* print 'count' number of lines.  If there IS enough room, the requested    */
/* number of lines is added to the line count for the current page and       */
/* control is returned to the calling function for the actual printing of    */
/* the requested lines.  If there ISN'T room, a form feed is issued and the  */
/* given title is printed.  Control is returned for printing requested lines */

int request_lines(count,title_str)
int count;
char *title_str;

{
   if ( (line_count+count >= (LINES_PER_PAGE-BOTTOM_MARGIN-2)) && (!escaped) )
     {
```

```
      form_feed();
      print_title(title_str);
      }

   line_count += count;

   return;
}

/* ------------------------------------------------------------------------ */

int form_feed()

{
   int i;

   line_feed(LINES_PER_PAGE-BOTTOM_MARGIN-lines_used-2);

   line_count=0;
   lines_used=0;

   if (!escaped)
   {
      spaces((PAGE_WIDTH - 6)/2);
      escaped_printf(0);
      fprintf(stream,"Page %1d\n\n",page_num);
      line_feed(BOTTOM_MARGIN);
   }
   else
   {
      fprintf(stream,"\n\n");
      line_feed(BOTTOM_MARGIN);
   }

   return;
}

/* ------------------------------------------------------------------------ */

/* line_feed sends num_lines number of line feeds to the current stream      */

line_feed(num_lines)
int num_lines;

{
   int i;

   for (i = 0; i < num_lines; i++) fprintf(stream,"\n");

   return;
}

/* ------------------------------------------------------------------------ */
```

```
int set_num_copies(number)
int number;

{
   num_copies = number;
   return;
}

/* ------------------------------------------------------------------------ */

escaped_printf(lines)
int lines;

{
   print_head = 0;
   if (!escaped)
   {
      lines_used = lines_used + lines;
      fprintf(stream,"%s",line_buf);
      escaped = search_key_buf(ESC);
   }

   return;
}

/* ------------------------------------------------------------------------ */

/* set_current_stream sets the current stream for subsequent output to the   */
/* given file ("stdprn" will set for output to the standard line printer)    */

set_current_output(output_file)
char *output_file;

{
   if (strcmp(output_file,"stdprn") == 0)
      stream = stdprn;
   else
      stream = fopen(output_file,"a");

   return;
}

/* ------------------------------------------------------------------------ */

savi_print_quit()

{
   dump_key_buffer();

   /* return a -1 to return to main menu level */
   return(-1);
}

/* ********************** end SAVIPRIN.C source file ********************** */
```

```
/* ************************ SAVIQUIT.C source file ************************** */

#include <wfd.h>
#include <savidefs.h>

/* ------------------------------------------------------------------------- */

/* savi_quit_no simply returns the user back to the main menu                */

savi_quit_no()

{
   dump_key_buffer();

   /* return a 1 to jump up one menu level */
   return(1);
}

/* ------------------------------------------------------------------------- */

/* savi_quit_yes sets the savi done switch to quit the main menu loop and    */
/* returns a -2 to quit the Windows for Data menu handling function          */

savi_quit_yes()

{
   int quit_ok;

   if (get_current_ASD_saved())
     quit_ok = TRUE;
   else
   {
     /* offer the user the chance to save the current ASD */
     /* prior to quitting SAVI                            */
     if (offer_file_save("Before Leaving SAVI ?"))
     {
        /* user has chosen to save or not save ASD */
        /* ok to quit SAVI                         */
        quit_ok = TRUE;
     }
     else
        /* user has escaped from save offer */
        /* do not quit SAVI                 */
        quit_ok = FALSE;
   }

   dump_key_buffer();

   if (quit_ok)
   {
      set_savi_done();

      /* return a -2 to quit the main menu */
```

```
      return(-2);
   }
   else
      return(0);
}

/* ------------------------------------------------------------------------ */

/* offer_file_save offers the user the chance to save the current ASD.      */
/* The given message string is displayed in the offer window.               */
/* Note: the message string cannot exceed 27 printable characters           */
/* If the user accepts the offer (ie. wishes to save the current ASD) then  */
/* he is given the opportunity to save the current ASD.                     */
/* If the user escapes from the window which offers the option to save the  */
/* current ASD, or if the user escapes from the save window, then this      */
/* function returns FALSE, otherwise it returns TRUE                        */

offer_file_save(messag)
char *messag;

{
   int key, save_ASD, done, escaped;

   save_ASD = TRUE;

   /* display the save ASD window */
   popup_save_ASD_offer_window(messag);

   /* highlight the save setting within the window */
   highlight_save_ASD_setting(save_ASD);

   done = FALSE;
   escaped = FALSE;

   /* process key press */
   while (!done)
   {
      /* dump the keyboard input buffer */
      dump_key_buffer();

      /* wait for a key press */
      key = ki();

      /* process key */
      switch (-key)
      {
         /* LEFT and UP arrows set choice to YES (TRUE) */
         case K_LEFT   : ;
         case K_UP     : save_ASD = TRUE;
                         highlight_save_ASD_setting(save_ASD);
                         break;

         /* RIGHT and DOWN arrows set choice to NO (FALSE) */
         case K_RIGHT  : ;
```

```
            case K_DN      : save_ASD = FALSE;
                             highlight_save_ASD_setting(save_ASD);
                             break;

            /* ENTER (Return) key */
            /* exit with current save choice */
            case -K_ENTER  : done = TRUE;
                             break;

            /* Y and y keys */
            /* set save ASD to YES (TRUE) and exit */
            case -'Y'      : ;
            case -'y'      : save_ASD = TRUE;
                             done = TRUE;
                             break;

            /* N and n keys */
            /* set save ASD to NO (FALSE) and exit */
            case -'N'      : ;
            case -'n'      : save_ASD = FALSE;
                             done = TRUE;
                             break;

            /* ESCAPE key */
            /* exit and indicate that user escaped */
            case -K_ESC    : save_ASD = FALSE;
                             done = TRUE;
                             escaped = TRUE;
                             break;

            default        : bell();
                             break;
         }
      }

      remove_save_ASD_offer_window();

      if (save_ASD)
      {
         /* allow user to save current ASD */
         if (savi_file_save() == 0)
            /* user escaped from save window */
            /* or some failure occurred in   */
            /* the save attempt              */
            escaped = TRUE;
      }

      return(!escaped);
   }

/* ********************** end SAVIQUIT.C source file ********************** */

/* ************************ SAVITHRE.C source file ************************ */
```

```
#include <wfd.h>
#include <savidefs.h>

#define THREAT_CHOICE_MIN_ROW 3
#define THREAT_CHOICES 3

WINDOW *get_threat_attributes_window_ptr();

/* ------------------------------------------------------------------------ */

/* savi_threat_attributes allows the user to select the current             */
/* threat attributes by choosing a threat from the threat table             */

savi_threat_attributes()

{
   int current_choice, done, return_code, page_size,
       current_threat, start_threat, end_threat, begin_threat;
   WINDOW *wn_ptr;

   page_size = THREAT_CHOICES - 1;

   /* display the threat attributes window */
   popup_threat_attributes_window();

   current_threat = get_current_threat();

   begin_threat = current_threat;

   /* determine initial choice and limits on window into threat table */
   if (current_threat > NUM_VALID_THREATS - THREAT_CHOICES)
   /* start with current choice within last page of threat table */
   {
      start_threat = max(0, NUM_VALID_THREATS - THREAT_CHOICES);
      current_choice = current_threat - start_threat;
   }
   else
   /* start with current choice at top of appropriate page */
   {
      start_threat = current_threat;
      current_choice = 0;
   }

   end_threat = get_end_num_in_range(start_threat, NUM_VALID_THREATS,
                                     THREAT_CHOICES);

   display_threat_attributes(start_threat,end_threat,NUM_VALID_THREATS-1);

   mark_threat(current_threat, start_threat, end_threat);

   done = FALSE;

   wn_ptr = get_threat_attributes_window_ptr();
```

```
while (!done)
{
   current_choice = get_user_choice(THREAT_CHOICE_MIN_ROW,
                                    THREAT_CHOICES,
                                    current_choice, wn_ptr,
                                    &return_code);

   switch (return_code)
   {
      /* escape from define threat attributes function */
      case -K_ESC  : done = TRUE;
                     break;

      /* move up one line above current page if possible */
      case -K_UP   : if (start_threat > 0)
                     {
                        start_threat--;
                        end_threat = get_end_num_in_range(start_threat,
                                                          NUM_VALID_THREATS,
                                                          THREAT_CHOICES);
                        display_threat_attributes(start_threat,end_threat,
                                                  NUM_VALID_THREATS-1);

                        mark_threat(current_threat, start_threat,
                                    end_threat);
                     }
                     break;

      /* move down one line below current page if possible */
      case -K_DN   : if (end_threat < NUM_VALID_THREATS - 1)
                     {
                        start_threat++;
                        end_threat = get_end_num_in_range(start_threat,
                                                          NUM_VALID_THREATS,
                                                          THREAT_CHOICES);
                        display_threat_attributes(start_threat,end_threat,
                                                  NUM_VALID_THREATS-1);

                        mark_threat(current_threat, start_threat,
                                    end_threat);
                     }
                     break;

      /* Display top page in choice list */
      /* with current choice at the top  */
      case -K_HOME : start_threat = 0;
                     end_threat = get_end_num_in_range(start_threat,
                                                       NUM_VALID_THREATS,
                                                       THREAT_CHOICES);
                     current_choice = 0;
                     display_threat_attributes(start_threat,end_threat,
                                               NUM_VALID_THREATS-1);

                     mark_threat(current_threat, start_threat,
```

```
                                end_threat);
                    break;

    /* Display bottom page in choice list */
    /* with current choice at the bottom  */
    case -K_END  : start_threat = max(0,NUM_VALID_THREATS-THREAT_CHOICES);
                    end_threat = NUM_VALID_THREATS - 1;
                    current_choice = end_threat - start_threat;
                    display_threat_attributes(start_threat,end_threat,
                                              NUM_VALID_THREATS-1);

                    mark_threat(current_threat, start_threat,
                                end_threat);
                    break;

    /* Page Up THREAT_CHOICES-1 lines or to top of choice list */
    case -K_PGUP : /* if the top of the list is less than one page  */
                    /* above the current page top, then adjust the   */
                    /* new current_choice within the new top page    */
                    if (start_threat < page_size)
                    {
                       current_choice = current_choice +
                                        (start_threat - page_size);
                       if (current_choice < 0) current_choice = 0;
                       start_threat = 0;
                    }
                    else
                       start_threat = start_threat - page_size;

                    end_threat = get_end_num_in_range(start_threat,
                                                      NUM_VALID_THREATS,
                                                      THREAT_CHOICES);

                    display_threat_attributes(start_threat,end_threat,
                                              NUM_VALID_THREATS-1);

                    mark_threat(current_threat, start_threat,
                                end_threat);

                    break;

    /* Page Down THREAT_CHOICES-1 lines or to bottom of choice list */
    case -K_PGDN : /* if the bottom of the list is on the current page */
                    /* then just move the current_choice to the bottom  */
                    /* of the list                                      */
                    if (end_threat == NUM_VALID_THREATS - 1)
                       current_choice = end_threat - start_threat;
                    else
                    /* if the bottom of the list is less than one page  */
                    /* below the current page bottom, then adjust the   */
                    /* new current_choice within the new bottom page    */
                    if ((end_threat + page_size) > NUM_VALID_THREATS - 1)
                    {
                       current_choice = current_choice + page_size -
```

```
                                    (NUM_VALID_THREATS - 1 - end_threat);
                        if (current_choice > THREAT_CHOICES - 1)
                           current_choice = THREAT_CHOICES - 1;
                     }

                     end_threat = min(NUM_VALID_THREATS - 1,
                                      end_threat + page_size);

                     start_threat = max(0,end_threat + 1 - THREAT_CHOICES);

                     display_threat_attributes(start_threat,end_threat,
                                               NUM_VALID_THREATS-1);

                     mark_threat(current_threat, start_threat,
                                 end_threat);

                     break;

            /* user has selected a threat */
            case  0      :  current_threat = current_choice + start_threat;
                            set_current_threat(current_threat);
                            display_threat_attributes(start_threat,end_threat,
                                                      NUM_VALID_THREATS-1);
                            mark_threat(current_threat, start_threat,
                                        end_threat);
                            done = TRUE;
                            break;

            default      :  bell();
                            break;
         }
      }

      remove_threat_attributes_window();

      /* if the threat was changed then set the vulnerability false */
      if (current_threat != begin_threat)
         set_vulnerability_current(FALSE);

      /* insert all HEL PEs if the threat transport mode is */
      /* a helicopter and delete all HEL PEs if not         */
      if (get_threat_transport_mode(current_threat) == HELICOPTER)
         insert_all_helicopter_PEs();
      else
         delete_all_helicopter_PEs();

      return(0);
   }

   /* ------------------------------------------------------------------------ */

   /* insert_all_helicopter_PEs inserts all PEs which have the type helicopter  */
   /* into the current ASD and displays them on the screen                      */
```

```
insert_all_helicopter_PEs()

{
   PROT_ELEM *PE_ptr;
   int layer, PE_num, inserted_one;

   for (layer = 0; layer < MAX_LAYERS_PER_ASD; layer++)
   {
      inserted_one = FALSE;

      for (PE_num = 0; PE_num < MAX_PE_PER_LAYER; PE_num++)
      {
         PE_ptr = get_ASD_PE_ptr(layer, PE_num);

         if ( (get_PE_cat_spec_ptr(PE_ptr->type)->helicopter) &&
              (!PE_ptr->present) )
         {
            PE_ptr->present = TRUE;
            display_PE(PE_ptr,layer,PE_num);
            inserted_one = TRUE;
            set_vulnerability_current(FALSE);
         }
      }

      /* unset the bypass for this layer if */
      /* at least one HEL PE was inserted   */
      /* onto a previously bypassed layer   */
      if (inserted_one && get_ASD_layer_bypass(layer))
      {
         set_ASD_layer_bypass(layer,FALSE);
         erase_layer_bypass(layer);
      }
   }

   return;
}

/* ------------------------------------------------------------------------ */

/* delete_all_helicopter_PEs deletes all PEs which have the type helicopter  */
/* from the current ASD and erases them from the screen                      */

delete_all_helicopter_PEs()

{
   PROT_ELEM *PE_ptr;
   int layer, PE_num, deleted_one;

   for (layer = 0; layer < MAX_LAYERS_PER_ASD; layer++)
   {
      deleted_one = FALSE;

      for (PE_num = 0; PE_num < MAX_DISPLAY_PE_PER_LAYER; PE_num++)
      {
```

```
         PE_ptr = get_ASD_PE_ptr(layer, PE_num);

         if ( (get_PE_cat_spec_ptr(PE_ptr->type)->helicopter) &&
              (PE_ptr->present) )
         {
            PE_ptr->present = FALSE;
            erase_PE(layer,PE_num);
            deleted_one = TRUE;
            set_vulnerability_current(FALSE);
         }
      }

      /* set the bypass for this layer if     */
      /* at least one HEL PE was deleted      */
      /* leaving no present PEs on this layer */
      if (deleted_one && (number_of_PEs_present(layer) == 0) )
      {
         set_ASD_layer_bypass(layer,TRUE);
         display_layer_bypass(layer);
      }

   }

   return;
}

/* ------------------------------------------------------------------------ */

/* savi_threat_objective allows the user to select the current              */
/* threat objective (entry only (sabotage) or entry/exit (theft))           */

savi_threat_objective()

{
   int key, current_direction, start_direction, marked_direction, done;

   current_direction = get_current_direction();
   start_direction = current_direction;

   /* display the threat objective window */
   popup_threat_objective_window();

   /* mark the current objective */
   mark_objective(current_direction);
   marked_direction = current_direction;

   /* highlight the current objective within the window */
   highlight_threat_objective(current_direction);

   done = FALSE;

   /* process key press */
   while (!done)
   {
```

```
/* dump the keyboard input buffer */
dump_key_buffer();

/* wait for a key press */
key = ki();

/* process key */
/* Note: ki() returns the negative of the key scan code  */
/* if an extended code key is pressed (ie: UP, DOWN etc) */
/* and returns the ASCII code directly otherwise         */
switch (-key)
{
   /* LEFT, UP, Home and PgUp highlight objective sabotage (ENTRY) */
   case K_LEFT   : ;
   case K_UP     : ;
   case K_HOME   : ;
   case K_PGUP   : if (current_direction != ENTRY)
                   {
                      unhighlight_threat_objective(current_direction);
                      current_direction = ENTRY;
                      highlight_threat_objective(current_direction);
                   }
                   break;

   /* RIGHT, DOWN, End and PgDn highlight objective theft (EXIT) */
   case K_RIGHT  : ;
   case K_DN     : ;
   case K_END    : ;
   case K_PGDN   : if (current_direction != EXIT)
                   {
                      unhighlight_threat_objective(current_direction);
                      current_direction = EXIT;
                      highlight_threat_objective(current_direction);
                   }
                   break;

   /* ESCAPE key exits with current direction */
   case -K_ESC    : done = TRUE;
                    break;

   /* ENTER key sets the current direction and exits */
   case -K_ENTER  : if (current_direction != marked_direction)
                    {
                       unmark_objective(marked_direction);
                       set_current_direction(current_direction);
                       mark_objective(current_direction);
                       marked_direction = current_direction;
                       highlight_threat_objective(current_direction);
                    }
                    done = TRUE;
                    break;

   default        : bell();
                    break;
```

```
      )
   )

   /* if the direction was changed then set the vulnerability false */
   if (marked_direction != start_direction)
      set_vulnerability_current(FALSE);

   remove_threat_objective_window();

   return(0);
}

/* ------------------------------------------------------------------------ */

/* savi_threat_strategy allows the user to select the current threat        */
/* strategy (FORCE) or (MIXED)                                              */

savi_threat_strategy()

{
   int key, current_strategy, start_strategy, marked_strategy, done;

   current_strategy = get_current_strategy();
   start_strategy = current_strategy;

   /* display the threat strategy window */
   popup_threat_strategy_window();

   /* mark the current strategy */
   mark_strategy(current_strategy);
   marked_strategy = current_strategy;

   /* highlight the current strategy within the window */
   highlight_threat_strategy(current_strategy);

   done = FALSE;

   /* process key press */
   while (!done)
   {
      /* dump the keyboard input buffer */
      dump_key_buffer();

      /* wait for a key press */
      key = ki();

      /* process key */
      /* Note: ki() returns the negative of the key scan code  */
      /* if an extended code key is pressed (ie: UP, DOWN etc) */
      /* and returns the ASCII code directly otherwise         */
      switch (-key)
      {
         /* LEFT, UP, Home and PgUp highlight strategy MIXED */
         case K_LEFT   : ;
```

```
case K_UP      : ;
case K_HOME    : ;
case K_PGUP    : if (current_strategy != MIXED)
                 {
                    unhighlight_threat_strategy(current_strategy);
                    current_strategy = MIXED;
                    highlight_threat_strategy(current_strategy);
                 }
                 break;

/* RIGHT, DOWN, End and PgDn highlight strategy FORCE */
case K_RIGHT   : ;
case K_DN      : ;
case K_END     : ;
case K_PGDN    : if (current_strategy != FORCE)
                 {
                    unhighlight_threat_strategy(current_strategy);
                    current_strategy = FORCE;
                    highlight_threat_strategy(current_strategy);
                 }
                 break;

/* ESCAPE key exits with current strategy */
case -K_ESC    : done = TRUE;
                 break;

/* ENTER key sets the current strategy and exits */
case -K_ENTER  : if (current_strategy != marked_strategy)
                 {
                    unmark_strategy(marked_strategy);
                    set_current_strategy(current_strategy);
                    mark_strategy(current_strategy);
                    marked_strategy = current_strategy;
                    highlight_threat_strategy(current_strategy);
                 }
                 done = TRUE;
                 break;

/* F and f keys */
/* set strategy to FORCE and exit */
case -'F'      : ;
case -'f'      : if (current_strategy != FORCE)
                 {
                    unhighlight_threat_strategy(current_strategy);
                    current_strategy = FORCE;
                 }
                 if (current_strategy != marked_strategy)
                 {
                    unmark_strategy(marked_strategy);
                    set_current_strategy(current_strategy);
                    mark_strategy(current_strategy);
                    marked_strategy = current_strategy;
                 }
                 highlight_threat_strategy(current_strategy);
```

```
                        done = TRUE;
                        break;

         /* M and m keys */
         /* set strategy to MIXED and exit */
         case -'M'      : ;
         case -'m'      : if (current_strategy != MIXED)
                          {
                             unhighlight_threat_strategy(current_strategy);
                             current_strategy = MIXED;
                          }
                          if (current_strategy != marked_strategy)
                          {
                             unmark_strategy(marked_strategy);
                             set_current_strategy(current_strategy);
                             mark_strategy(current_strategy);
                             marked_strategy = current_strategy;
                          }
                          highlight_threat_strategy(current_strategy);
                          done = TRUE;
                          break;

         default        : bell();
                          break;
      }
   }

   /* if the strategy was changed then set the vulnerability false */
   if (marked_strategy != start_strategy)
      set_vulnerability_current(FALSE);

   remove_threat_strategy_window();

   return(0);
}

/* ------------------------------------------------------------------------ */

savi_threat_quit()

{
   /* return a 1 to jump up one menu level */
   return(1);
}

/* ********************** end SAVITHRE.C source file ********************** */

/* ************************ SAVIDIST.C source file ************************ */

#include <wfd.h>
#include <savidefs.h>

/* ------------------------------------------------------------------------ */
```

```
/* savi_define_distance_areas allows the user to set the transit distance    */
/* across the areas in the current ASD                                       */

savi_define_distance_areas()

{
   int area_num, key, done, window_on;

   area_num = 1;

   done = FALSE;
   window_on = TRUE;

   /* highlight the first modifiable area */
   /* and display the distance setting    */
   highlight_area_label(area_num);
   popup_area_transit_input_window();
   display_area_transit_val(area_num);

   /* process key press */
   while (!done)
   {
      /* dump the keyboard input buffer */
      dump_key_buffer();

      /* wait for a key press */
      key = ki();

      /* process key */
      /* Note: ki() returns the negative of the key scan code  */
      /* if an extended code key is pressed (ie: UP, DOWN etc) */
      /* and returns the ASCII code directly otherwise         */
      switch (-key)
      {
         /* LEFT and UP arrows */
         /* highlight the area label above current area */
         /* do not wrap around top to bottom */
         case K_LEFT  : ;
         case K_UP    : if (area_num > 1)
                        {
                           unhighlight_area_label(area_num);
                           area_num--;

                           if (window_on)
                           {
                              popup_area_transit_input_window();
                              display_area_transit_val(area_num);
                           }

                           highlight_area_label(area_num);
                        }
                        else
                           bell();
```

```
                break;

/* RIGHT and DOWN arrows */
/* highlight the area label below current area */
/* do not wrap around bottom to top */
case K_RIGHT : ;
case K_DN    : if (area_num < MAX_MODIFIABLE_AREAS)
               {
                  unhighlight_area_label(area_num);
                  area_num++;

                  if (window_on)
                  {
                     popup_area_transit_input_window();
                     display_area_transit_val(area_num);
                  }

                  highlight_area_label(area_num);
               }
               else
                  bell();

               break;

/* HOME and PAGE UP keys */
/* highlight the first modifiable area label */
case K_HOME  : ;
case K_PGUP  : if (area_num != 1)
               {
                  unhighlight_area_label(area_num);
                  area_num = 1;

                  if (window_on)
                  {
                     popup_area_transit_input_window();
                     display_area_transit_val(area_num);
                  }

                  highlight_area_label(area_num);
               }
               else
                  bell();

               break;

/* END and PAGE DOWN keys */
/* highlight the last modifiable area label */
case K_END   : ;
case K_PGDN  : if (area_num != MAX_MODIFIABLE_AREAS)
               {
                  unhighlight_area_label(area_num);
                  area_num = MAX_MODIFIABLE_AREAS;

                  if (window_on)
```

```
                        {
                           popup_area_transit_input_window();
                           display_area_transit_val(area_num);
                        }

                        highlight_area_label(area_num);
                     }
                     else
                        bell();

                     break;

      /* ENTER (Return) key */
      /* allow user to set transit value on current area */
      case -K_ENTER : if (!window_on)
                      {
                         popup_area_transit_input_window();
                         display_area_transit_val(area_num);
                      }

                      input_area_transit_value(area_num);

                      display_area_transit_val(area_num);

                      if (!window_on)
                         remove_area_transit_input_window();

                      break;

      /* SPACE bar */
      /* toggle the window display */
      case -K_SPACE : if (popup_window_on_screen())
                      {
                         window_on = FALSE;
                         remove_area_transit_input_window();
                      }
                      else
                      {
                         window_on = TRUE;
                         popup_area_transit_input_window();
                         display_area_transit_val(area_num);
                      }

                      break;

      /* ESCAPE key */
      case -K_ESC   : done = TRUE;
                      break;

      default       : bell();
                      break;

   }
}
```

```
   /* remove the transit input window and unhighlight the current area label */
   remove_area_transit_input_window();
   unhighlight_area_label(area_num);

   dump_key_buffer();

   return(0);
}

/* ------------------------------------------------------------------------ */

/* input_area_transit_value allows the user to set the distance across the   */
/* given area in meters                                                      */

input_area_transit_value(area_num)
int area_num;

{
   DFIELDPTR field_ptr;
   int new_dist, start_dist;

   start_dist = get_ASD_area_dist(area_num);
   new_dist = start_dist;

   /* set the input form fields location and size */
   mod_wn(13,33,1,6,STDWNP);

   /* set the input form to have no border */
   sw_border(BDR_OP,STDWNP);

   /* define the form input field        */
   /* independent of the prompt location */
   field_ptr = def_fld(0,0,"","9999",
                       F_INT,&new_dist);
   sf_location(0,8,field_ptr);
   sf_atts(LWHITBLUE,LHIGHLITE,field_ptr);

   /* accept input from user */
   rd_flds();

   /* set the new distance in the given area */
   set_ASD_area_dist(area_num,new_dist);

   if (new_dist != start_dist)
      set_vulnerability_current(FALSE);

   return;
}

/* ------------------------------------------------------------------------ */

/* savi_define_distance_elements allows the user to set the transit distance */
/* across any PE in the current ASD                                          */
```

```
savi_define_distance_elements()

{
   int layer, layer_temp, PE_num, PE_temp, key, done, window_on;
   PROT_ELEM *PE_ptr;

   /* set the layer to the first layer in the ASD with present PEs */
   layer = first_layer_with_PEs_present();

   /* set the PE number to the first present PE on the set layer */
   PE_num = first_present_PE_num_on_layer(layer);

   done = FALSE;
   window_on = TRUE;

   /* display the transit input window and highlight the first PE */
   popup_PE_transit_input_window(layer,PE_num);
   display_PE_transit_val(layer,PE_num);
   highlight_PE(layer,PE_num);

   /* process key press */
   while (!done)
   {
      /* dump the keyboard input buffer */
      dump_key_buffer();

      /* wait for a key press */
      key = ki();

      /* process key */
      /* Note: ki() returns the negative of the key scan code  */
      /* if an extended code key is pressed (ie: UP, DOWN etc) */
      /* and returns the ASCII code directly otherwise         */
      switch (-key)
      {
         /* UP arrow */
         /* highlight PE above current PE */
         /* do not wrap around top to bottom */
         /* move to first layer above the current layer */
         /* with present PEs                            */
         /* if new PE is non-present or null then move to */
         /* the closest present PE on the new layer       */
         case K_UP    : unhighlight_PE(layer,PE_num);
                        layer_temp = layer_with_PEs_present_above(layer);
                        if ( (layer > 0) && (layer_temp != layer) )
                           {
                              layer = layer_temp;
                              PE_ptr = get_ASD_PE_ptr(layer,PE_num);
                              if (!(PE_ptr->present) || (PE_ptr->type == 0))
                                 PE_num = closest_present_PE_num(layer,PE_num
                           }
                        else
                           bell();
```

```
                dump_key_buffer();

                if (window_on)
                {
                   popup_PE_transit_input_window(layer,PE_num);
                   display_PE_transit_val(layer,PE_num);
                }
                highlight_PE(layer,PE_num);
                break;

/* DOWN arrow */
/* highlight PE below current PE */
/* do not wrap around bottom to top */
/* move to first layer below the current layer */
/* with present PEs                            */
/* if new PE is non-present or null then move to */
/* the closest present PE on the new layer       */
case K_DN    : unhighlight_PE(layer,PE_num);
                layer_temp = layer_with_PEs_present_below(layer);
                if ( (layer < MAX_LAYERS_PER_ASD) &&
                     (layer_temp != layer) )
                   {
                      layer = layer_temp;
                      PE_ptr = get_ASD_PE_ptr(layer,PE_num);
                      if (layer == MAX_LAYERS_PER_ASD) PE_num = 0;
                      if (!(PE_ptr->present) || (PE_ptr->type == 0))
                         PE_num = closest_present_PE_num(layer,PE_num);
                   }
                else
                   bell();

                dump_key_buffer();

                if (window_on)
                {
                   popup_PE_transit_input_window(layer,PE_num);
                   display_PE_transit_val(layer,PE_num);
                }
                highlight_PE(layer,PE_num);
                break;

/* LEFT arrow */
/* highlight present PE to the left of current PE */
/* wrap around to last present PE on right */
case K_LEFT  : unhighlight_PE(layer,PE_num);
                PE_temp = closest_left_present_PE_num(layer,PE_num);
                if ( (PE_num > 0) && (PE_temp != PE_num) )
                   PE_num = PE_temp;
                else
                   PE_num = last_present_PE_num_on_layer(layer);

                dump_key_buffer();
```

```
                if (window_on)
                {
                   popup_PE_transit_input_window(layer,PE_num);
                   display_PE_transit_val(layer,PE_num);
                }
                highlight_PE(layer,PE_num);
                break;

/* RIGHT arrow */
/* highlight present PE to the right of current PE */
/* wrap around to first present PE on left */
case K_RIGHT : unhighlight_PE(layer,PE_num);
                PE_temp = closest_right_present_PE_num(layer,PE_num);
                if ( (PE_num < MAX_PE_PER_LAYER-1) &&
                     (PE_temp != PE_num) )
                   PE_num = PE_temp;
                else
                   PE_num = first_present_PE_num_on_layer(layer);

                dump_key_buffer();

                if (window_on)
                {
                   popup_PE_transit_input_window(layer,PE_num);
                   display_PE_transit_val(layer,PE_num);
                }
                highlight_PE(layer,PE_num);
                break;

/* HOME key */
/* highlight the first present PE */
case K_HOME  : PE_temp = first_present_PE_num_on_layer(layer);
                if (PE_num == PE_temp) break;
                unhighlight_PE(layer,PE_num);
                PE_num = PE_temp;

                dump_key_buffer();

                if (window_on)
                {
                   popup_PE_transit_input_window(layer,PE_num);
                   display_PE_transit_val(layer,PE_num);
                }
                highlight_PE(layer,PE_num);
                break;

/* END key */
/* highlight the last present PE */
case K_END   : PE_temp = last_present_PE_num_on_layer(layer);
                if (PE_num == PE_temp) break;
                unhighlight_PE(layer,PE_num);
                PE_num = PE_temp;

                dump_key_buffer();
```

```
                if (window_on)
                {
                   popup_PE_transit_input_window(layer,PE_num);
                   display_PE_transit_val(layer,PE_num);
                }
                highlight_PE(layer,PE_num);
                break;

/* PAGE UP key */
/* highlight present PE on highest layer with present PEs */
case K_PGUP  : unhighlight_PE(layer,PE_num);
                layer_temp = first_layer_with_PEs_present();
                if ( (layer > 0) && (layer_temp != layer) )
                   {
                      layer = layer_temp;
                      PE_ptr = get_ASD_PE_ptr(layer,PE_num);
                      if (!(PE_ptr->present) || (PE_ptr->type == 0))
                         PE_num = closest_present_PE_num(layer,PE_num);
                   }
                else
                   bell();

                dump_key_buffer();

                if (window_on)
                {
                   popup_PE_transit_input_window(layer,PE_num);
                   display_PE_transit_val(layer,PE_num);
                }
                highlight_PE(layer,PE_num);
                break;

/* PAGE DOWN key */
/* highlight present PE on lowest layer with present PEs */
case K_PGDN  : unhighlight_PE(layer,PE_num);
                layer_temp = last_layer_with_PEs_present();
                if ( (layer < MAX_LAYERS_PER_ASD) &&
                     (layer_temp != layer) )
                   {
                      layer = layer_temp;
                      PE_ptr = get_ASD_PE_ptr(layer,PE_num);
                      if (layer == MAX_LAYERS_PER_ASD) PE_num = 0;
                      if (!(PE_ptr->present) || (PE_ptr->type == 0))
                         PE_num = closest_present_PE_num(layer,PE_num);
                   }
                else
                   bell();

                dump_key_buffer();

                if (window_on)
                {
                   popup_PE_transit_input_window(layer,PE_num);
```

```
                        display_PE_transit_val(layer,PE_num);
                    }
                    highlight_PE(layer,PE_num);
                    break;

        /* ENTER (Return) key */
        /* allow user to set transit value on current PE */
        case -K_ENTER : if (!window_on)
                        {
                            popup_PE_transit_input_window(layer,PE_num);
                            display_PE_transit_val(layer,PE_num);
                        }

                        input_PE_transit_value(layer,PE_num);

                        display_PE_transit_val(layer,PE_num);

                        if (!window_on)
                            remove_PE_transit_input_window();

                        break;

        /* SPACE bar */
        /* toggle the window display */
        case -K_SPACE : if (popup_window_on_screen())
                        {
                            window_on = FALSE;
                            remove_PE_transit_input_window();
                        }
                        else
                        {
                            window_on = TRUE;
                            popup_PE_transit_input_window(layer,PE_num);
                            display_PE_transit_val(layer,PE_num);
                        }

                        break;

        /* ESCAPE key */
        case -K_ESC   : done = TRUE;
                        break;

        default       : bell();
                        break;

    }
}

/* unhighlight the current PE and remove the transit input window */
unhighlight_PE(layer,PE_num);
remove_PE_transit_input_window();

dump_key_buffer();
```

```
   return(0);
}

/* ------------------------------------------------------------------------ */

/* input_PE_transit_value allows the user to set the distance across the     */
/* given PE in meters                                                        */

input_PE_transit_value(layer,PE_num)
int layer, PE_num;

{
   PROT_ELEM *PE_ptr;
   DFIELDPTR field_ptr;
   int new_dist, start_dist;

   PE_ptr = get_ASD_PE_ptr(layer,PE_num);
   start_dist = PE_ptr->transit_dist;
   new_dist = start_dist;

   /* set the input form fields location and size */
   mod_wn(10,get_popup_col()+3,1,6,STDWNP);

   /* set the input form to have no border */
   sw_border(BDR_OP,STDWNP);

   /* define the form input field        */
   /* independent of the prompt location */
   field_ptr = def_fld(0,0,"","9999",
                       F_INT,&new_dist);
   sf_location(0,8,field_ptr);
   sf_atts(LWHITBLUE,LHIGHLITE,field_ptr);

   /* accept input from user */
   rd_flds();

   /* set the new distance in the given PE */
   PE_ptr->transit_dist = new_dist;

   if (new_dist != start_dist)
      set_vulnerability_current(FALSE);

   return;
}

/* ------------------------------------------------------------------------ */

/* combined_area_transit_time returns the time in seconds necessary to       */
/* transit the given start area based on the area type. If the given area is */
/* connected to any other areas by bypasses then the transit time includes   */
/* the transit time for these areas also. Note: segment must be INPUT_SEG    */
/* or OUTPUT_SEG to indicate which path segment of the PE in question is     */
/* connected to the given start area.                                        */
```

```
float combined_area_transit_time(start_area_num, segment)
int start_area_num, segment;

{
   int area_step, layer_step, area_num, done;
   float combined_time;

   if (segment == INPUT_SEG)
   {
      area_step = -1;
      layer_step = -1;
   }
   else
   {
      area_step = 1;
      layer_step = 0;
   }

   area_num = start_area_num;
   combined_time = 0.0;
   done = FALSE;

   while (!done)
   {
      combined_time += area_transit_time(area_num);

      if ( (area_num == 0) || (area_num == MAX_AREAS_PER_ASD-1) ||
           !get_ASD_layer_bypass(area_num + layer_step) )
         done = TRUE;
      else
         area_num += area_step;
   }

   return(combined_time);
}

/* ------------------------------------------------------------------------ */

/* area_transit_time returns the time in seconds necessary to transit the    */
/* given area based on the area type (INTERIOR or EXTERIOR)                  */

float area_transit_time(area_num)
int area_num;

{
   float area_dist, transport_speed;

   /* OFFSITE and TARGET are points without distance or transit time */
   if ( (area_num == 0) || (area_num == MAX_AREAS_PER_ASD-1) )
      return(0.0);

   area_dist = (float)get_ASD_area_dist(area_num);

   /* determine the transport speed */
```

```
   if ( (get_ASD_area_type(area_num) == EXTERIOR) &&
        (get_threat_transport_mode(get_current_threat()) == TRUCK) )
      transport_speed = get_transport_speed(TRUCK);
   else
      transport_speed = get_transport_speed(FOOT);

   return(area_dist / transport_speed);
}

/* ------------------------------------------------------------------------ */

/* element_transit_time returns the time in seconds necessary to transit the */
/* given PE based on the type of PE (INTERIOR or EXTERIOR) and the current   */
/* threat transport mode                                                     */
/* Note: if the given PE is the Helicopter Flight Path the transit time is   */
/* calculated based on Helicopter speed independent of the current threat    */
/* since only a helicopter can traverse this flight path.                    */

float element_transit_time(layer, PE_num)
int layer, PE_num;

{
   PROT_ELEM *PE_ptr;
   PE_SPECS *PE_cat_spec_ptr;
   float element_dist, transport_speed;

   PE_ptr = get_ASD_PE_ptr(layer, PE_num);
   PE_cat_spec_ptr = get_PE_cat_spec_ptr(PE_ptr->type);

   element_dist = (float)(PE_ptr->transit_dist);

   /* determine the transport speed */
   if (PE_cat_spec_ptr->helicopter)
      transport_speed = get_transport_speed(HELICOPTER);
   else
   {
      if ( (get_PE_cat_spec_ptr(PE_ptr->type)->ext_int == EXTERIOR) &&
           (get_threat_transport_mode(get_current_threat()) == TRUCK) )
         transport_speed = get_transport_speed(TRUCK);
      else
         transport_speed = get_transport_speed(FOOT);
   }

   return(element_dist / transport_speed);
}

/* ------------------------------------------------------------------------ */

savi_define_distance_quit()

{
   /* return a 1 to jump up one menu level */
   return(1);
}
```

```
/* ********************** end SAVIDIST.C source file ************************ */

/* ************************* SAVIWIND.C source file ************************* */

#include <wfd.h>
#include <savidefs.h>
#include <stdlib.h>
#include <string.h>

#define WAIT_WINDOW_TOP 10
#define WAIT_WINDOW_COL 28
#define WAIT_WINDOW_HEIGHT 4
#define WAIT_WINDOW_WIDTH 38

#define ANALYZE_ERROR_WINDOW_TOP 10
#define ANALYZE_ERROR_WINDOW_COL 26
#define ANALYZE_ERROR_WINDOW_HEIGHT 4
#define ANALYZE_ERROR_WINDOW_WIDTH 41

#define FATAL_ERROR_WINDOW_TOP 10
#define FATAL_ERROR_WINDOW_COL 19
#define FATAL_ERROR_WINDOW_HEIGHT 4
#define FATAL_ERROR_WINDOW_WIDTH 41

#define DRIVE_ERROR_WINDOW_TOP 10
#define DRIVE_ERROR_WINDOW_COL 29
#define DRIVE_ERROR_WINDOW_HEIGHT 4
#define DRIVE_ERROR_WINDOW_WIDTH 36

#define LACK_DISK_SPACE_WINDOW_TOP 10
#define LACK_DISK_SPACE_WINDOW_COL 29
#define LACK_DISK_SPACE_WINDOW_HEIGHT 4
#define LACK_DISK_SPACE_WINDOW_WIDTH 36

#define INCORRECT_VERSION_WINDOW_TOP 9
#define INCORRECT_VERSION_WINDOW_COL 33
#define INCORRECT_VERSION_WINDOW_HEIGHT 5
#define INCORRECT_VERSION_WINDOW_WIDTH 28

#define NO_FILE_WINDOW_TOP 9
#define NO_FILE_WINDOW_COL 29
#define NO_FILE_WINDOW_HEIGHT 6
#define NO_FILE_WINDOW_WIDTH 36

#define NO_DIRECTORY_WINDOW_TOP 10
#define NO_DIRECTORY_WINDOW_COL 29
#define NO_DIRECTORY_WINDOW_HEIGHT 4
#define NO_DIRECTORY_WINDOW_WIDTH 36

#define REPLACE_REQUEST_WINDOW_TOP 6
#define REPLACE_REQUEST_WINDOW_COL 29
#define REPLACE_REQUEST_WINDOW_HEIGHT 11
#define REPLACE_REQUEST_WINDOW_WIDTH 36
```

```
#define PRINTER_NOT_READY_WINDOW_TOP 10
#define PRINTER_NOT_READY_WINDOW_COL 29
#define PRINTER_NOT_READY_WINDOW_HEIGHT 4
#define PRINTER_NOT_READY_WINDOW_WIDTH 36

#define RESULTS_INDEX_WINDOW_TOP 9
#define RESULTS_INDEX_WINDOW_COL 1
#define RESULTS_INDEX_WINDOW_HEIGHT 7
#define RESULTS_INDEX_WINDOW_WIDTH 18

/* PE label window height */
#define PE_LABEL_WINDOW_HEIGHT 11

/* diagnostics window height */
#define DIAGNOSTICS_WINDOW_HEIGHT 19

/* recommendations window height */
#define RECOMMENDATIONS_WINDOW_HEIGHT 20

WINDOW popup_wn;
WINDOW popup_wn_1, popup_wn_2, popup_wn_3;

char temp_string_1[50], temp_string_2[50];

/* ------------------------------------------------------------------------ */

/* init_windows initializes the window display capabilities                 */

init_windows()

{
   /* define the popup windows */
   defs_wn(&popup_wn,
           POPUP_TOP,POPUP_RIGHT_COL,
           POPUP_HEIGHT,
           POPUP_WIDTH,
           BDR_LNP);

   defs_wn(&popup_wn_1,
           POPUP_TOP,POPUP_RIGHT_COL,
           POPUP_HEIGHT,
           POPUP_WIDTH,
           BDR_LNP);

   defs_wn(&popup_wn_2,
           POPUP_TOP,POPUP_RIGHT_COL,
           POPUP_HEIGHT,
           POPUP_WIDTH,
           BDR_LNP);

   defs_wn(&popup_wn_3,
           POPUP_TOP,POPUP_RIGHT_COL,
           POPUP_HEIGHT,
```

```
        POPUP_WIDTH,
        BDR_LNP);

    /* set each popup window to be a pop-up window */
    sw_popup(ON,&popup_wn);
    sw_popup(ON,&popup_wn_1);
    sw_popup(ON,&popup_wn_2);
    sw_popup(ON,&popup_wn_3);

    /* set the popup window color attributes */
    sw_att(LWHITE,&popup_wn);
    sw_bdratt(LBLUE,&popup_wn);

    sw_att(LWHITE,&popup_wn_1);
    sw_bdratt(LBLUE,&popup_wn_1);

    sw_att(LWHITE,&popup_wn_2);
    sw_bdratt(LBLUE,&popup_wn_2);

    sw_att(LWHITE,&popup_wn_3);
    sw_bdratt(LBLUE,&popup_wn_3);

    /* set the standard input window to be a pop-up window */
    sw_popup(ON,STDWNP);

    /* set the standard input window color attributes */
    sw_att(LWHITE,STDWNP);
    sw_bdratt(LBLUE,STDWNP);

    /* set the standard input window for no margins */
    sw_margin(0,0,STDWNP);

    /* set the pop-up windows                       */
    /* to not clear to end of line (ie: support     */
    /* character addressing)                        */
    sw_cleor(NO_CLEAR,&popup_wn);
    sw_cleor(NO_CLEAR,&popup_wn_1);
    sw_cleor(NO_CLEAR,&popup_wn_2);
    sw_cleor(NO_CLEAR,&popup_wn_3);

    /* disable exit verification on forms */
    se_verexit(OFF);

    /* disable auto move to next field when */
    /* current field is filled on forms     */
    se_automv(OFF);

    /* disable auto exit from form when last */
    /* field is filled on forms              */
    se_autoexit(OFF);

    return;
}
```

```
/* ------------------------------------------------------------------------ */

/* highlight_PE_label highlights the PE label within the PE label window    */
/* Note: the PE label window must be currently on the screen                */

highlight_PE_label()

{
   /* highlight the PE label within the label window */     -
   mv_cs(2,0,&popup_wn);
   v_natt(LHIGHLITE,ROW,&popup_wn);
   return;
}

/* ------------------------------------------------------------------------ */

/* highlight_PE_output_area highlights the PE output area within the PE     */
/* label window                                                             */
/* Note: the PE label window must be currently on the screen                */

highlight_PE_output_area()

{
   /* highlight the PE output_area within the label window */
   mv_cs(6,0,&popup_wn);
   v_natt(LHIGHLITE,ROW,&popup_wn);
   return;
}

/* ------------------------------------------------------------------------ */

/* popup_wait_window displays a popup window with the given message to      */
/* notify the user to patiently wait                                        */

popup_wait_window(wait_message)
char *wait_message;

{
   /* place wait window on screen */
   mod_wn(WAIT_WINDOW_TOP,
          WAIT_WINDOW_COL,
          WAIT_WINDOW_HEIGHT,
          WAIT_WINDOW_WIDTH,
          &popup_wn);

   sw_att(LCYAN,&popup_wn);
   set_wn(&popup_wn);

   /* display the wait window text */
   v_plst(0,CENTER_TXT,wait_message,&popup_wn);
   sw_att(LWHITE,&popup_wn);
   mv_cs(1,0,&popup_wn);
   v_st("           Please Wait            ",&popup_wn);
```

```
   return;
}

/* ------------------------------------------------------------------------ */

/* remove_wait_window removes the pop-up wait window                        */

remove_wait_window()

{
   if (window_on_screen(&popup_wn))
      unset_wn(&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* popup_analysis_error_window displays a popup window to notify the user   */
/* that an analysis error has occurred by displaying the given error        */
/* message string.  The error_message string must not exceed                */
/* MAX_ERROR_STRING_LEN including the terminating null byte.                */

popup_analysis_error_window(error_message)
char *error_message;

{
   bell();

   /* place window on screen */
   mod_wn(ANALYZE_ERROR_WINDOW_TOP,
          ANALYZE_ERROR_WINDOW_COL,
          ANALYZE_ERROR_WINDOW_HEIGHT,
          ANALYZE_ERROR_WINDOW_WIDTH,
          &popup_wn);

   sw_att(LNORMAL,&popup_wn);
   set_wn(&popup_wn);

   /* display the error window text */
   sw_att(LRED,&popup_wn);
   mv_cs(0,0,&popup_wn);
   v_st("Cannot Calculate Path Vulnerabilities",&popup_wn);
   sw_att(LWHITE,&popup_wn);
   v_plst(1,CENTER_TXT,error_message,&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* remove_analysis_error_window removes the analysis error message window   */

remove_analysis_error_window()
```

```
{
   if (window_on_screen(&popup_wn))
      unset_wn(&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* popup_fatal_error_window displays a popup window to notify the user      */
/* that a fatal error has occurred by displaying the given error            */
/* message string.  The error_message string must not exceed                */
/* MAX_ERROR_STRING_LEN including the terminating null byte.                */

popup_fatal_error_window(error_message)
char *error_message;

{
   bell();

   /* place window on screen */
   mod_wn(FATAL_ERROR_WINDOW_TOP,
          FATAL_ERROR_WINDOW_COL,
          FATAL_ERROR_WINDOW_HEIGHT,
          FATAL_ERROR_WINDOW_WIDTH,
          &popup_wn);

   sw_att(LNORMAL,&popup_wn);
   set_wn(&popup_wn);

   /* display the error window text */
   sw_att(LRED,&popup_wn);
   mv_cs(0,0,&popup_wn);
   v_st("              FATAL ERROR              ",&popup_wn);
   sw_att(LWHITE,&popup_wn);
   v_plst(1,CENTER_TXT,error_message,&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* remove_fatal_error_window removes the fatal error message window         */

remove_fatal_error_window()

{
   if (window_on_screen(&popup_wn))
      unset_wn(&popup_wn);

   return;
}
```

```
/* ------------------------------------------------------------------------ */

/* popup_drive_error_window displays a popup window to notify the user      */
/* that a disk drive error has occurred                                     */

popup_drive_error_window(error_num,drive_letter)
int error_num;
char drive_letter;

{
   bell();

   /* place window on screen */
   mod_wn(DRIVE_ERROR_WINDOW_TOP,
          DRIVE_ERROR_WINDOW_COL,
          DRIVE_ERROR_WINDOW_HEIGHT,
          DRIVE_ERROR_WINDOW_WIDTH,
          &popup_wn);

   sw_att(LNORMAL,&popup_wn);
   set_wn(&popup_wn);

   /* display the error window text */
   sw_att(LRED,&popup_wn);
   switch (error_num)
   {
      case DRIVE_NOT_READY   :
         v_plst(0,0,"      Disk Drive Not Ready      ",&popup_wn);
         break;
      case DISK_NOT_READABLE :
         v_plst(0,0,"       Disk Not Readable        ",&popup_wn);
         break;
   }

   /* create the check drive message string */
   strcpy(temp_string_1,"Check Drive ");
   temp_string_2[0] = drive_letter;
   temp_string_2[1] = '\0';
   strcat(temp_string_1,temp_string_2);

   sw_att(LWHITE,&popup_wn);
   v_plst(1,CENTER_TXT,temp_string_1,&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* remove_drive_error_window removes the drive error message window         */

remove_drive_error_window()

{
   if (window_on_screen(&popup_wn))
```

```
      unset_wn(&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* popup_lack_disk_space_window displays a popup window to notify the user   */
/* that a not enough free disk space exists on the given drive               */

popup_lack_disk_space_window(drive_letter)
char drive_letter;

{
   bell();

   /* place window on screen */
   mod_wn(LACK_DISK_SPACE_WINDOW_TOP,
          LACK_DISK_SPACE_WINDOW_COL,
          LACK_DISK_SPACE_WINDOW_HEIGHT,
          LACK_DISK_SPACE_WINDOW_WIDTH,
          &popup_wn);

   sw_att(LNORMAL,&popup_wn);
   set_wn(&popup_wn);

   sw_att(LRED,&popup_wn);
   v_plst(0,0,"Not Enough Free Space Available ",&popup_wn);

   /* create the message string */
   sprintf(temp_string_1,"On Drive %c to Save Current ASD",drive_letter);
   sw_att(LWHITE,&popup_wn);
   v_plst(1,CENTER_TXT,temp_string_1,&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* remove_lack_disk_space_window removes the lack disk space error window    */

remove_lack_disk_space_window()

{
   if (window_on_screen(&popup_wn))
      unset_wn(&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* popup_no_file_window displays a popup window to notify the user that the  */
/* given file does not exist in the current data directory                   */
```

```
popup_no_file_window(full_ASD_file_name)
char *full_ASD_file_name;

{
   bell();

   /* place window on screen */
   mod_wn(NO_FILE_WINDOW_TOP,
          NO_FILE_WINDOW_COL,
          NO_FILE_WINDOW_HEIGHT,
          NO_FILE_WINDOW_WIDTH,
          &popup_wn);

   sw_att(LNORMAL,&popup_wn);
   set_wn(&popup_wn);

   /* display the window text */
   sw_att(LRED,&popup_wn);
   v_plst(0,0,"            ASD File            ",&popup_wn);
   sw_att(LWHITE,&popup_wn);
   v_plst(1,CENTER_TXT,full_ASD_file_name,&popup_wn);
   sw_att(LRED,&popup_wn);
   v_plst(2,0,"          Not Found             ",&popup_wn);
   sw_att(LCYAN,&popup_wn);
   v_plst(3,0," in Current Data File Directory ",&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* remove_no_file_window removes the pop-up no file found window            */

remove_no_file_window()

{
   if (window_on_screen(&popup_wn))
      unset_wn(&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* popup_incorrect_version_window displays a popup window to notify the user */
/* that the given file's version is not compatible with the current software */

popup_incorrect_version_window(full_ASD_file_name)
char *full_ASD_file_name;

{
   bell();
```

```
   /* place window on screen */
   mod_wn(INCORRECT_VERSION_WINDOW_TOP,
          INCORRECT_VERSION_WINDOW_COL,
          INCORRECT_VERSION_WINDOW_HEIGHT,
          INCORRECT_VERSION_WINDOW_WIDTH,
          &popup_wn);

   sw_att(LNORMAL,&popup_wn);
   set_wn(&popup_wn);

   /* display the window text */
   sw_att(LRED,&popup_wn);
   v_plst(0,0,"        ASD File        ",&popup_wn);
   sw_att(LWHITE,&popup_wn);
   v_plst(1,CENTER_TXT,full_ASD_file_name,&popup_wn);
   sw_att(LRED,&popup_wn);
   v_plst(2,0," Version Not Supported  ",&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* remove_incorrect_version_window removes the pop-up window which indicates */
/* that the requested ASD file is not the correct version                    */

remove_incorrect_version_window()

{
   if (window_on_screen(&popup_wn))
      unset_wn(&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* popup_no_directory_window displays a popup window to notify the user that */
/* given directory does not exist                                            */

popup_no_directory_window(path)
char *path;

{
   bell();

   /* place window on screen */
   mod_wn(NO_DIRECTORY_WINDOW_TOP,
          NO_DIRECTORY_WINDOW_COL,
          NO_DIRECTORY_WINDOW_HEIGHT,
          NO_DIRECTORY_WINDOW_WIDTH,
          &popup_wn);

   sw_att(LWHITE,&popup_wn);
```

```
   set_wn(&popup_wn);

   /* display the window text */
   sw_att(LWHITE,&popup_wn);
   v_plst(0,CENTER_TXT,path,&popup_wn);
   sw_att(LRED,&popup_wn);
   v_plst(1,0,"      Directory Not Found       ",&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* remove_no_directory_window removes the pop-up no directory found window   */

remove_no_directory_window()

{
   if (window_on_screen(&popup_wn))
      unset_wn(&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* popup_replace_request_window displays a popup window to offer the user    */
/* the option to replace the given ASD file within the current data          */
/* directory                                                                 */

popup_replace_request_window(full_ASD_file_name)
char *full_ASD_file_name;

{
   bell();

   /* place window on screen */
   mod_wn(REPLACE_REQUEST_WINDOW_TOP,
          REPLACE_REQUEST_WINDOW_COL,
          REPLACE_REQUEST_WINDOW_HEIGHT,
          REPLACE_REQUEST_WINDOW_WIDTH,
          &popup_wn);

   sw_att(LNORMAL,&popup_wn);
   set_wn(&popup_wn);

   /* display the window text */
   sw_att(LRED,&popup_wn);
   v_plst(0,0,"            ASD File            ",&popup_wn);
   sw_att(LWHITE,&popup_wn);
   v_plst(1,CENTER_TXT,full_ASD_file_name,&popup_wn);
   sw_att(LRED,&popup_wn);
   v_plst(2,0,"         Already Exists         ",&popup_wn);
   sw_att(LCYAN,&popup_wn);
```

```
   v_plst(3,0," in Current Data File Directory ",&popup_wn);
   sw_att(LWHITE,&popup_wn);
   v_plst(5,0," Replace with the Current ASD ? ",&popup_wn);
   sw_att(LBLUE,&popup_wn);
   v_plst(4,0,"        DDDDDDDDDDDDDDD        ",&popup_wn);
   v_plst(6,0,"        ZDDDD?   ZDDDDD?       ",&popup_wn);
   v_plst(7,0,"        3    3   3     3       ",&popup_wn);
   v_plst(8,0,"        @DDDDY   @DDDDDY       ",&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* highlight_replace_request highlights the options YES or NO based on the  */
/* given state. If state is TRUE the YES option is highlighted within the   */
/* replace_request window, otherwise the NO option is highlighted.          */

highlight_replace_request(state)
int state;

{
   /* if state is TRUE then unhighlight NO and highlight YES */
   if (state)
   {
      sw_att(LWHITE,&popup_wn);
      v_plst(7,9," NO ",&popup_wn);
      sw_att(LHIGHLITE,&popup_wn);
      v_plst(7,18," YES ",&popup_wn);
   }
   /* otherwise unhighlight YES and highlight NO */
   else
   {
      sw_att(LWHITE,&popup_wn);
      v_plst(7,18," YES ",&popup_wn);
      sw_att(LHIGHLITE,&popup_wn);
      v_plst(7,9," NO ",&popup_wn);
   }

   /* set the popup window attribute back to normal */
   sw_att(LWHITE,&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* remove_replace_request_window removes the pop-up replace request window  */

remove_replace_request_window()

{
   if (window_on_screen(&popup_wn))
      unset_wn(&popup_wn);
```

```
   return;
}

/* ------------------------------------------------------------------------ */

/* popup_printer_not_ready_window displays a popup window to notify the user */
/* that the printer is not ready.  This may be due to a printer malfunction, */
/* the printer being off-line or no printer connected to the current port    */

popup_printer_not_ready_window()

{
   bell();

   /* place printer not ready window on screen */
   mod_wn(PRINTER_NOT_READY_WINDOW_TOP,
          PRINTER_NOT_READY_WINDOW_COL,
          PRINTER_NOT_READY_WINDOW_HEIGHT,
          PRINTER_NOT_READY_WINDOW_WIDTH,
          &popup_wn);

   sw_att(LNORMAL,&popup_wn);
   set_wn(&popup_wn);

   /* display the window text */
   sw_att(LRED,&popup_wn);
   mv_cs(0,0,&popup_wn);
   v_st("     Printing Not Possible      ",&popup_wn);
   sw_att(LWHITE,&popup_wn);
   mv_cs(1,0,&popup_wn);
   v_st("       Check the Printer        ",&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* remove_printer_not_ready_window removes the pop-up print not ready window */

remove_printer_not_ready_window()

{
   if (window_on_screen(&popup_wn))
      unset_wn(&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* popup_PE_label_window displays a pop-up window for a PE label for the PE  */
/* specified by layer and PE_num in the current ASD ensuring that the PE box */
/* is not obscured by the pop-up window                                      */
```

```
/* If the PE label window is already on the screen it is moved if necessary  */
/* or left alone if it will not obscure the given PE. If it is not on the    */
/* screen the label window is placed on the screen so as not to obscure the  */
/* given PE.                                                                 */

popup_PE_label_window(layer,PE_num)
int layer, PE_num;

{
   int popup_col;

   /* determine which side the PE label window should be on */
   if (PE_on_left(layer,PE_num) ||
       (PE_base_row(layer,PE_num)+1) >
       (POPUP_TOP+PE_LABEL_WINDOW_HEIGHT-3)
      )
      popup_col = POPUP_RIGHT_COL;
   else
      popup_col = POPUP_LEFT_COL;

   /* if the PE label window is already on the screen in the */
   /* correct place so as not to obscure the given PE then   */
   /* return, else remove it from the screen to prepare for  */
   /* moving                                                 */
   if (window_on_screen(&popup_wn))
      {
         /* set window dimensions to FULL to check left column */
         dim_wn(FULL,&popup_wn);

         /* check if window is already in correct place */
         if (popup_wn.cb == popup_col)
            {
               /* window is already in correct place   */
               /* set window dimensions back to INSIDE */
               /* before returning                     */
               dim_wn(INSIDE,&popup_wn);
               return;
            }
         else
            {
               /* window will obscure PE if not moved  */
               /* set window dimensions back to INSIDE */
               /* before removing window               */
               dim_wn(INSIDE,&popup_wn);
               remove_PE_label_window();
            }
      }

   /* place PE label window on screen in correct place */
   mod_wn(POPUP_TOP,popup_col,
          PE_LABEL_WINDOW_HEIGHT,
          POPUP_WIDTH,
          &popup_wn);
```

```
   sw_att(LCYAN,&popup_wn);
   set_wn(&popup_wn);

   /* display the label window titles */
   mv_cs(0,0,&popup_wn);
   v_st("     Protection Element Type      ",&popup_wn);

   mv_cs(3,0,&popup_wn);
   v_st("             between              ",&popup_wn);

   mv_cs(5,0,&popup_wn);
   v_st("               and                ",&popup_wn);

   sw_att(LBLUE,&popup_wn);
   mv_cs(1,0,&popup_wn);
   v_st("DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD",&popup_wn);
   mv_cs(7,0,&popup_wn);
   v_st("DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD",&popup_wn);

   /* set PE label window attribute back */
   sw_att(LWHITE,&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* display_PE_label displays the PE label information within the PE label    */
/* window for the PE specified by layer and PE_num                           */

display_PE_label(layer,PE_num)
int layer,PE_num;

{
   PROT_ELEM *PE_ptr;

   PE_ptr = get_ASD_PE_ptr(layer,PE_num);

   /* clear the PE label field */
   mv_cs(2,0,&popup_wn);
   v_qch(' ',MAX_PE_LABEL_LEN-1,&popup_wn);

   /* display the PE label */
   v_plst(2,CENTER_TXT,
          get_PE_cat_spec_ptr(PE_ptr->type)->label,
          &popup_wn);

   /* clear the input area field */
   mv_cs(4,0,&popup_wn);
   v_qch(' ',MAX_PE_LABEL_LEN-1,&popup_wn);

   /* display the input area label */
   v_plst(4,CENTER_TXT,
          get_full_ASD_area_label(PE_ptr->input_area),
```

```
            &popup_wn);

    /* clear the output area field */
    mv_cs(6,0,&popup_wn);
    v_qch(' ',MAX_PE_LABEL_LEN-1,&popup_wn);

    /* display the output area label */
    v_plst(6,CENTER_TXT,
           get_full_ASD_area_label(PE_ptr->output_area),
           &popup_wn);

    /* display the present state */
    sw_att(LCYAN,&popup_wn);
    mv_cs(8,0,&popup_wn);
    if (PE_ptr->present)
       v_qch(' ',MAX_PE_LABEL_LEN-1,&popup_wn);
    else
       v_st("Not Currently Present in ASD Model",&popup_wn);

    /* set PE label window attribute back */
    sw_att(LWHITE,&popup_wn);

    return;
}

/* ------------------------------------------------------------------------ */

/* remove_PE_label_window removes the PE label pop-up window                */

remove_PE_label_window()

{
    if (window_on_screen(&popup_wn))
       unset_wn(&popup_wn);

    return;
}

/* ------------------------------------------------------------------------ */

/* popup_results_index_window displays a pop-up window for displaying the   */
/* path number and rft for indexing into the vulnerability results          */
/* If the window is currently on the screen it is left alone.               */

popup_results_index_window()

{
    if (window_on_screen(&popup_wn_1)) return;

    /* place results index window on screen */
    mod_wn(RESULTS_INDEX_WINDOW_TOP,
           RESULTS_INDEX_WINDOW_COL,
           RESULTS_INDEX_WINDOW_HEIGHT,
           RESULTS_INDEX_WINDOW_WIDTH,
```

```
        &popup_wn_1);

    sw_att(LCYAN,&popup_wn_1);
    set_wn(&popup_wn_1);

    /* display the path number label */
    v_plst(0,0,"Path Number   ",&popup_wn_1);

    /* display the Response Force Time Label */
    v_plst(2,0,"   RFT #      ",&popup_wn_1);

    sw_att(LBLUE,&popup_wn_1);
    v_plst(1,0,"DDDDDDDDDDDDD",&popup_wn_1);

    return;
}

/* ------------------------------------------------------------------------ */

/* display_results_index displays the given path number and rft number and   */
/* time in the results index window for the analysis results functions       */
/* Note: the results index window must be on the screen before call          */

display_results_index(rft_num,path_num)
int rft_num, path_num;

{
    int rft, rft_minutes, rft_seconds;

    /* display the path number and rft number */
    sw_att(LWHITE,&popup_wn_1);
    sprintf(temp_string_1,"%-2d",path_num+1);
    v_plst(0,12,temp_string_1,&popup_wn_1);
    sprintf(temp_string_1,"%-2d",rft_num+1);
    v_plst(2,9,temp_string_1,&popup_wn_1);

    /* partition the Response Force Time */
    /* into minutes and seconds          */
    rft = get_ASD_rft(rft_num);
    partition_seconds(rft,&rft_minutes,&rft_seconds);

    /* display the response force time */
    itoa(rft_minutes,temp_string_1,10);
    right_just(temp_string_1,3);
    v_plst(3,2,temp_string_1,&popup_wn_1);

    itoa(rft_seconds,temp_string_1,10);
    right_just(temp_string_1,3);
    v_plst(4,2,temp_string_1,&popup_wn_1);

    /* display the minutes label correctly for 1 minute */
    sw_att(LCYAN,&popup_wn_1);
    if (rft_minutes == 1)
        v_plst(3,5,"minute ",&popup_wn_1);
```

```
   else
      v_plst(3,5,"minutes",&popup_wn_1);

   /* display the seconds label correctly for 1 second */
   if (rft_seconds == 1)
      v_plst(4,5,"second ",&popup_wn_1);
   else
      v_plst(4,5,"seconds",&popup_wn_1);

   return;
}

/* ------------------------------------------------------------------------ */

/* remove_results_index_window removes the results index pop-up window      */

remove_results_index_window()

{
   if (window_on_screen(&popup_wn_1))
      unset_wn(&popup_wn_1);

   return;
}

/* ------------------------------------------------------------------------ */

/* popup_diagnostics_window displays a pop-up window for displaying the     */
/* probability of interruption and remaining time values for the most       */
/* vulnerable paths within the current ASD. If the window is currently on   */
/* the screen it is left alone.                                             */

popup_diagnostics_window()

{
   float number_of_paths;

   if (window_on_screen(&popup_wn)) return;

   /* place diagnostics window on screen */
   mod_wn(POPUP_TOP,POPUP_RIGHT_COL,
          DIAGNOSTICS_WINDOW_HEIGHT,
          POPUP_WIDTH,
          &popup_wn);

   sw_att(LCYAN,&popup_wn);
   set_wn(&popup_wn);

   /* create the number of paths string */
   number_of_paths = get_valid_paths(get_current_rft_num());
   if (number_of_paths == 1)
      sprintf(temp_string_1,"of one path");
   else
      sprintf(temp_string_1,"of %.0f paths",number_of_paths);
```

```
   /* display the number of paths */
   sw_att(LCYAN,&popup_wn);
   v_plst(1,CENTER_TXT,temp_string_1,&popup_wn);

   /* display the Response Force Time Label */
   v_plst(14,0,"     Response Force Time #       ",&popup_wn);

   sw_att(LBLUE,&popup_wn);
   v_plst(2,0,"DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD",&popup_wn);
   v_plst(5,0,"          DDDDDDDDDDDDDDDD          ",&popup_wn);
   v_plst(9,0,"          DDDDDDDDDDDDDDDD          ",&popup_wn);
   v_plst(13,0,"          DDDDDDDDDDDDDDDD          ",&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* display_diagnostics_vals displays the diagnostics information within      */
/* the diagnostics window for the specified path within the set of most      */
/* vulnerable paths for the given rft number                                 */

display_diagnostics_vals(rft_num, path_num)
int rft_num, path_num;

{
   PROT_ELEM *PE_ptr;
   int layer, PE_num, segment, direction, auth_unauth, test,
       first_delay, temp_eqn;
   float det, temp_float;

   /* rft is Response Force Time */
   int rft, rft_minutes, rft_seconds;

   /* trd is time remaining after detection */
   int trd, trd_minutes, trd_seconds;

   /* tri is time remaining after interruption */
   int tri, tri_minutes, tri_seconds;

   /* create the cardinal path name string */
   if (path_num == 0)
      strcpy(temp_string_1,"");
   else
      strcpy(temp_string_1,cardinal_number(path_num+1));

   strcat(temp_string_1,"Most Vulnerable Path");

   /* display the cardinal path name string */
   sw_att(LWHITE,&popup_wn);
   v_plst(0,CENTER_TXT,temp_string_1,&popup_wn);

   /* display the probability of interruption */
```

```
det = get_prob_interruption(rft_num, path_num);
mv_cs(4,14,&popup_wn);
v_printf(&popup_wn,"%6.4f",det);

/* partition the Response Force Time */
/* into minutes and seconds          */
rft = get_ASD_rft(rft_num);
partition_seconds(rft,&rft_minutes,&rft_seconds);

/* display the Response Force # and time */
sw_att(LWHITE,&popup_wn);
sprintf(temp_string_1,"%-2d",rft_num+1);
v_plst(14,27,temp_string_1,&popup_wn);
itoa(rft_minutes,temp_string_1,10);
right_just(temp_string_1,3);
v_plst(15,13,temp_string_1,&popup_wn);

itoa(rft_seconds,temp_string_1,10);
right_just(temp_string_1,3);
v_plst(16,13,temp_string_1,&popup_wn);

/* display the minutes label correctly for 1 minute */
sw_att(LCYAN,&popup_wn);
if (rft_minutes == 1)
   v_plst(15,16,"minute ",&popup_wn);
else
   v_plst(15,16,"minutes",&popup_wn);

/* display the seconds label correctly for 1 second */
if (rft_seconds == 1)
   v_plst(16,16,"second ",&popup_wn);
else
   v_plst(16,16,"seconds",&popup_wn);

/* display the probability of interruption label */
if (det == 0)
   sw_att(LRED,&popup_wn);
else
   sw_att(LCYAN,&popup_wn);
v_plst(3,0,"   Probability of Interruption    ",&popup_wn);

/* determine the correct data labels depending on the */
/* timely_detection_point function                    */
test = timely_detection_point(rft_num, path_num,
                              &layer, &PE_num,
                              &segment,&direction);

switch (test)
{
   case TIMELY_DETECTION_OK :
      sw_att(LCYAN,&popup_wn);
      v_plst(6,0,"     Time Remaining after CDP     ",&popup_wn);
      v_plst(10,0,"Time Remaining after Interruption ",&popup_wn);
      break;
```

```
   case NO_TIMELY_DETECTION :
      sw_att(LCYAN,&popup_wn);
      v_plst(6,0,"     Time Remaining after CDP     ",&popup_wn);
      sw_att(LRED,&popup_wn);
      v_plst(10,0,"           D WARNING D            ",&popup_wn);
      v_plst(11,0,"  No Detection Exists Up To and   ",&popup_wn);
      v_plst(12,0,"Including Critical Detection Point",&popup_wn);
      break;

   case TDP_OFFSITE           : ;

   case PATH_DELAY_DEFICIENCY :
      sw_att(LCYAN,&popup_wn);
      v_plst(6,0, "      Cumulative Path Delay       ",&popup_wn);
      sw_att(LRED,&popup_wn);
      v_plst(10,0," Cumulative Path Delay Deficiency ",&popup_wn);
      break;
}

/* partition the time remaining after detection */
/* into minutes and seconds                     */
trd = get_time_remaining(rft_num, path_num);
partition_seconds(trd,&trd_minutes,&trd_seconds);

/* display the minutes label correctly for 1 minute */
sw_att(LCYAN,&popup_wn);
if (trd_minutes == 1)
   v_plst(7,16,"minute ",&popup_wn);
else
   v_plst(7,16,"minutes",&popup_wn);

/* display the seconds label correctly for 1 second */
if (trd_seconds == 1)
   v_plst(8,16,"second ",&popup_wn);
else
   v_plst(8,16,"seconds",&popup_wn);

/* display the time to goal from TDP */
sw_att(LWHITE,&popup_wn);
itoa(trd_minutes,temp_string_1,10);
right_just(temp_string_1,4);
v_plst(7,12,temp_string_1,&popup_wn);

itoa(trd_seconds,temp_string_1,10);
right_just(temp_string_1,3);
v_plst(8,13,temp_string_1,&popup_wn);

/* if no detection exists on this path do not    */
/* display the time remaining after interruption */
/* since interruption is not possible without    */
/* prior detection                               */
if (test == NO_TIMELY_DETECTION) return;
```

```
tri = abs(trd - rft);

/* if the TDP is OFFSITE or the cumulative       */
/* path delay is deficient then the              */
/* path delay deficiency is the difference       */
/* between the RFT and the cumulative path delay */
/* without the delay on the input path segment   */
/* of the first PE on the path.  Also, add in    */
/* one second since a tie goes to the intruder.  */
if ( (test == TDP_OFFSITE) || (test == PATH_DELAY_DEFICIENCY) )
{
   /* determine the first PE on the path */
   get_first_present_PE_on_path(rft_num, path_num,
                                &layer, &PE_num,
                                &direction, &auth_unauth);
   PE_ptr = get_ASD_PE_ptr(layer,PE_num);

   /* calculate the delay on the input path segment of */
   /* the first PE on the path                         */
   min_delay_traversal(layer, PE_num, auth_unauth, INPUT_SEG, direction,
                       &temp_float, &temp_eqn);
   first_delay = (int)temp_float;

   tri = abs(trd - rft - first_delay);

   /* add in one second if result will be <= 5999 (99 min, 59 sec) */
   if (tri < 5999) tri += 1;
}

/* partition the time remaining after interruption */
/* into minutes and seconds                        */
partition_seconds(tri,&tri_minutes,&tri_seconds);

/* display the minutes label correctly for 1 minute */
sw_att(LCYAN,&popup_wn);
if (tri_minutes == 1)
   v_plst(11,16,"minute ",&popup_wn);
else
   v_plst(11,16,"minutes",&popup_wn);

/* display the seconds label correctly for 1 second */
if (tri_seconds == 1)
   v_plst(12,16,"second ",&popup_wn);
else
   v_plst(12,16,"seconds",&popup_wn);

/* display the time to goal after interruption */
sw_att(LWHITE,&popup_wn);
itoa(tri_minutes,temp_string_1,10);
right_just(temp_string_1,4);
v_plst(11,12,temp_string_1,&popup_wn);

itoa(tri_seconds,temp_string_1,10);
right_just(temp_string_1,3);
```

```
   v_plst(12,13,temp_string_1,&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* remove_diagnostics_window removes the diagnostics pop-up window          */

remove_diagnostics_window()

{
   if (window_on_screen(&popup_wn))
      unset_wn(&popup_wn);

   return;
}

/* ------------------------------------------------------------------------ */

/* popup_recommendations_window displays a pop-up window for displaying the */
/* recommended improvements to the delay and detection values along any path */
/* If the window is currently on the screen it is left alone.               */

popup_recommendations_window()

{
   float number_of_paths;

   if (window_on_screen(&popup_wn)) return;

   /* place recommendations window on screen */
   mod_wn(POPUP_TOP,POPUP_RIGHT_COL,
          RECOMMENDATIONS_WINDOW_HEIGHT,
          POPUP_WIDTH,
          &popup_wn);

   sw_att(LCYAN,&popup_wn);
   set_wn(&popup_wn);

   /* display the recommendations window titles */

   /* create the number of paths string */
   number_of_paths = get_valid_paths(get_current_rft_num());
   if (number_of_paths == 1)
      sprintf(temp_string_1,"of one path");
   else
      sprintf(temp_string_1,"of %.0f paths",number_of_paths);

   /* display the number of paths */
   sw_att(LCYAN,&popup_wn);
   v_plst(1,CENTER_TXT,temp_string_1,&popup_wn);

   return;
```

```
}

/* ------------------------------------------------------------------------ */

/* display_recommendations displays the improvement information within the   */
/* recommendations window for the given path for the given rft               */

display_recommendations(rft_num, path_num)
int rft_num, path_num;

{
   PROT_ELEM *PE_ptr;
   int layer, PE_num, test, segment, direction, auth_unauth, tr, rft,
       del_def, del_def_minutes, del_def_seconds,
       first_delay, temp_eqn;
   float temp_float;

   test = timely_detection_point(rft_num, path_num,
                                 &layer, &PE_num,
                                 &segment, &direction);

   /* create the cardinal path name string */
   if (path_num == 0)
      strcpy(temp_string_1,"");
   else
      strcpy(temp_string_1,cardinal_number(path_num+1));

   strcat(temp_string_1,"Most Vulnerable Path");

   /* display the cardinal path name string */
   sw_att(LWHITE,&popup_wn);
   v_plst(0,CENTER_TXT,temp_string_1,&popup_wn);

   sw_att(LBLUE,&popup_wn);
   v_plst(2,0,
          "DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD",
          &popup_wn);

   /* display the improvement recommendations */
   switch (test)
   {
      /* if the timely_detection point exists, recommend improving */
      /* detection up to and including the TDP and delay after     */
      case NO_TIMELY_DETECTION : ;
      case TIMELY_DETECTION_OK : PE_ptr = get_ASD_PE_ptr(layer,PE_num);

                                 sw_att(LWHITE,&popup_wn);

                                 if (segment == PATH_OUT)
                                    v_plst(7,0,
                                       "       Output Path Segment        ",
                                       &popup_wn);
                                 else
                                    v_plst(7,0,
```

```
      "       Input Path Segment        ",
      &popup_wn);

/* clear the PE label field */
mv_cs(9,0,&popup_wn);
v_qch(' ',MAX_PE_LABEL_LEN-1,&popup_wn);

/* display the PE label */
v_plst(9,CENTER_TXT,
   get_PE_cat_spec_ptr(PE_ptr->type)->label,
   &popup_wn);

/* clear the input area field */
mv_cs(11,0,&popup_wn);
v_qch(' ',MAX_PE_LABEL_LEN-1,&popup_wn);

/* clear the output area field */
mv_cs(13,0,&popup_wn);
v_qch(' ',MAX_PE_LABEL_LEN-1,&popup_wn);

if (direction == ENTRY)
{
   /* display the input area label */
   v_plst(11,CENTER_TXT,
      get_full_ASD_area_label(
         PE_ptr->input_area),
      &popup_wn);

   /* display the output area label */
   v_plst(13,CENTER_TXT,
      get_full_ASD_area_label(
         PE_ptr->output_area),
      &popup_wn);
}
else
{
   /* display the output area label */
   v_plst(11,CENTER_TXT,
      get_full_ASD_area_label(
         PE_ptr->output_area),
      &popup_wn);

   /* display the input area label */
   v_plst(13,CENTER_TXT,
      get_full_ASD_area_label(
         PE_ptr->input_area),
      &popup_wn);
}

/* display the recommended improvements */
v_plst(3,0,
   "   Consider Improving Detection   ",
   &popup_wn);
```

```
                        sw_att(LCYAN,&popup_wn);
                        v_plst(4,0,
                           "       on any Path Segment        ",
                           &popup_wn);
                        v_plst(5,0,
                           "     up to and including the      ",
                           &popup_wn);
                        v_plst(6,0,
                           " Critical Detection Point on the  ",
                           &popup_wn);
                        v_plst(8,0,
                           "              of the              ",
                           &popup_wn);
                        v_plst(10,0,
                           "          leading from            ",
                           &popup_wn);
                        v_plst(12,0,
                           "                to                ",
                           &popup_wn);

                        sw_att(LWHITE,&popup_wn);
                        v_plst(15,0,
                           "    Consider Increasing Delay     ",
                           &popup_wn);

                        sw_att(LCYAN,&popup_wn);
                        v_plst(16,0,
                           "       on any Path Segment        ",
                           &popup_wn);
                        v_plst(17,0,
                           "   after this point to the Goal   ",
                           &popup_wn);

                        sw_att(LBLUE,&popup_wn);
                        v_plst(14,0,
                           "         DDDDDDDDDDDDDDDD         ",
                           &popup_wn);

                        break;

/* if the TDP is offsite, adding delay on the input */
/* segment of the first PE will not improve the     */
/* path, since no detection prior to delay exists.  */
/* recommend adding delay to all path segments      */
/* except the input path segment of the first PE on */
/* the path                                         */
case TDP_OFFSITE        : sw_att(LRED,&popup_wn);
                          v_plst(3,0,
                             "   No Critical Detection Point    ",
                            &popup_wn);

                          break;

/* Also, if a cumulative path delay deficiency   */
```

```
   /* exists, recommend adding delay to all path    */
   /* segments except the input path segment of the */
   /* first PE on the path                          */
   case PATH_DELAY_DEFICIENCY : sw_att(LRED,&popup_wn);
                                v_plst(3,0,
                                   " Cumulative Path Delay Deficiency ",
                                   &popup_wn);

                                break;
}

if ( (test == TDP_OFFSITE) || (test == PATH_DELAY_DEFICIENCY) )
{
   sw_att(LWHITE,&popup_wn);
   v_plst(5,0,
      "     Increase the Cumulative      ",
     &popup_wn);
   v_plst(6,0,
      "      Path Delay by at least      ",
     &popup_wn);

   /* determine the first PE on the path */
   get_first_present_PE_on_path(rft_num, path_num,
                                &layer, &PE_num,
                                &direction, &auth_unauth);
   PE_ptr = get_ASD_PE_ptr(layer,PE_num);

   /* the path delay deficiency is the difference   */
   /* between the RFT and the cumulative path delay */
   /* without the delay on the input path segment   */
   /* of the first PE on the path.  Also, add in    */
   /* one second since a tie goes to the intruder.  */
   tr = get_time_remaining(rft_num, path_num);
   rft = get_ASD_rft(rft_num);

   /* calculate the delay on the input path segment of */
   /* the first PE on the path                         */
   min_delay_traversal(layer, PE_num, auth_unauth, INPUT_SEG, direction,
                       &temp_float, &temp_eqn);
   first_delay = (int)temp_float;

   del_def = abs(tr - rft - first_delay);

   /* add in one second if result will be <= 5999 (99 min, 59 sec) */
   if (del_def < 5999) del_def += 1;

   /* partition the delay deficiency */
   /* into minutes and seconds       */
   partition_seconds(del_def,
                     &del_def_minutes,
                     &del_def_seconds);

   /* display the minutes label */
   /* correctly for 1 minute    */
```

```
sw_att(LCYAN,&popup_wn);
if (del_def_minutes == 1)
   v_plst(7,15,"minute ",&popup_wn);
else
   v_plst(7,15,"minutes",&popup_wn);

/* display the seconds label */
/* correctly for 1 second    */
if (del_def_seconds == 1)
   v_plst(8,15,"second ",&popup_wn);
else
   v_plst(8,15,"seconds",&popup_wn);

/* display the path delay deficiency */
sw_att(LWHITE,&popup_wn);
itoa(del_def_minutes,temp_string_1,10);
right_just(temp_string_1,4);
v_plst(7,11,temp_string_1,&popup_wn);

itoa(del_def_seconds,temp_string_1,10);
right_just(temp_string_1,3);
v_plst(8,12,temp_string_1,&popup_wn);

sw_att(LWHITE,&popup_wn);

/* clear the PE label field */
mv_cs(13,0,&popup_wn);
v_qch(' ',MAX_PE_LABEL_LEN-1,&popup_wn);

/* display the PE label */
v_plst(13,CENTER_TXT,
   get_PE_cat_spec_ptr(PE_ptr->type)->label,
   &popup_wn);

/* clear the input area field */
mv_cs(15,0,&popup_wn);
v_qch(' ',MAX_PE_LABEL_LEN-1,&popup_wn);

/* display the input area label */
v_plst(15,CENTER_TXT,
   get_full_ASD_area_label(PE_ptr->input_area),
   &popup_wn);

/* clear the output area field */
mv_cs(17,0,&popup_wn);
v_qch(' ',MAX_PE_LABEL_LEN-1,&popup_wn);

/* display the output area label */
v_plst(17,CENTER_TXT,
   get_full_ASD_area_label(PE_ptr->output_area),
   &popup_wn);

sw_att(LCYAN,&popup_wn);
v_plst(9,0,
```

```
            "       on the Path Segments       ",
            &popup_wn);
         v_plst(10,0,
            "     between the Goal and the     ",
            &popup_wn);
         sw_att(LWHITE,&popup_wn);
         v_plst(11,0,
            "       Output Path Segment        ",
            &popup_wn);
         sw_att(LCYAN,&popup_wn);
         v_plst(12,0,
            "               of the             ",
            &popup_wn);
         v_plst(14,0,
            "            leading from          ",
            &popup_wn);
         v_plst(16,0,
            "                 to               ",
            &popup_wn);

         sw_att(LBLUE,&popup_wn);
         v_plst(4,0,
            "DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD",
            &popup_wn);
      }

      return;
}

/* ------------------------------------------------------------------------ */

/* remove_recommendations_window removes the recommendations pop-up window  */

remove_recommendations_window()

{
      if (window_on_screen(&popup_wn))
         unset_wn(&popup_wn);

      return;
}

/* ------------------------------------------------------------------------ */

/* get_popup_col returns the popup window's current left edge column number */

get_popup_col()

{
      return(popup_wn.cb);
}

/* ------------------------------------------------------------------------ */
```

```
/* popup_window_on_screen returns TRUE if the standard popup window is on    */
/* the screen and returns FALSE otherwise                                     */

popup_window_on_screen()

{
   return(window_on_screen(&popup_wn));
}

/* ------------------------------------------------------------------------- */

/* window_on_screen returns TRUE if the window which wn_ptr points to is on  */
/* the screen and returns FALSE otherwise                                     */

window_on_screen(wn_ptr)
WINDOW *wn_ptr;

{
   return((wn_ptr->setsw) != 0);
}

/* ********************** end SAVIWIND.C source file ********************** */
   #include <dos.h>
   #include <stdio.h>
   #include <string.h>
   #include "pathdef.h"
   #include "frntdecs.h"

   #define LINT_ARGS       1
   #define LOWERLIMIT      1
   #define UPPERLIMIT      800
   #define ULIMIT          15
   #define MAX_PATHS       10
   #define MAX_ELEMENTS    100
   #define MAX_THRESHOLDS  10
   #define TRUE            1
   #define FALSE           0
   #define NIL             0

   int nlinks,f_userlimit,f_verbose;
   int parent[UPPERLIMIT],child[UPPERLIMIT];
   int sum1[UPPERLIMIT],sum2[UPPERLIMIT],sum3[UPPERLIMIT];
   float prod1[UPPERLIMIT],prod2[UPPERLIMIT],prod3[UPPERLIMIT];
   int clone[UPPERLIMIT];
   FILE *results;
   FILE *asdptr;
   char asdfile[15];

   int f_target,f_origin,f_thmin,f_thmax,f_thnum,f_thres[MAX_THRESHOLDS];

   int f_dependency,f_diag,f_done;
   float f_count;        /*number of paths*/

   char request;
```

```
   char threat_label[45],ASD_label[70];

/*************************************************/
/*          inititialization and input           */
/*************************************************/

inputs()
      /*read in the matrix which corresponds to */
      /*connections in the graph*/


{
   int i,err,areadone,layerdone,parnt;
   char ch;

   /* allow the user to declare how many elements are in this*/
   /* temporary matrix which describes connections in the graph*/

   f_userlimit = 9999;
   while ((f_userlimit < LOWERLIMIT) || (f_userlimit > UPPERLIMIT))
      {
      printf("Enter target area number : ");
      scanf("%d",&f_userlimit);
      f_target = f_userlimit;
      }

   printf("Do you want to see paths both in and out ? ");
   ch = getch();
   if ((ch == 'Y') || (ch == 'y'))
      f_dependency = TRUE;
   else
      f_dependency = FALSE;
   f_origin = 0;
   free_tree();
   err = turbo_init(MAX_THRESHOLDS,MAX_PATHS,
              f_userlimit,f_origin,f_target,f_dependency);
   if (err != 0)
     printf("turbo_boffed, err=%d\n",err);

   i = 0;
   nlinks = 0;
   parnt=0;
   while (parnt >=0 )
   {
      printf("=================================================================\n");

            printf("    Enter Parent >");
            scanf("%d",&parnt);
            printf("%d\n",parnt);
           if (parnt >= 0)
            {
            parent[nlinks]=parnt;
            printf("    Enter Child >");
            scanf("%d",&child[nlinks]);
```

```
            printf("%d\n",child[nlinks]);
            printf("    Enter Time Delay for input  segment >");
            scanf("%d",&sum1[nlinks]);
            printf("%d\n",sum1[nlinks]);
               printf("     Enter Detection  for input segment >");
               scanf("%f",&prod1[nlinks]);
               printf("%f\n",prod1[nlinks]);
            printf("     Enter Time Delay for output segment >");
            scanf("%d",&sum2[nlinks]);
            printf("%d\n",sum2[nlinks]);
               printf("     Enter Detection  for output segment >");
               scanf("%f",&prod2[nlinks]);
               printf("%f\n",prod2[nlinks]);
            printf("     Enter Time Delay for dependence     >");
            scanf("%d",&sum3[nlinks]);
            printf("%d\n",sum3[nlinks]);
               printf("     Enter Detection  for dependence     >");
               scanf("%f",&prod3[nlinks]);
               printf("%f\n",prod3[nlinks]);
            printf("     Enter Clone Brother (-1 if none)    >");
            scanf("%d",&clone[nlinks]);
            printf("%d\n",clone[nlinks]);
            i = nlinks;
         insert_link(parent[i],child[i],sum1[i],prod1[i],sum2[i],prod2[i],
                     sum3[i],prod3[i],clone[i]);
          nlinks++;
         }
        printf("\n");
       }

    } /*inputs*/



    findpath()
    {
    int i,j,lim;
    float num;
    if (f_diag)
       printf("enter findpath\n");
    f_count = 0.0; /*note that we may use this later as max paths to return*/
    f_count = turbo_SEApath(f_thres,f_thnum);

    printf("done total number of paths =%.0f (%d RFTs)\n",f_count,f_thnum);
    }

menu (ch)
    char ch;

    {
    int i,j,lim,num,th,pn;
    float half;
    PATH_INFO *pptr;
    int err,rft;
```

```
long start,end;
char ch2;

while (((ch < '1') || (ch > '9'))&&
        (ch != 's')&&(ch != 'l')&&(ch != 'q')&&(ch != 'd'))
   {
   printf("seapath menu:%d\n",f_userlimit);
   printf("1.   input connection matrix\n");
   printf("2.   set response force time\n");
   printf("3.   view the tree\n");
   printf("4.   preview number of paths\n");
   printf("5.   Analyze the ASD\n");
   printf("6.   view saved paths\n");
   printf("7.   Sensitivity Analysis\n");
   printf("8.   view a single path\n");
   printf("9.   toggle in/out   on/off \n");
   printf("d    directory of ASD files\n");
   printf("s    save the current ASD\n");
   printf("l    load an ASD\n");
   printf("q.   quit\n");
   ch = getch();
   }
switch(ch)
   {
   case '1' :
      printf("initializing...\n");
      inputs();
      break;
   case '2' :
      printf("Enter minimum response force time [%d]:",f_thmin);
      scanf("%d",&f_thmin);
      printf("  .. %d\n",f_thmin);
      printf("Enter maximum response force time [%d]:",f_thmax);
      scanf("%d",&f_thmax);
      printf("  .. %d\n",f_thmax);
      printf("Enter number of response force times [%d]:",f_thnum);
      scanf("%d",&f_thnum);
      printf("  .. %d\n",f_thnum);
      if (f_thnum == 1)
         f_thres[0] = f_thmin;
      else
         for (i=0;i<f_thnum;i++)
            f_thres[i] = f_thmin + (i*((f_thmax - f_thmin)/(f_thnum - 1)));
      break;
   case '3' :
      traversetree(stdout);
      break;
   case '4' :
      num = get_num_paths();
      printf("This ASD has %d legal paths...\n",num);
      break;
   case '5' :
      printf("Analyzing...\n");
      start = time();
```

```
            findpath();
            end = time();
            printf("Elapsed time = %ld\n",end-start);
            break;
         case '6' :
            for (j=0; j<f_thnum; j++)
            {
               lim = get_displayable_paths(j);
               printf("displayable paths for thresh %d = %d (of %.0f total)\n",
                       j,lim,get_valid_paths(j));
               if (lim>MAX_PATHS)
                  lim = MAX_PATHS;
               for (i= 0; i <lim; i++)
                  printout(stdout,j,i);
            }
            break;
         case '7' :
            rft = 0;
            graph_service(f_thres,f_thnum,&rft);
            break;
         case '8' :
            for (i=0; i<f_thnum; i++)
            {
               for (j=0; j<10; j++)
                  printf("%.0f ",get_frequency_ptr(i)[j]);
               printf("\n");
            }
            printf("Enter threshold set:[0-%d]",f_thnum-1);
            scanf("%d",&th);
            printf ("Enter path number:");
            scanf("%d",&pn);
            pptr = get_path_ptr(th,pn);
            printf ("pptr = %lx\n",pptr);
            if (pptr != NIL)
            {
               for (i=0; i<pptr->number_of_nodes; i++)
                  printf (" %d",pptr->nodes[i]);
               printf ("\n");
            }
            else
               printf ("No such path!\n");
            break;
         case '9' :
            printf("toddling tree..");
            free_tree();
                    f_dependency = !f_dependency;
                    printf("dependency is %d\n",f_dependency);
            f_origin = 0;
            err=turbo_init(MAX_THRESHOLDS,MAX_PATHS,
                f_target,f_origin,f_target,f_dependency);
            if (err != 0)
              printf("turbo_boffed, err=%d\n",err);
            for (i=0; i<nlinks; i++)
rod3[i]);      insert_link(parent[i],child[i],sum1[i],prod1[i],sum2[i],prod2[i],sum3[
```

```
            break;
         case 's' :
            save_asd();
            break;
         case 'l' :
            load_asd();
            break;
         case 'q' :
            f_done = TRUE;
            printf("goodbye\n");
            break;
         case 'd' :
            system("dir *.asd");
            break;

         }
         if (ch != 'q')
            {
            printf("enter request, 0 to see menu....\n");
            ch = getch();
            }

      return(ch);
      } /*menu*/

int save_asd()
{
   int i;

         printf("Enter ASD filename to save: ");
         scanf("%s",asdfile);
         strcat(asdfile,".asd");
         asdptr = fopen(asdfile,"w");
         if (asdptr == NULL)
         {
            printf("\x7 Error opening %s\n",asdfile);
            return;
         }
         else
         {
            fprintf(asdptr,"%d\n",f_dependency);
            fprintf(asdptr,"%d %d %d\n",f_thmin,f_thmax,f_thnum);
            fprintf(asdptr,"%d\n",f_target);
            fprintf(asdptr,"%d\n",nlinks);
            for (i=0; i<nlinks; i++)
               {
               fprintf(asdptr,"%d %d\n",parent[i],child[i]);
               fprintf(asdptr,"%d %f\n",sum1[i],prod1[i]);
               fprintf(asdptr,"%d %f\n",sum2[i],prod2[i]);
               fprintf(asdptr,"%d %f\n",sum3[i],prod3[i]);
               fprintf(asdptr,"%d\n",clone[i]);
               }
         }
         fclose(asdptr);
```

```
}

int load_asd()
{
   int i,err;

         printf("Enter ASD filename: ");
         scanf("%s",asdfile);
         strcat(asdfile,".asd");
         asdptr = fopen(asdfile,"r");
         if (asdptr == NULL)
         {
            printf("\x7 Error opening %s\n",asdfile);
            return;
         }
         else
         {
            fscanf(asdptr,"%d",&f_dependency);
            fscanf(asdptr,"%d %d %d",&f_thmin,&f_thmax,&f_thnum);
            if (f_thnum == 1)
               f_thres[0] = f_thmin;
            else
               for (i=0;i<f_thnum;i++)
                  f_thres[i] = f_thmin + (i*((f_thmax - f_thmin)/(f_thnum - 1)));
            fscanf(asdptr,"%d",&f_target);
            free_tree();
            f_origin = 0;
            err=turbo_init(MAX_THRESHOLDS,MAX_PATHS,
                f_target,f_origin,f_target,f_dependency);
            if (err != 0)
              printf("turbo_boffed, err=%d\n",err);
            fscanf(asdptr,"%d",&nlinks);
            for (i=0; i<nlinks; i++)
               {
               fscanf(asdptr,"%d %d",&parent[i],&child[i]);
               fscanf(asdptr,"%d %f",&sum1[i],&prod1[i]);
               fscanf(asdptr,"%d %f",&sum2[i],&prod2[i]);
               fscanf(asdptr,"%d %f",&sum3[i],&prod3[i]);
               fscanf(asdptr,"%d",&clone[i]);
               insert_link(parent[i],child[i],sum1[i],prod1[i],
                           sum2[i],prod2[i],sum3[i],prod3[i],clone[i]);
               }
         }
         fclose(asdptr);
}

char *get_threat_label(dummy)
int dummy;
{
   return(threat_label);
}

char *get_ASD_label()
{
```

```
   return(ASD_label);
}

get_current_threat()
{
   return(0);
}

get_current_direction()
{
   return(1);
}

get_current_strategy()
{
   return(1);
}

/*******************************************************/
/*          MAIN PROGRAM                               */
/*******************************************************/

main()
{
   int err;

   printf("SEAPATH\n");
   results = fopen("results","r+b");
   if (results == NULL)
      printf("Results file not found\n");
   else
   {
      fread((char *)&f_thnum,sizeof(int),1,results);
      fread((char *)f_thres,(MAX_THRESHOLDS*sizeof(int)),1,results);
      err = load_results(results);
      printf("load_result = %d\n",err);
   }
   init_meta();
   sprintf(threat_label,"Outsiders with Metal/Explosives & Helicopter");
   sprintf(ASD_label,"Sample Facility for Front (Simple Front End)");
   f_done = FALSE;
   request = ' ';
   while (!(f_done))
   request = menu(request);
   printf("Good bye from Turbo!!\n");
   if (results != NULL)
      fclose(results);
   results = fopen("results","w+b");
   if (results == NULL)
      printf("error opening results file...\n");
   else
   {
      fwrite((char *)&f_thnum,sizeof(int),1,results);
      fwrite((char *)f_thres,(MAX_THRESHOLDS*sizeof(int)),1,results);
```

```
      save_results(results);
   }
   if (results != NULL)
      fclose(results);
}

int display_turbo_progress(percent,timeRemaining)
int percent;
float timeRemaining;
{
   printf("Percentage done=%d, Time remaining=%.0f minutes\n",
          percent,timeRemaining);
}

#define PRINTER_INTERRUPT       23
#define PRINTER_STATUS_SERVICE  2
#define ON_LINE_AND_NOT_BUSY    144
/* ------------------------------------------------------------------------ */

/* printer_ready returns TRUE only if the given printer (0-1) is on-line and */
/* not busy. Otherwise FALSE is returned.                                     */

printer_ready(printer_number)
int printer_number;

{
   union REGS regs;

   regs.h.ah = PRINTER_STATUS_SERVICE;
   regs.x.dx = printer_number;

   int86(PRINTER_INTERRUPT,&regs,&regs);

   return(regs.h.ah == ON_LINE_AND_NOT_BUSY);
}

#define LINT_ARGS 1
#define DEBUG 0

#include <stdio.h>
#include <time.h>
#include <conio.h>
#include <malloc.h>
#include "pathdef.h"
#include "pathglobals.h"
#include "trbodecs.h"

/*************************************************************************/
/*                          path package                                 */
/*************************************************************************/
/* Included here are all functions to create and traverse the tree. */
/* reset    - marks all nodes from a given point in the tree to     */
/*            unvisited so other paths below this node can be found */
/* getpaths - a series of functions allowing traversal of the tree  */
```

```
/*            from a given point to the end of the tree (until a    */
/*            nil pointer is found).  If the element at the end of  */
/*            the path is a target then a valid path has been found */
/*     loadstack - put a node onto the stack and continue pushing   */
/*                 what the stacktop node is connected to until no  */
/*                 further connections are found                    */
/*     readstack - print out all element names from bottom to       */
/*                 stacktop.  The count of total paths may be       */
/*                 incremented here since a valid path has been     */
/*                 found before calling this function               */
/*     backtrack - look for another path from the node on the top   */
/*                 of the stack. If this node has other connections */
/*                 mark it visited, put its connection on top of    */
/*                 the stack, and attempt to follow this path.  If  */
/*                 there are no other connections, mark it visited, */
/*                 and pop it off the stack until a valid           */
/*                 connection is found, or the stack is empty       */
/*******************************************************************/


/*******************************************************************/
/*******************************************************************/
int true_name(alias)
int alias;
{
   if (alias > target)
      return(alias-target-1);
   else
      return(alias);
}


/*******************************************************************/
/*                              getpaths                           */
/*******************************************************************/
SEGMENT_INFO *backtrack()
{

   SEGMENT_INFO *ptr;

   ptr = STACK.index[STACK.stack_top];
   STACK.stack_top = STACK.stack_top - 1;
   ptr->visited = TRUE;

n))        while ((ptr->next_segment == NIL)&&(STACK.index[STACK.stack_top]->node_num != or
   {
      ptr = STACK.index[STACK.stack_top];
      STACK.stack_top = STACK.stack_top - 1;
      ptr->visited = TRUE;
   }

   if (tree[ptr->node_num] != NIL)
      reset(tree[ptr->node_num]);
```

```
   if (ptr->next_segment != NIL)
   {
      while (ptr->visited)
         ptr = ptr->next_segment;
   }
   if ((STACK.index[STACK.stack_top]->node_num == origin)&&!(ptr->visited))
      reset(ptr);
   STACK.stack_top = STACK.stack_top + 1;
   STACK.index[STACK.stack_top] = ptr;
   if (ptr->visited)
   {
      ptr = STACK.index[STACK.stack_top];
      STACK.stack_top = STACK.stack_top - 1;
   }
   return(ptr);
}

int load_stack(ptr)

SEGMENT_INFO *ptr;
{
   while (ptr != NIL)
   {
      if ((STACK.index[STACK.stack_top]->node_num != destination) &&
          (ptr->node_num == destination))
      {
         STACK.stack_top = STACK.stack_top + 1;
         STACK.index[STACK.stack_top] = ptr;
         ptr = NIL;
      }
      else
      {
         if (tree[ptr->node_num] != NIL)
         {
            if (!(tree[ptr->node_num]->visited))
            {
               STACK.stack_top = STACK.stack_top + 1;
               STACK.index[STACK.stack_top] = tree[ptr->node_num];
               ptr = tree[ptr->node_num];
               if (ptr->node_num == destination)
                  ptr = NIL;
            }
            else
               ptr = ptr->next_segment;
         }
         else
            ptr = NIL;
      }
   }
   return;
}

#if DEBUG > 0
int print_stack(out)
```

```
FILE *out;
{
   int i;

   for (i=STACK.stack_top; i>=0; i--)
      fprintf(out,"(%d) ",true_name(STACK.index[i]->node_num));
   fprintf(out,"\n");
   for (i=STACK.stack_top; i>=0; i--)
      fprintf(out,"[%d] ",STACK.index[i]->link_num);
   fprintf(out,"\n");
   return;
}
#endif

int reset(ptr)
SEGMENT_INFO *ptr;
{
   if (ptr != NIL)
   {
      if (ptr->visited)
      {
         ptr->visited = FALSE;
         reset(ptr->next_segment);
         reset(tree[ptr->node_num]);
      }
   }
   return;
}

float get_num_paths()
{
   int save_dep,save_dest;
   float count;
   SEGMENT_INFO *path_ptr,*save_tree;

   save_dep = dependency;
   save_dest = destination;
   destination = target;
   dependency = FALSE;
   count = 0;
   save_tree = tree[target];
   tree[target] = NIL;
   path_ptr = tree[0];
   reset(path_ptr);
/*   STACK.stack_top = STACK.stack_top + 1;*/
STACK.stack_top = 1;
   STACK.index[STACK.stack_top] = path_ptr;

   while (STACK.index[STACK.stack_top]->node_num != origin)
   {
      load_stack(path_ptr); /*get next unvisited path*/
      if (STACK.index[STACK.stack_top]->node_num == destination)
         count = count + 1;
      path_ptr = backtrack(); /*step back to get other connections*/
```

```
   }

   if (save_dep)
      count = count * count;
   dependency = save_dep;
   destination = save_dest;
   tree[target] = save_tree;

   return(count);
}/* get_num_paths */

float turbo_SEApath(thresh,nthresh)
int *thresh,nthresh;
{
   int i,j,ch,checkcount,CheckLoops,ratio;
   long start,tm;
   float count,numpaths,timeLeft;
   SEGMENT_INFO *path_ptr;

   threshold = thresh;
   n_thresholds = nthresh;

   numpaths = get_num_paths();
   time(&tm);
   start = tm;
   CheckLoops = CHECKLOOPS/nthresh;
   checkcount = 0;
   count = 0;
   init_valid_paths_array();
   init_frequency_array();
   for (j=0;j<n_thresholds;j++)
   {
      paths = path_array_ptr[j];
      for (i=0;i<maxp;i++)
      {
         paths[i].cum_prod = 0;
         paths[i].cum_sum=INFINITY;
         paths[i].number_of_nodes=0;
         paths[i].break_point=0;
      }
   }
   for (j=0; j<n_thresholds; j++)
   {
      for (i =0;i<maxp;i++)
      {
         index[j*maxp+i] = i;
      }
   }
   path_ptr = tree[0];
   reset(path_ptr);
/*   STACK.stack_top = STACK.stack_top + 1;*/
STACK.stack_top = 1;
   STACK.index[STACK.stack_top] = path_ptr;
   while (STACK.index[STACK.stack_top]->node_num != origin)
```

```
   {
      load_stack(path_ptr); /*get next unvisited path*/
      if (STACK.index[STACK.stack_top]->node_num == destination)
      {
         count = count+1;
/*temp    print_stack(stdout);*/
         quantify();
         checkcount++;
         if (checkcount > CheckLoops)
         {
            checkcount = 0;
            while (kbhit())
            {
               ch=getch();
               if (ch==ESC)
               {
                  while (kbhit())getch();
                  return(-count);
               }
            }
            time(&tm);
            if (count==numpaths)
               timeLeft = 0.01;
            else
               timeLeft = (tm-start)/((numpaths/(numpaths-count)-1)*60.0);
            display_turbo_progress((int)(100*count/numpaths),timeLeft);
         }
      }
      path_ptr = backtrack(); /*step back to get other connections*/
   }
   return(count);
}/* turbo_SEApath */

/*=========================================================================*/
/*                          seasupp package                                */
/*=========================================================================*/
/*************************************************************************/
/*                           insert package                              */
/*************************************************************************/
int insert_link(parent,child,sum1,prod1,sum2,prod2,sum3,prod3,userfield)
int parent,child;
float prod1,prod2,prod3;
int sum1,sum2,sum3;
int userfield;

{
   int err;

   if (parent >= child)
      return(BAD_NODE_NUM);
   if ((parent < 0)||(parent >= headnodes))
      return(BAD_PARENT);
   numlinks++;
   err = DoInsert(numlinks,parent,child,sum1,prod1,sum2,prod2,sum3,prod3,
```

```
                    userfield);
    if (err != OK)
       return(OUT_OF_MEMORY);
    if (dependency)
    {
       err = DoInsert(numlinks,child+target+1,parent+target+1,sum1,prod1,
                      sum2,prod2,sum3,prod3,userfield);
       tree[target] = tree[2*target+1];
       if (err != OK)
          return(OUT_OF_MEMORY);
    }
    return(numlinks);
}

int DoInsert(linknum,parent,child,sum1,prod1,sum2,prod2,sum3,prod3,userfield)
int linknum,parent,child;
float prod1,prod2,prod3;
int sum1,sum2,sum3;
int userfield;
{
    SEGMENT_INFO *ptr;

    ptr = (SEGMENT_INFO*)malloc(sizeof(SEGMENT_INFO));
    if (ptr == NIL)
       return(OUT_OF_MEMORY);
    /* assign the proper values to the new node */
    ptr->link_num = linknum;
    ptr->node_num = child;
    ptr->visited = FALSE;
    ptr->sum_entry = sum1;
    ptr->sum_exit  = sum2;
    ptr->sum_ex_dep = sum3;
    ptr->prod_entry = prod1;
    ptr->prod_exit  = prod2;
    ptr->prod_ex_dep = prod3;
    ptr->user_field = userfield;
    ptr->next_segment = NIL;

    /* put this new node at the beginning of the list */
    ptr->next_segment = tree[parent];

    tree[parent] = ptr;

    return(OK);
}



/*****************************************************************/
/*                        deallocate package                     */
/*****************************************************************/

int free_tree()
{
```

```
    int          i,j;
    SEGMENT_INFO *ptr;

    if (!allocated)
       return;

    /* Retain the pointer to the dummy node at stack position 0  then */
    /* load all the children of tree[i] for deletion                  */
    /* NOTE: this limits the number of children any node can have to  */
    /* STACK_LIMIT or less !                                          */
    STACK.stack_top = EMPTY+1;

    for (i=0; i<headnodes; i++)
    {
       ptr = tree[i];
       while (ptr != NIL)
       {
          STACK.stack_top++;
          STACK.index[STACK.stack_top] = ptr;
          ptr = ptr->next_segment;
       }
       tree[i] = NIL;
       while (STACK.stack_top != EMPTY)
       {
          free(STACK.index[STACK.stack_top]);
          STACK.stack_top--;
       }
    }
    free(path_location);
    free(path_array_ptr);
    free(tree);
    allocated = FALSE;
    return;
}

/*************************************************************************/
/*                          initialize package                           */
/*************************************************************************/

int turbo_init(maxthresh,maxpaths,num_headnodes,org,targ,dep)
int maxthresh,maxpaths,num_headnodes,org,targ,dep;
{
    int i;
    SEGMENT_INFO *path_ptr;

    if (allocated)
       return(SEAPATH_INITED);

    origin = org;
    target = targ;
    dependency = dep;
    in_headnodes = num_headnodes;
    if (dependency)
    {
```

```
      destination = target + 1;
      headnodes = 2*num_headnodes + 2;
   }
   else
   {
      destination = target;
      headnodes = num_headnodes + 1;
   }
   maxp = maxpaths;
   maxt = maxthresh;

   /* allocate space for all path information */
   path_array_ptr = (PATH_INFO**)malloc(maxthresh * sizeof(PATH_INFO*));
   if (path_array_ptr == NIL)
         return(OUT_OF_MEMORY);
   path_location = (PATH_INFO *)malloc((unsigned)(maxthresh * sizeof(PATH_INFO) *
               maxpaths));
   if (path_location == NIL)
         return(OUT_OF_MEMORY);

   for (i = 0;i < maxthresh; i++)
      path_array_ptr[i] =(PATH_INFO *)((long) path_location +
            (long)(i * maxpaths * sizeof(PATH_INFO)));
   /* allocate space for the tree */
   tree = (SEGMENT_INFO**)malloc(headnodes * sizeof(SEGMENT_INFO *));
   if (tree == NIL)
         return(OUT_OF_MEMORY);

   for (i = 0;i<headnodes;i++)
      tree[i] = NIL;

   allocated = TRUE;

   STACK.stack_top = EMPTY;
   /* push a dummy node onto the stack */
   numlinks = 0;
   path_ptr = (SEGMENT_INFO*)malloc(sizeof(SEGMENT_INFO));
   /* assign the proper values to the new node */
   path_ptr->link_num = 0;
   path_ptr->node_num = origin;
   path_ptr->visited = FALSE;
   path_ptr->user_field = EMPTY;
   path_ptr->next_segment = NIL;
   STACK.stack_top = STACK.stack_top + 1;
   STACK.index[STACK.stack_top] = path_ptr;

   return(OK);
}

int save_results(stream)
FILE *stream;
{
   int int_size,float_size,err;
```

```
   int_size = sizeof(int);
   float_size = sizeof(float);

   /* save variables required by turbo_init... */
   fwrite((char *)&maxt,int_size,1,stream);
   fwrite((char *)&maxp,int_size,1,stream);
   fwrite((char *)&in_headnodes,int_size,1,stream);
   fwrite((char *)&origin,int_size,1,stream);
   fwrite((char *)&target,int_size,1,stream);
   fwrite((char *)&dependency,int_size,1,stream);

   /* save variables required to retrieve path information... */
   fwrite((char *)path_location,sizeof(PATH_INFO),maxt*maxp,stream);
   fwrite((char *)index,int_size,MAX_THRESH*MAX_RETURNED,stream);
   fwrite((char *)valid_paths,float_size,MAX_THRESH,stream);
   fwrite((char *)get_frequency_ptr(0),float_size,MAX_THRESH*FREQ_BINS,
            stream);
   fwrite((char *)&n_thresholds,int_size,1,stream);
   return;
}

int get_results_size()
{
   int results_size;

   results_size = (7+MAX_THRESH*MAX_RETURNED)*sizeof(int)+
                  (1+MAX_THRESH)*sizeof(float)+
                  maxt*maxp*sizeof(PATH_INFO);
   return(results_size);
}

int load_results(stream)
FILE *stream;
{
   int int_size,float_size,err,count;
   int maxthresh,maxpaths,headnodes,org,targ,dep;

   int_size = sizeof(int);
   float_size = sizeof(float);

   count = 0;

   /* load variables required by turbo_init... */
   count += fread((char *)&maxthresh,int_size,1,stream);
   count += fread((char *)&maxpaths,int_size,1,stream);
   count += fread((char *)&headnodes,int_size,1,stream);
   count += fread((char *)&org,int_size,1,stream);
   count += fread((char *)&targ,int_size,1,stream);
   count += fread((char *)&dep,int_size,1,stream);

   free_tree();
   err = turbo_init(maxthresh,maxpaths,headnodes,org,targ,dep);
   if (err != OK)
      return(err);
```

```
   /* load variables required to retrieve path information... */
   count += fread((char *)path_location,sizeof(PATH_INFO),maxthresh*maxpaths,
            stream);
   count += fread((char *)index,int_size,MAX_THRESH*MAX_RETURNED,stream);
   count += fread((char *)valid_paths,float_size,MAX_THRESH,stream);
   count += fread((char *)get_frequency_ptr(0),float_size,MAX_THRESH*FREQ_BINS,
            stream);
   count += fread((char *)&n_thresholds,int_size,1,stream);
   if (count != 7+maxthresh*maxpaths+MAX_THRESH*MAX_RETURNED+MAX_THRESH+
                MAX_THRESH*FREQ_BINS)
      return(READ_FAILURE);
   else
      return(OK);
}

PATH_INFO *get_path_ptr(thresh,path_number)
int thresh,path_number;
{
   paths = path_array_ptr[thresh];
   if ((path_number < 0)||(path_number >= maxp)||
       (thresh < 0)||(thresh >= n_thresholds))
      return(NIL);
   else
      return(&paths[index[thresh*maxp+path_number]]);
}

#if DEBUG > 0
int printout(out,thresh,path_number)
FILE *out;
int thresh,path_number;
{
   int j,posn;

   if ((path_number < 0)||(path_number >= maxp)||
       (thresh < 0)||(thresh >= n_thresholds))
   {
      fprintf(out,"No such path: %d, %d\n",thresh,path_number);
      return;
   }

   paths = path_array_ptr[thresh];
   posn = thresh*maxp+path_number;
   fprintf(out," Path %d (Thresh = %d)\n",path_number+1,threshold[thresh]);
   fprintf(out,"    Product Value      = %f\n",paths[index[posn]].cum_prod);
   fprintf(out,"    Sum Value          = %d\n",paths[index[posn]].cum_sum);
   if (paths[index[posn]].break_point < 0)
      fprintf(out,"    Break point        = (Threshold never passed)\n");
   else
      fprintf(out,"    Break point        = %d\n",
                  paths[index[posn]].break_point);
   fprintf(out,"    Path is >>");
   for (j=0;j<paths[index[posn]].number_of_nodes;j++)
      fprintf(out," %d",paths[index[posn]].nodes[j]);
```

```
   fprintf(out," <<\n");
   fprintf(out,"    User is >>");
   for (j=0;j<paths[index[posn]].number_of_nodes;j++)
      fprintf(out," %d",paths[index[posn]].user[j]);
   fprintf(out," <<\n\n");
   return;
}
#endif

#if DEBUG > 0
traversetree(out) /*view the "tree", which is a linked list of connections*/
FILE *out;
{
   SEGMENT_INFO *ptr;
   int i, j, lmt;

   lmt = headnodes;

   fprintf(out,"traverse: limit=%d dependency=%d target=%d destination=%d\n",
                     lmt,dependency,target,destination);
   for (i = 0;i< headnodes;i++)
   {
      if ((tree[i] != NIL)&&(i != 2*target+1)) /*there's a connection*/
      {
         fprintf(out,"%3d: >>",i);
         ptr = tree[i];
         while (ptr != NIL)  /*printf out all connetions on this level*/
         {
            fprintf(out,"(%3d ",ptr->node_num);
            if (ptr->visited)
              fprintf(out,"v) ");
            else
              fprintf(out,"n) ");
            ptr = ptr->next_segment;
         }
         fprintf (out,"<<\n");
      }
      else
      {
         if (i == destination)
            fprintf(out,"%3d: >>  NIL  <<\n",i);
      }
   }
} /*traverse tree*/
#endif

#if DEBUG > 0
saviwalk(thresh,nthresh) /* write the tree in 'walker asd' format to file 'saviwalk
d' */
int *thresh,nthresh;
{
   SEGMENT_INFO *ptr;
   int i,links;
   FILE *asdptr;
```

```
   asdptr = fopen("saviwalk.asd","w");

   links = 0;
   for (i = 0;i< target;i++) /* count the connections */
   {
      if (tree[i] != NIL) /*there's a connection*/
      {
         ptr = tree[i];
         while (ptr != NIL)  /*count all connetions on this level*/
         {
            links++;
            ptr = ptr->next_segment;
         }
      }
   }
   fprintf(asdptr,"%d\n",dependency);
   fprintf(asdptr,"%d %d %d\n",thresh[0],thresh[nthresh-1],nthresh);
   fprintf(asdptr,"%d\n",target);
   fprintf(asdptr,"%d\n",links);

   for (i = 0;i< target;i++) /* write the connections */
   {
      if (tree[i] != NIL) /*there's a connection*/
      {
         ptr = tree[i];
         while (ptr != NIL)  /*write out all connetions on this level*/
         {
            links++;
            fprintf(asdptr,"%d %d\n",i,ptr->node_num);
            fprintf(asdptr,"%d %f\n",ptr->sum_entry,ptr->prod_entry);
            fprintf(asdptr,"%d %f\n",ptr->sum_exit,ptr->prod_exit);
            fprintf(asdptr,"%d %f\n",ptr->sum_ex_dep,ptr->prod_ex_dep);
            fprintf(asdptr,"%d\n",ptr->user_field);
            ptr = ptr->next_segment;
         }
      }
   }
   fclose(asdptr);

} /*saviwalk*/
#endif

#if DEBUG > 0
int print_segment_info(seg_ptr,out)
SEGMENT_INFO *seg_ptr;
FILE *out;
{
   fprintf(out,"Segment_Info at %x:\n",seg_ptr);
   fprintf(out," Link Num: %d\n",seg_ptr->link_num);
   fprintf(out,"  node_num: %d\n",seg_ptr->node_num);
   fprintf(out,"   prod_entry: %f   sum_entry: %d\n",
           seg_ptr->prod_entry,seg_ptr->sum_entry);
   fprintf(out,"    prod_exit: %f    sum_exit: %d\n",
           seg_ptr->prod_exit,seg_ptr->sum_exit);
```

```
   fprintf(out,"  prod_ex_dep: %f  sum_ex_dep: %d\n",
           seg_ptr->prod_ex_dep,seg_ptr->sum_ex_dep);

   if (seg_ptr->next_segment == NIL)
      fprintf(out,"  next_seg = <NIL>\n");
   else
      fprintf(out,"  next_seg = %x\n",seg_ptr->next_segment);

   fprintf(out,"  visisted = %d\n\n\n",seg_ptr->visited);
}
#endif

/*************************************************************************/
/*                        get number of valid paths                      */
/*************************************************************************/
float get_valid_paths(thresh)
int thresh;
{
   return(valid_paths[thresh]);
}

int get_displayable_paths(thresh)
int thresh;
{
   int i;
   i = 0;
   paths = path_array_ptr[thresh];
   while ((paths[index[thresh*maxp+i]].number_of_nodes > 0)&&(i<maxp))
      i++;
   return(i);
}

/* ********************************************************************** */
/*                        Quantify  Package                               */
/* ********************************************************************** */

/* ====================================================================== */
/*                        Quantify                                        */
/* ====================================================================== */
/* Accumulate the  weights on the given path.  There are two weights here*/
/* to be saved and are the basis for ranking the MAX_RETURNED number of  */
/* paths.  The primary weight is path_prod, a product of probabilities of*/
/* detection for each segment. The other is the path_sum, the sum of all */
/* time associated with each segment.                                    */
int quantify()
{
   int ndx,newpos,i,j,thresh,base,good_path,next_thresh,position,parasite;
   int node,OldPE,pathnode,stacknode;
   int dependent[NODE_LIMIT];


   /* locate the target on the stack */
   midpoint = STACK.stack_top + 1;
   for (i=0;i<=STACK.stack_top;i++)
```

```
   {
      if(STACK.index[i]->node_num == target)
      {
         midpoint = i;
         break;
      }
   }

   /* load the array of dependence flags for all nodes from midpoint  */
   /* to stacktop. check for invalid combinations                     */

   if (dependency)
   {
      for (i=STACK.stack_top;i > midpoint;i--)
      {
         dependent[i] = FALSE;
         for (j=0;j<=midpoint;j++)
         {
            /* check for invalid paths (deceit after force) */
            if (STACK.index[i]->user_field == STACK.index[j]->link_num)
/*temp  {printf("invalid path\n");*/
               return;
/*temp  }*/
            /* check for repeated segments */
            if (STACK.index[j]->link_num == STACK.index[i]->link_num)
               dependent[i] = TRUE;
         }
      }
   }

   for (thresh=0; thresh<n_thresholds; thresh++)
   {
      good_path = TRUE;
      OldPE = FALSE; /* TRUE if leaving a PE */
      next_thresh = FALSE;
      paths = path_array_ptr[thresh];
      path_prod = 1;
      path_sum  = 0;
      breakpoint = THRESH_NEVER_PASSED;

      node = STACK.stack_top;
      while ((path_sum <= threshold[thresh]) && (node > 0))
      {
         if ((STACK.index[node]->user_field > 0)&&(OldPE))
         {
/*temp  printf("redundant path\n");*/
            good_path = FALSE;
            break;
         }
         else
         {
            if (node <= midpoint) /* TRUE if target passed */
               path_sum += STACK.index[node]->sum_entry;
            else
```

```
            {
               if (dependent[node])
                  path_sum += STACK.index[node]->sum_ex_dep;
               else
                  path_sum += STACK.index[node]->sum_exit;
            }
         }
         OldPE = !OldPE;
         node--;
      }

      if (good_path)
      {
         if (path_sum > threshold[thresh])
            breakpoint = node + 1;
         while (node > 0)
         {
            if (node <= midpoint) /* TRUE if target passed */
               path_prod *= STACK.index[node]->prod_entry;
            else
            {
               if (dependent[node])
                  path_prod *= STACK.index[node]->prod_ex_dep;
               else
                  path_prod *= STACK.index[node]->prod_exit;
            }
            node--;
         }

         valid_paths[thresh]++;

         /* rank this path's path_prod in the frequency array */
         ndx = (int)(path_prod*FREQ_BINS);
         if (ndx >= FREQ_BINS)
            ndx = FREQ_BINS-1;
         frequency[thresh][ndx] = frequency[thresh][ndx] + 1.0;

         /* Check for the presence of 'parasite' paths, i.e. paths whose    */
         /* elements are the same (user_field is used for this              */
         /* determination if it is non-negative)                            */
         base = thresh*maxp;
         for (parasite=0; parasite<maxp; parasite++)
         {
            if (STACK.stack_top+1 ==
                paths[index[base+parasite]].number_of_nodes)
            {
               for (j=0; j<=STACK.stack_top; j++)
               {
                  /* user[] contains decloned link_num of saved path.  */
                  /* if the link_num of the stack path is the same, we */
                  /* have a possible parasite... otherwise, break      */
                  if (STACK.index[j]->user_field > 0)
                  {
                     if (STACK.index[j]->user_field !=
```

```
                        paths[index[base+parasite]].user[j])
                        break;
                    }
                    else
                    {
                        if (STACK.index[j]->link_num !=
                            paths[index[base+parasite]].user[j])
                        break;
                    }
                }
                if (j == STACK.stack_top+1)
/*temp  {printf("parasite path\n");*/
                    break;
/*temp  }*/
            }
        }

        if (parasite < maxp)
        {
            good_path = FALSE;
            if ((paths[index[base+parasite]].cum_prod < path_prod) ||
                ((paths[index[base+parasite]].cum_prod == path_prod) &&
                 (paths[index[base+parasite]].cum_sum > path_sum)))
                {
                    newpos = index[base+parasite];
                    for (i=parasite; i<maxp-1; i++)
                        index[base+i] = index[base+i+1];
                    index[base+maxp-1] = newpos;
                    good_path = TRUE;
                }
        }
    }

    if (good_path)
    {
        /* Save only the MAX_RETURNED most paths, in the array called      */
        /* 'paths'.  This array is indexed through the array called 'index */
        /* so that only pointers are shuffled.                             */

        ndx = 0;

        /* Locate the spot in the index array for this path by skipping    */
        /* all probability values less than that for this path             */

        while ((ndx < maxp) && (paths[index[base+ndx]].cum_prod > path_prod))
            ndx++;
        /*skip all time values  greater  than  that for this path */
        if (paths[index[base+ndx]].cum_prod == path_prod)
        {
            while (((ndx< maxp) && (paths[index[base+ndx]].cum_sum < path_sum))
                  && (paths[index[base+ndx]].cum_prod == path_prod))
                  ndx++;
        }
```

```
         /* found the spot to insert it into */
         if(ndx < maxp)
         /*delete the least critical item and assign new values*/
         {
            position = base + ndx;
            newpos = index[base+maxp-1];
            for (i=maxp-1;i>ndx;i--)
               index[base+i] = index[base+i-1];
            index[position] = newpos;
            paths[newpos].cum_prod = path_prod;
            paths[newpos].cum_sum  = path_sum;
            if (breakpoint<0)
               paths[newpos].break_point = THRESH_NEVER_PASSED;
            else
               paths[newpos].break_point = breakpoint-1;
            paths[newpos].number_of_nodes = STACK.stack_top + 1;

            for (i=0;i<=STACK.stack_top;i++)
            {
               paths[newpos].nodes[i] =
                    true_name(STACK.index[i]->node_num);
               if (STACK.index[i]->user_field > 0)
                  paths[newpos].user[i] = STACK.index[i]->user_field;
               else
                  paths[newpos].user[i] = STACK.index[i]->link_num;
            }
         }
      }
   }

   return;
}

float *get_frequency_ptr(which_set)
int which_set;
{
      return(&frequency[which_set][0]);
}

int init_frequency_array()
{
   int i,j;
   for (i=0; i<MAX_THRESH; i++)
   {
      for (j=0; j<FREQ_BINS; j++)
         frequency[i][j] = 0;
   }
}

int init_valid_paths_array()
{
   int i;
   for (i=0; i<MAX_THRESH; i++)
      valid_paths[i] = 0.0;
```

```
}
#include <dos.h>
#include <stdio.h>
#include <math.h>
#include <malloc.h>
#include <string.h>
#include <GRconst.h>
#include <GRports.h>          /* MetaWINDOW header files  */
#include <GRextrn.h>
#include "plot.h"

GR_ENVIRONMENT graph;

struct
{
   int stacktop;
   GR_ENVIRONMENT environs[ENV_STACK_LIMIT];
} EnvStack;
rect ScreenR,PlotR;

int InitPlot()
{
   EnvStack.stacktop = EMPTY;
   graph.ResX = GetBMapField(c_pixWidth);
   graph.ResY = GetBMapField(c_pixHeight);
   SetBorderArea(20,20,20,20);
   SetMaxCount(15);
   SetMinor(XAXIS,0);
   SetMinor(YAXIS,0);
   SetAxis(XAXIS,LINTYPE);
   SetAxis(YAXIS,LINTYPE);
   ScalePlot(0.0,100.0,0.0,100.0);
   return;
}

int SetBorderArea(left,right,bottom,top)
int left,right,bottom,top;
{
   int PortOrgX,PortOrgY;

   graph.BorderLeft = left;
   graph.BorderRight = right;
   graph.BorderBottom = bottom;
   graph.BorderTop = top;
   PortOrgX = -(graph.ResX*graph.BorderLeft/100);
   PortOrgY = -(graph.ResY*graph.BorderBottom/100);
   graph.PlotXMax = graph.ResX-(graph.ResX*graph.BorderRight/100)+PortOrgX;
   graph.PlotYMax = graph.ResY-(graph.ResY*graph.BorderTop/100)+PortOrgY;

   SetOrigin(PortOrgX,PortOrgY);

   SetRect(&ScreenR,PortOrgX,PortOrgY,PortOrgX+graph.ResX,PortOrgY+graph.ResY);
   SetRect(&PlotR,0,0,graph.PlotXMax,graph.PlotYMax);
   return;
```

```
}

int ScalePlot(UsrXMin,UsrXMax,UsrYMin,UsrYMax)
float UsrXMin,UsrXMax,UsrYMin,UsrYMax;

{
   float min,max,inc;

   graph.XCount = SetScale(UsrXMin,UsrXMax,graph.XType,&min,&max,&inc);
   graph.XMin = min;
   graph.XMax = max;
   graph.XInc = inc;

   graph.YCount = SetScale(UsrYMin,UsrYMax,graph.YType,&min,&max,&inc);
   graph.YMin = min;
   graph.YMax = max;
   graph.YInc = inc;

   return;
}

int SetAxis(theAxis,axisType)
int theAxis,axisType;
{
   if (theAxis == XAXIS)
      graph.XType = axisType;
   else
      graph.YType = axisType;
   return;
}

int ScaledX(x)
float x;
{
   int i;
   if (graph.XType == LOGTYPE)
      x = log10(x);
   i = (int)((x-graph.XMin)/(graph.XMax-graph.XMin)*graph.PlotXMax);
   return(i);
}

int ScaledY(y)
float y;
{
   int i;
   if (graph.YType == LOGTYPE)
      y = log10(y);
   i = (int)((y-graph.YMin)/(graph.YMax-graph.YMin)*graph.PlotYMax);
   return(i);
}

int SetScale(UserMin,UserMax,axisType,min,max,inc)
float UserMin,UserMax,*min,*max,*inc;
int axisType;
```

```
{
   static int choices[3] =
   {
      1,2,5
   };
   int power,i,count;
   float difference,domain,transform,tinc;

   if (UserMax <= UserMin)
      return(0);
   if (axisType == LOGTYPE)
   {
      UserMin = log10(UserMin);
      UserMax = log10(UserMax);
   }
   difference = UserMax-UserMin;
   domain = ((int)log10(difference) - 2);
   transform = pow(10.0,domain);
   power = (int)log10(difference) - 2;
   i = 0;
   count = graph.MaxCount + 1;
   while (count > graph.MaxCount)
   {
      (*inc) = choices[i] * pow(10.0,(double)power);
      tinc = (*inc)/transform;
      (*min) = floor((UserMin/transform)/tinc);
      (*min) = transform * (*min) * tinc;
      count = (UserMax - *min)/(*inc) - 1;
      while (UserMax/(*min + count*(*inc)) > 1.0001) count++;
      (*max) = *min + count*(*inc);
      i++;
      if (i==3)
      {
         power++;
         i = 0;
      }
   }
   return(count);
}

int SetCount(UserMin,UserMax,axisType,count,min,max,inc)
float UserMin,UserMax,*min,*max,*inc;
int axisType,count;
{
   static int options[3] =
   {
      1,2,5
   };
   int power,i;
   float difference,domain,transform,tinc;

   if (UserMax <= UserMin)
      return(NOT_OK);
   if (axisType == LOGTYPE)
```

```
    {
       UserMin = log10(UserMin);
       UserMax = log10(UserMax);
    }
    difference = UserMax-UserMin;
    domain = ((int)log10(difference) - 2);
    transform = pow(10.0,domain);
    power = (int)log10(difference) - 2;
    i = 0;
    (*max) = UserMin;
    while ((*max) < UserMax)
    {
       (*inc) = options[i] * pow(10.0,(double)power);
       tinc = (*inc)/transform;
       (*min) = floor((UserMin/transform)/tinc);
       (*min) = transform * (*min) * tinc;
       (*max) = *min + count*(*inc);
       i++;
       if (i==3)
       {
          power++;
          i = 0;
       }
    }
    return(OK);
}

int SetMinor(theAxis,minorTics)
int theAxis,minorTics;
{
    if (minorTics < 0)
       minorTics = 0;
    if (theAxis == XAXIS)
       graph.XMinor = minorTics;
    else
       graph.YMinor = minorTics;
    return;
}

int XTic(XPosition,YPosition,TicUp,TicDown)
float XPosition,YPosition;
int TicUp,TicDown;
{
    int x,y;
    float range;
    range = graph.YMax - graph.YMin;
    x = ScaledX(XPosition);
    y = ScaledY(YPosition + TicUp * range/100.0);
    MoveTo(x,y);
    y = ScaledY(YPosition - TicDown * range/100.0);
    LineTo(x,y);
    return;
}
```

```
int YTic(XPosition,YPosition,TicLeft,TicRight)
float XPosition,YPosition;
int TicLeft,TicRight;
{
   int x,y;
   float range;
   range = graph.XMax - graph.XMin;
   y = ScaledY(YPosition);
   x = ScaledX(XPosition + TicRight * range/100.0);
   MoveTo(x,y);
   x = ScaledX(XPosition - TicLeft * range/100.0);
   LineTo(x,y);
   return;
}

int XTicMinor(TicType,XPosition,YPosition,TicUp,TicDown)
float XPosition,YPosition;
int TicType,TicUp,TicDown;
{
   int j;
   if (TicType == LOGTYPE)
      for (j=2; j<10; j++)
         XTic(XPosition+graph.XInc*log10((double)j),YPosition,
              TicUp,TicDown);
   else
      if (graph.XMinor > 0)
         for (j=1; j<graph.XMinor; j++)
            XTic(XPosition+j*graph.XInc/graph.XMinor,YPosition,TicUp,TicDown);
   return;
}

int YTicMinor(TicType,XPosition,YPosition,TicLeft,TicRight)
float XPosition,YPosition;
int TicType,TicLeft,TicRight;
{
   int j;
   if (TicType == LOGTYPE)
      for (j=2; j<10; j++)
         YTic(XPosition,YPosition+graph.YInc*log10((double)j),
              TicLeft,TicRight);
   else
      if (graph.XMinor > 0)
         for (j=1; j<graph.YMinor; j++)
            YTic(XPosition,YPosition+j*graph.YInc/graph.YMinor,
                 TicLeft,TicRight);
   return;
}

int DrawAxis(theAxis,DrawFlag,TicIn,TicOut,MinorIn,MinorOut)
int theAxis,DrawFlag,TicIn,TicOut,MinorIn,MinorOut;
{
   int i,Decimals,Sign,Count,saveXAxis,saveYAxis;
   float dx,dy,x,y,yStart,xStart;
```

```
saveXAxis = graph.XType;
saveYAxis = graph.YType;
SetAxis(XAXIS,LINTYPE);
SetAxis(YAXIS,LINTYPE);

if ((theAxis == TOPAXIS) || (theAxis == BOTTOMAXIS))
{
   dx = graph.XInc;
   dy = 0;
   Count = graph.XCount;
   xStart = graph.XMin;
   if (theAxis == TOPAXIS)
   {
      Swap(&TicIn,&TicOut);
      Swap(&MinorIn,&MinorOut);
      yStart = graph.YMax;
   }
   else
      yStart = graph.YMin;
   if (DrawFlag)
   {
      MoveTo(ScaledX(graph.XMin),ScaledY(yStart));
      LineTo(ScaledX(graph.XMax),ScaledY(yStart));
   }
}
else
{
   dx = 0;
   dy = graph.YInc;
   Count = graph.YCount;
   yStart = graph.YMin;
   if (theAxis == LEFTAXIS)
   {
      Swap(&TicIn,&TicOut);
      Swap(&MinorIn,&MinorOut);
      xStart = graph.XMin;
   }
   else
      xStart = graph.XMax;
   if (DrawFlag)
   {
      MoveTo(ScaledX(xStart),ScaledY(graph.YMin));
      LineTo(ScaledX(xStart),ScaledY(graph.YMax));
   }
}
for (i = 0;i <= Count; i++)
{
   x = xStart + (i*dx);
   y = yStart + (i*dy);
   if ((theAxis == TOPAXIS) || (theAxis == BOTTOMAXIS))
   {
      XTic(x,y,TicIn,TicOut);
      if (i != Count)
         XTicMinor(saveXAxis,x,y,MinorIn,MinorOut);
```

```
      }
      else
      {
         YTic(x,y,TicIn,TicOut);
         if (i != Count)
            YTicMinor(saveYAxis,x,y,MinorIn,MinorOut);
      }
   }
   SetAxis(XAXIS,saveXAxis);
   SetAxis(YAXIS,saveYAxis);
   return;
}

int LabelAxis(theAxis,labels,LabelOut)
int theAxis,LabelOut;
LABEL *labels;
{
   int i,Count,AlignHorz,AlignVert,saveXAxis,saveYAxis;
   float dx,dy,x,y,range;

   saveXAxis = graph.XType;
   saveYAxis = graph.YType;
   SetAxis(XAXIS,LINTYPE);
   SetAxis(YAXIS,LINTYPE);

   if ((theAxis == TOPAXIS) || (theAxis == BOTTOMAXIS))
   {
      dx = graph.XInc;
      dy = 0;
      Count = graph.XCount;
      range = graph.YMax-graph.YMin;
      x = graph.XMin;
      if (theAxis == TOPAXIS)
         y = graph.YMax + range * LabelOut/100.0;
      else
         y = graph.YMin - range * LabelOut/100.0;
   }
   else
   {
      dx = 0;
      dy = graph.YInc;
      Count = graph.YCount;
      range = graph.XMax - graph.XMin;
      y = graph.YMin;
      if (theAxis == LEFTAXIS)
         x = graph.XMin - range * LabelOut/100.0;
      else
         x = graph.XMax + range * LabelOut/100.0;
   }
   AlignVert = alignMiddle;
   if ((theAxis == TOPAXIS) || (theAxis == BOTTOMAXIS))
      AlignHorz = alignCenter;
   else
      {
```

```
      if (theAxis == LEFTAXIS)
         AlignHorz = alignRight;
      else
         AlignHorz = alignLeft;
      )
   TextAlign(AlignHorz,AlignVert);
   for (i = 0;i <= Count; i++)
   {
      MoveTo(ScaledX(x),ScaledY(y));
      TextPath(0);
      SpecialText(labels[i]);
      x = x + dx;
      y = y + dy;
   }
   SetAxis(XAXIS,saveXAxis);
   SetAxis(YAXIS,saveYAxis);
   return;
}

int SetLabels(LabelType,labels,LabelFirst,LabelLast,Start,Count,Increment)
int LabelType,LabelFirst,LabelLast,Count;
float Start,Increment;
LABEL *labels;
{
   int i;

   for (i=0;i(=Count;i++)
      switch(LabelType)
         (
         case INT_LABELS:
            sprintf(labels[i],"%d",(int)(Start + i * Increment));
            break;
         case FL1_LABELS:
            sprintf(labels[i],"%.1f",Start + i * Increment);
            break;
         case FL2_LABELS:
            sprintf(labels[i],"%.2f",Start + i * Increment);
            break;
         case FL3_LABELS:
            sprintf(labels[i],"%.3f",Start + i * Increment);
            break;
         case FL4_LABELS:
            sprintf(labels[i],"%.4f",Start + i * Increment);
            break;
         case LOG_LABELS:
            sprintf(labels[i],"10^%d",(int)(Start + i * Increment));
            break;
         case SCI_LABELS:
            break;
         )
      if (!LabelFirst)
         sprintf(labels[0],"\0");
      if (!LabelLast)
         sprintf(labels[Count],"\0");
```

```
    return;
}

int TitleAxis(theAxis,title,TitleOut)
int theAxis,TitleOut;
char *title;
{
   float x,y;
   int path,saveXAxis,saveYAxis;

   saveXAxis = graph.XType;
   saveYAxis = graph.YType;
   SetAxis(XAXIS,LINTYPE);
   SetAxis(YAXIS,LINTYPE);
   switch(theAxis)
   {
      case LEFTAXIS:
         x = graph.XMin - TitleOut * (graph.XMax - graph.XMin) / 100.0;
         y = graph.YMin + (graph.YMax - graph.YMin)/2.0;
/*temp   path = 2700;*/
         path = 0;
         break;
      case RIGHTAXIS:
         x = graph.XMax + TitleOut * (graph.XMax - graph.XMin) / 100.0;
         y = graph.YMin + (graph.YMax - graph.YMin)/2.0;
/*temp   path = 2700;*/
         path = 0;
         break;
      case TOPAXIS:
         x = graph.XMin + (graph.XMax - graph.XMin)/2.0;
         y = graph.YMax + TitleOut * (graph.YMax - graph.YMin) / 100.0;
         path = 0;
         break;
      case BOTTOMAXIS:
         x = graph.XMin + (graph.XMax - graph.XMin)/2.0;
         y = graph.YMin - TitleOut * (graph.YMax - graph.YMin) / 100.0;
         path = 0;
         break;
   }

   MoveTo(ScaledX(x),ScaledY(y));
   TextPath(path);
   TextAlign(alignCenter,alignMiddle);
   SpecialText(title);
   TextPath(0);
   SetAxis(XAXIS,saveXAxis);
   SetAxis(YAXIS,saveYAxis);

   return;
}

int Bar(position,width,height,alignment,pattern)
float position,width,height;
int alignment,pattern;
```

```
{
   rect ClipR,bar;
   float min;

   DupRect(&PlotR,&ClipR);
   InsetRect(&ClipR,1,1);
   ClipRect(&ClipR);

   if (graph.YType == LOGTYPE)
   {
      if (height < pow(10.0,graph.YMin))
      {
         ClipRect(&ScreenR);
         return;
      }
      if (height > pow(10.0,graph.YMax))
         height = pow(10.0,graph.YMax);
      min = pow(10.0,graph.YMin);
   }
   else
   {
      if (height <= graph.YMin)
      {
         ClipRect(&ScreenR);
         return;
      }
      if (height > graph.YMax)
         height = graph.YMax;
      min = graph.YMin;
   }
   switch (alignment)
   {
      case LEFT:
         SetRect(&bar,
                 ScaledX(position),ScaledY(min),
                 ScaledX(position+width),ScaledY(height));
         break;
      case RIGHT:
         SetRect(&bar,
                 ScaledX(position-width),ScaledY(min),
                 ScaledX(position),ScaledY(height));
         break;
      default:
         SetRect(&bar,
                 ScaledX(position-width/2.),ScaledY(min),
                 ScaledX(position+width/2.),ScaledY(height));
   }
   FillRect(&bar,pattern);
   FrameRect(&bar);
   ClipRect(&ScreenR);
   return;
}

int ClipScreen()
```

```
{
   ClipRect(&ScreenR);
   return;
}

int ClipPlot()
{
   rect ClipR;

   DupRect(&PlotR,&ClipR);
   InsetRect(&ClipR,1,1);
   ClipRect(&ClipR);
   return;
}

int EraseScreen()
{
   EraseRect(&ScreenR);
   return;
}

int ErasePlot()
{
   rect PlotAreaR;

   DupRect(&PlotR,&PlotAreaR);
   InsetRect(&PlotAreaR,1,1);
   EraseRect(&PlotAreaR);
   return;
}

int PushEnviron()
{
   if (EnvStack.stacktop < ENV_STACK_LIMIT)
   {
      EnvStack.stacktop++;
      CopyEnviron(&EnvStack.environs[EnvStack.stacktop]);
      return(OK);
   }
   else
      return(NOT_OK);
}

int PopEnviron()
{
   if (EnvStack.stacktop != EMPTY)
   {
      SetEnviron(&EnvStack.environs[EnvStack.stacktop]);
      EnvStack.stacktop--;
      return(OK);
   }
   else
      return(NOT_OK);
}
```

```
int CopyEnviron(TargetEnviron)
GR_ENVIRONMENT *TargetEnviron;
/* copies current graph environment into 'TargetEnviron' */
{
   TargetEnviron->ResX = graph.ResX;
   TargetEnviron->ResY = graph.ResY;
   TargetEnviron->PlotXMax = graph.PlotXMax;
   TargetEnviron->PlotYMax = graph.PlotYMax;
   TargetEnviron->BorderLeft = graph.BorderLeft;
   TargetEnviron->BorderRight = graph.BorderRight;
   TargetEnviron->BorderBottom = graph.BorderBottom;
   TargetEnviron->BorderTop = graph.BorderTop;
   TargetEnviron->MaxCount = graph.MaxCount;
   TargetEnviron->XType = graph.XType;
   TargetEnviron->XMin = graph.XMin;
   TargetEnviron->XMax = graph.XMax;
   TargetEnviron->XInc = graph.XInc;
   TargetEnviron->XCount = graph.XCount;
   TargetEnviron->XMinor = graph.XMinor;
   TargetEnviron->YType = graph.YType;
   TargetEnviron->YMin = graph.YMin;
   TargetEnviron->YMax = graph.YMax;
   TargetEnviron->YInc = graph.YInc;
   TargetEnviron->YCount = graph.YCount;
   TargetEnviron->YMinor = graph.YMinor;
   return;
}

int SetEnviron(NewEnviron)
GR_ENVIRONMENT *NewEnviron;
/* set the current graph environment to 'NewEnviron' */
{
   graph.ResX = NewEnviron->ResX;
   graph.ResY = NewEnviron->ResY;
   graph.PlotXMax = NewEnviron->PlotXMax;
   graph.PlotYMax = NewEnviron->PlotYMax;
   graph.BorderLeft = NewEnviron->BorderLeft;
   graph.BorderRight = NewEnviron->BorderRight;
   graph.BorderBottom = NewEnviron->BorderBottom;
   graph.BorderTop = NewEnviron->BorderTop;
   graph.MaxCount = NewEnviron->MaxCount;
   graph.XType = NewEnviron->XType;
   graph.XMin = NewEnviron->XMin;
   graph.XMax = NewEnviron->XMax;
   graph.XInc = NewEnviron->XInc;
   graph.XCount = NewEnviron->XCount;
   graph.XMinor = NewEnviron->XMinor;
   graph.YType = NewEnviron->YType;
   graph.YMin = NewEnviron->YMin;
   graph.YMax = NewEnviron->YMax;
   graph.YInc = NewEnviron->YInc;
   graph.YCount = NewEnviron->YCount;
   graph.YMinor = NewEnviron->YMinor;
```

```
   return;
}

int SetMaxCount(max)
int max;
{
   graph.MaxCount = max;
   return;
}

float GetXMin()
{
   return(graph.XMin);
}

int SetXMin(min)
float min;
{
   graph.XMin = min;
   return;
}

float GetXMax()
{
   return(graph.XMax);
}

int SetXMax(max)
float max;
{
   graph.XMax = max;
   return;
}

float GetYMin()
{
   return(graph.YMin);
}

int SetYMin(min)
float min;
{
   graph.YMin = min;
   return;
}

float GetYMax()
{
   return(graph.YMax);
}

int SetYMax(max)
float max;
{
```

```
   graph.YMax = max;
   return;
}

int GetXCount()
{
   return(graph.XCount);
}

int SetXCount(count)
int count;
{
   graph.XCount = count;
   return;
}

int GetYCount()
{
   return(graph.YCount);
}

int SetYCount(count)
int count;
{
   graph.YCount = count;
   return;
}

float GetXInc()
{
   return(graph.XInc);
}

int SetXInc(inc)
float inc;
{
   graph.XInc = inc;
   return;
}

float GetYInc()
{
   return(graph.YInc);
}

int SetYInc(inc)
float inc;
{
   graph.YInc = inc;
   return;
}

int GetXType()
{
```

```
   return(graph.XType);
}

int GetYType()
{
   return(graph.YType);
}

int matherr(x)
struct exception *x;
{
   fprintf(stdout,"ERROR: %d in %s\n arg1 = %lf\n",
            x->type,x->name,x->arg1);
   x->retval = 100.0;
   return(1);
}

int Text(textstr)
char *textstr;
{
   DrawString(textstr);

/* The following printf statement is a kluge solution to a MetaWindows
   problem with DrawString.  Nothing is sent to the screen, but the
   printf seems to repair the damage done by DrawString.  */

   printf("\r");
   return;
}

int SpecialText(textPtr)
char *textPtr;
{
   int x,y,StartX,StartY,i,j,ch,hilite,wid,TxAlignH,TxAlignV;
   char CleanString[101];

   i=0;
   j=0;
   ch = textPtr[i];
   while (ch != '\0')
   {
      ch = textPtr[i];
      switch(ch)
      {
         case '^':
         case '|':
         default:
            CleanString[j] = textPtr[i];
            j++;
            break;
      }
      i++;
   }
   wid = StringWidth(CleanString);
```

```
    QueryPosn(&x,&y);
    TxAlignH = GetPortField(c_txAlign+c_X);
    TxAlignV = GetPortField(c_txAlign+c_Y);
    switch(TxAlignH)
    {
       case alignLeft:
          break;
       case alignCenter:
          x = x - wid/2 + 1;
          break;
       case alignRight:
          x = x - wid + 1;
          break;
    }
    TextAlign(alignLeft,TxAlignV);
    StartX = x;
    StartY = y;
    MoveTo(x,y);
    j=0;
    ch = textPtr[0];
    while (ch != '\0')
    {
       ch = textPtr[j];
       j++;
       switch(ch)
       {
          case 'I':
             if (GetPortField(c_txMode)==Replace)
                TextMode(Invert);
             else
                TextMode(Replace);
             break;
          case '^':
/* temporary -                                                          */
/* Metagraphics did not include font record definitions, so this code has */
/* no way of determining font character height.  Height is hardwired to 7 */
             QueryPosn(&x,&y);
             MoveTo(x,y+3);
             break;
          default:
             DrawChar(ch);
             break;
       }
    }
    TextAlign(TxAlignH,TxAlignV);
/*temp*/
    printf("\r");
    return;
}

int Swap(first,second)
int *first,*second;

{
```

```
    int temp;

    temp = *first;
    *first = *second;
    *second = temp;
    return;
}

#define VIDEO_INTERRUPT 16
#define SET_COLOR_PALETTE 11

/* ------------------------------------------------------------------------ */

/* SetHiResColor sets the color for the high res screen (CGA 640x200)       */
/* color range (0-15)                                                       */

int SetHiResColor(color)
int color;

{
    union REGS regs;

    regs.h.ah = SET_COLOR_PALETTE;

    /* Note: when BH is zero, BL specifies the foreground color */
    /* for high res mode (640x200)                              */
    regs.h.bh = 0;
    regs.h.bl = color;

    int86(VIDEO_INTERRUPT,&regs,&regs);

    return;
}

/* ************************   METAMENU ROUTINES  ********************** */

#define LINT_ARGS 1
#define TRUE      1
#define FALSE     0
#define NOCHOICE  (-1)
#define SPECIAL   (-2)
#define EMPTY     (-1)

#include "keys.h"
#include "menu.h"

pusharea SaveArea;
MENUSTACK MenuTrail;
static int DeactiveStyle = GONE_STYLE;

MENUBAR *MakeNewMenuBar()
{
    MENUBAR *theMenuBar;
```

```
    theMenuBar = (MENUBAR *)malloc(sizeof(MENUBAR));
    if (theMenuBar != NULL)
       InitMenuBar(theMenuBar);
    return(theMenuBar);
}

int InitMenuBar(theMenuBar)
MENUBAR *theMenuBar;
{
    int i;

    HomeMenuBar(theMenuBar);
    theMenuBar->NumberOfMenus = 0;
    theMenuBar->Selection = 0;
    theMenuBar->Padding[0] = ' ';
    theMenuBar->Padding[1] = '\0';
    theMenuBar->Divider[0] = '\0';
    for (i=0; i<MAX_MENUS_PER_BAR; i++)
    {
       theMenuBar->MenuName[i][0] = '\0';
       theMenuBar->MenuFunction[i] = NULL;
       theMenuBar->NextMenuBar[i] = NULL;
       theMenuBar->MenuDescrip[i][0] = '\0';
       theMenuBar->MenuIsActive[i] = FALSE;
    }
    return;
}

int HomeMenuBar(theMenuBar)
MENUBAR *theMenuBar;
{
    int OldBarX,OldBarY,OldBarWid;

    OldBarX = theMenuBar->BarX;
    OldBarY = theMenuBar->BarY;
    OldBarWid = theMenuBar->BarWid;
    theMenuBar->BarX = GetPortField(c_portRect + c_xmin);
    theMenuBar->BarY = GetPortField(c_portRect + c_ymax);
    theMenuBar->BarWid = GetPortField(c_portRect+c_xmax)-theMenuBar->BarX;
    if ((OldBarX != theMenuBar->BarX)||(OldBarY != theMenuBar->BarY)||
        (OldBarWid != theMenuBar->BarWid))
       theMenuBar->MenuChanged = TRUE;
    return;
}

int FreeMenuBar(theMenuBar)
MENUBAR *theMenuBar;
{
    free((char *)theMenuBar);
    return;
}

int SetMenuStyle(MenuStyle)
int MenuStyle;
```

```
{
   DeactiveStyle = MenuStyle;
}

int SetMenuSpacing(theMenuBar,padding,divider)
MENUBAR *theMenuBar;
char *padding,*divider;
{
   if ((strcmp(padding,&(theMenuBar->Padding[0])))||
       (strcmp(divider,&(theMenuBar->Divider[0]))))
      theMenuBar->MenuChanged = TRUE;
   sprintf(&(theMenuBar->Padding[0]),"%s",padding);
   sprintf(&(theMenuBar->Divider[0]),"%s",divider);
   return;
}

int AddMenuField(theMenuBar,Name,Function,SubMenu,Description)
MENUBAR *theMenuBar,*SubMenu;
int (*Function)();
char *Name,*Description;
{
   int theMenu;

   if ((Function==NULL)&&(SubMenu==NULL))
      return(EMPTY);

   if (theMenuBar->NumberOfMenus < MAX_MENUS_PER_BAR)
   {
      theMenuBar->MenuChanged = TRUE;
      theMenu = theMenuBar->NumberOfMenus;
      theMenuBar->NumberOfMenus += 1;
      sprintf(&(theMenuBar->MenuName[theMenu][0]),"%s",Name);
      theMenuBar->MenuFunction[theMenu] = Function;
      theMenuBar->NextMenuBar[theMenu] = SubMenu;
      sprintf(&(theMenuBar->MenuDescrip[theMenu][0]),"%s",Description);
      theMenuBar->MenuIsActive[theMenu] = TRUE;
   }
   return(theMenu);
}

int EditMenuField(theMenuBar,theMenu,Name,Function,SubMenu,Description)
MENUBAR *theMenuBar,*SubMenu;
int (*Function)();
int theMenu;
char *Name,*Description;
{
   if ((Function==NULL)&&(SubMenu==NULL))
      return(EMPTY);

   if (theMenu < theMenuBar->NumberOfMenus)
   {
      if ((strcmp(Name,&(theMenuBar->MenuName[theMenu][0])))||
          (theMenuBar->MenuFunction[theMenu] != Function)||
          (theMenuBar->NextMenuBar[theMenu] != SubMenu)||
```

```
          (strcmp(Description,&(theMenuBar->MenuDescrip[theMenu][0]))))
         theMenuBar->MenuChanged = TRUE;
      sprintf(&(theMenuBar->MenuName[theMenu][0]),"%s",Name);
      theMenuBar->MenuFunction[theMenu] = Function;
      theMenuBar->NextMenuBar[theMenu] = SubMenu;
      sprintf(&(theMenuBar->MenuDescrip[theMenu][0]),"%s",Description);
   }
   return(0);
}

int SetSpecialKey(theMenuBar,theKey,Function)
MENUBAR *theMenuBar;
char theKey;
int (*Function)();
{
   theMenuBar->SpecialKey = theKey;
   theMenuBar->KeyFunction = Function;
   return;
}

int SelectMenu(theMenuBar,theMenu)
MENUBAR *theMenuBar;
int theMenu;
{
   if ((theMenuBar->MenuIsActive[theMenu])||(DeactiveStyle==GRAY_STYLE))
      theMenuBar->Selection = theMenu;
   return;
}

int ActivateMenu(theMenuBar,theMenu)
MENUBAR *theMenuBar;
int theMenu;
{
   if ((theMenu < theMenuBar->NumberOfMenus)&&
       (!theMenuBar->MenuIsActive[theMenu]))
   {
      theMenuBar->MenuChanged = TRUE;
      theMenuBar->MenuIsActive[theMenu] = TRUE;
   }
   return;
}

int DeactivateMenu(theMenuBar,theMenu)
MENUBAR *theMenuBar;
int theMenu;
{
   if ((theMenu < theMenuBar->NumberOfMenus)&&
       (theMenuBar->MenuIsActive[theMenu]))
   {
      theMenuBar->MenuChanged = TRUE;
      theMenuBar->MenuIsActive[theMenu] = FALSE;
   }
   return;
}
```

```
int RefreshMenuBar(theMenuBar)
MENUBAR *theMenuBar;
{
   if (theMenuBar->MenuChanged)
      DrawMenuBar(theMenuBar);
   theMenuBar->MenuChanged = FALSE;
   return;
}

int DrawMenuBar(theMenuBar)
MENUBAR *theMenuBar;
{
   BlankMenuBar(theMenuBar);
   DrawMainBar(theMenuBar);
   DrawDescBar(theMenuBar);
   InvertSelectedMenu(theMenuBar);
   return;
}

int EraseMenuBar(theMenuBar)
MENUBAR *theMenuBar;
{
   int CharHeight;
   rect MenuR;

   theMenuBar->MenuChanged = TRUE;

   /*temp - Metagraphics did not include font definitions.  Font height  */
   /* is temporarily hardwired to 8 pixels                               */
   CharHeight = 8;

   SetRect(&MenuR,theMenuBar->BarX,theMenuBar->BarY-(2*CharHeight+3),
           theMenuBar->BarX+theMenuBar->BarWid,theMenuBar->BarY);
   FillRect(&MenuR,0);
   return;
}

int BlankMenuBar(theMenuBar)
MENUBAR *theMenuBar;
{
   int CharHeight;
   rect MenuR;

   PushGrafix(&SaveArea);

   theMenuBar->MenuChanged = TRUE;

   /*temp - Metagraphics did not include font definitions.  Font height  */
   /* is temporarily hardwired to 8 pixels                               */
   CharHeight = 8;

   PenColor(1);
   PenMode(Replace);
```

```
    SetRect(&MenuR,theMenuBar->BarX,theMenuBar->BarY-(CharHeight+2),
            theMenuBar->BarX+theMenuBar->BarWid,theMenuBar->BarY);
    FrameRect(&MenuR);
    InsetRect(&MenuR,1,1);
    EraseRect(&MenuR);
    SetRect(&MenuR,theMenuBar->BarX,
            theMenuBar->BarY-(2*CharHeight+3),
            theMenuBar->BarX+theMenuBar->BarWid,
            theMenuBar->BarY-(CharHeight+3));
    FillRect(&MenuR,1);

    PopGrafix(&SaveArea);

    return;
}

int DrawMainBar(theMenuBar)
MENUBAR *theMenuBar;
{
    char MenuBar[MAX_CHARS_PER_BAR+5];
    int i,CharHeight,Position,theMenu,x,y;
    MENUBAR *SubMenu;
    rect MenuR;

    PushGrafix(&SaveArea);

    /* build the Menu Bar string */
    Position = 0;
    for (i=0; i<theMenuBar->NumberOfMenus; i++)
        if ((theMenuBar->MenuIsActive[i])||(DeactiveStyle==GRAY_STYLE))
            Position += sprintf(MenuBar+Position,"%s%s%s%s",
                                &(theMenuBar->Padding[0]),
                                &(theMenuBar->MenuName[i][0]),
                                &(theMenuBar->Padding[0]),
                                &(theMenuBar->Divider[0]));

    /*temp - Metagraphics did not include font definitions.  Font height  */
    /* is temporarily hardwired to 8 pixels                               */
    CharHeight = 8;
    TextMode(Replace);
    TextAlign(alignLeft,alignBaseline);

    PenMode(Replace);
    PenPattern(1);
    SetRect(&MenuR,theMenuBar->BarX,theMenuBar->BarY-(CharHeight+2),
            theMenuBar->BarX+theMenuBar->BarWid,theMenuBar->BarY);
    FrameRect(&MenuR);
    x = theMenuBar->BarX+1;
    y = theMenuBar->BarY-CharHeight;
    MoveTo(x,y);
    Text(MenuBar);
    if (DeactiveStyle == GRAY_STYLE)
        for (i=0; i<theMenuBar->NumberOfMenus; i++)
            if (!theMenuBar->MenuIsActive[i])
```

```
            GrayMenu(theMenuBar,i);

    PopGrafix(&SaveArea);

    return;
}

int DrawDescBar(theMenuBar)
MENUBAR *theMenuBar;
{
    char DescBar[MAX_CHARS_PER_BAR+1];
    int i,CharHeight,Position,theMenu,x,y;
    MENUBAR *SubMenu;
    rect DescR;

    PushGrafix(&SaveArea);

    /* build the Description Bar string */
    Position = 0;
    theMenu = theMenuBar->Selection;
    SubMenu = theMenuBar->NextMenuBar[theMenu];
    if (SubMenu == NULL)
       sprintf(DescBar," %s",&(theMenuBar->MenuDescrip[theMenu][0]));
    else
       for (i=0; i<SubMenu->NumberOfMenus; i++)
          if ((SubMenu->MenuIsActive[i])||(DeactiveStyle==GRAY_STYLE))
             Position += sprintf(DescBar+Position,"%s%s%s%s",
                                 &(SubMenu->Padding[0]),
                                 &(SubMenu->MenuName[i][0]),
                                 &(SubMenu->Padding[0]),
                                 &(SubMenu->Divider[0]));

    /*temp - Metagraphics did not include font definitions.  Font height  */
    /* is temporarily hardwired to 8 pixels                               */
    CharHeight = 8;
    TextMode(Invert);
    TextAlign(alignLeft,alignBaseline);

    PenMode(Replace);
    PenPattern(1);
    SetRect(&DescR,theMenuBar->BarX,
            theMenuBar->BarY-(2*CharHeight+3),
            theMenuBar->BarX+theMenuBar->BarWid,
            theMenuBar->BarY-(CharHeight+3));
    FillRect(&DescR,1);
    x = theMenuBar->BarX+1;
    y = theMenuBar->BarY-(2*CharHeight+3)+2;
    MoveTo(x,y);
    PenMode(zORz);
    PenPattern(GRAYPATTERN);
    Text(DescBar);
    if ((!theMenuBar->MenuIsActive[theMenu])&&(DeactiveStyle==GRAY_STYLE))
       PaintRect(&DescR);
```

```
    PopGrafix(&SaveArea);

    return;
}

int InvertSelectedMenu(theMenuBar)
MENUBAR *theMenuBar;
{
    char MenuBar[MAX_CHARS_PER_BAR+5];
    int i,CharHeight,Position,x,y;
    rect MenuR;

    PushGrafix(&SaveArea);

    /* build the Menu Bar string */
    Position = 0;
    MenuBar[0] = '\0';
    for (i=0; i<theMenuBar->Selection; i++)
        if ((theMenuBar->MenuIsActive[i])||(DeactiveStyle==GRAY_STYLE))
            Position += sprintf(MenuBar+Position,"%s%s%s%s",
                                &(theMenuBar->Padding[0]),
                                &(theMenuBar->MenuName[i][0]),
                                &(theMenuBar->Padding[0]),
                                &(theMenuBar->Divider[0]));

    /*temp - Metagraphics did not include font definitions.  Font height  */
    /* is temporarily hardwired to 8 pixels                                */
    CharHeight = 8;

    PenMode(Invert);
    PenPattern(1);
    x = theMenuBar->BarX+StringWidth(MenuBar)+1;
    y = theMenuBar->BarY-(CharHeight+1);
    sprintf(MenuBar,"%s%s%s",&(theMenuBar->Padding[0]),
                             &(theMenuBar->MenuName[i][0]),
                             &(theMenuBar->Padding[0]));
    SetRect(&MenuR,x,y,x+StringWidth(MenuBar),y+CharHeight);
    PaintRect(&MenuR);

    PopGrafix(&SaveArea);

    return;
}

int GrayMenu(theMenuBar,theMenu)
MENUBAR *theMenuBar;
int theMenu;
{
    char MenuBar[MAX_CHARS_PER_BAR+5];
    int i,CharHeight,Position,x,y;
    rect MenuR;

    PushGrafix(&SaveArea);
```

```
   /* build the Menu Bar string */
   Position = 0;
   MenuBar[0] = '\0';
   for (i=0; i<theMenu; i++)
      if ((theMenuBar->MenuIsActive[i])||(DeactiveStyle==GRAY_STYLE))
         Position += sprintf(MenuBar+Position,"%s%s%s%s",
                             &(theMenuBar->Padding[0]),
                             &(theMenuBar->MenuName[i][0]),
                             &(theMenuBar->Padding[0]),
                             &(theMenuBar->Divider[0]));

   /*temp - Metagraphics did not include font definitions.  Font height  */
   /* is temporarily hardwired to 8 pixels                               */
   CharHeight = 8;

   PenMode(zANDz);
   PenPattern(GRAYPATTERN);
   x = theMenuBar->BarX+StringWidth(MenuBar)+1;
   y = theMenuBar->BarY-(CharHeight+1);
   sprintf(MenuBar,"%s%s%s",&(theMenuBar->Padding[0]),
                            &(theMenuBar->MenuName[i][0]),
                            &(theMenuBar->Padding[0]));
   SetRect(&MenuR,x,y,x+StringWidth(MenuBar),y+CharHeight);
   PaintRect(&MenuR);

   PopGrafix(&SaveArea);

   return;
}

int GetMenuChoice(theMenuBar)
MENUBAR *theMenuBar;
{
   int Choice,i;
   char ch;

   Choice = NOCHOICE;
   do
   {
      ch = GetKey();
      DumpKeyBuffer();
      if (ch == theMenuBar->SpecialKey)
         return(SPECIAL);
      switch (ch)
      {
         case ESC:
            return(NOCHOICE);
            break;
         case RETURN:
            Choice = theMenuBar->Selection;
            break;
         case PAGEUP:
         case UPARROW:
         case LEFTARROW:
```

```
    InvertSelectedMenu(theMenuBar);
    if (DeactiveStyle == GRAY_STYLE)
    {
       (theMenuBar->Selection)--;
       if (theMenuBar->Selection < 0)
          theMenuBar->Selection = theMenuBar->NumberOfMenus-1;
    }
    else
       do
       {
          (theMenuBar->Selection)--;
          if (theMenuBar->Selection < 0)
             theMenuBar->Selection = theMenuBar->NumberOfMenus-1;
       } while (!theMenuBar->MenuIsActive[theMenuBar->Selection]);
    InvertSelectedMenu(theMenuBar);
    DrawDescBar(theMenuBar);
    break;
case PAGEDOWN:
case DNARROW:
case RIGHTARROW:
    InvertSelectedMenu(theMenuBar);
    if (DeactiveStyle == GRAY_STYLE)
    {
       (theMenuBar->Selection)++;
       if (theMenuBar->Selection >= theMenuBar->NumberOfMenus)
          theMenuBar->Selection = 0;
    }
    else
       do
       {
          (theMenuBar->Selection)++;
          if (theMenuBar->Selection >= theMenuBar->NumberOfMenus)
             theMenuBar->Selection = 0;
       } while (!theMenuBar->MenuIsActive[theMenuBar->Selection]);
    InvertSelectedMenu(theMenuBar);
    DrawDescBar(theMenuBar);
    break;
case HOME:
    InvertSelectedMenu(theMenuBar);
    if (DeactiveStyle == GRAY_STYLE)
       theMenuBar->Selection = 0;
    else
    {
       theMenuBar->Selection = -1;
       do
       {
          (theMenuBar->Selection)++;
       } while (!theMenuBar->MenuIsActive[theMenuBar->Selection]);
    }
    InvertSelectedMenu(theMenuBar);
    DrawDescBar(theMenuBar);
    break;
case END:
    InvertSelectedMenu(theMenuBar);
```

```
            if (DeactiveStyle == GRAY_STYLE)
               theMenuBar->Selection = theMenuBar->NumberOfMenus-1;
            else
            {
               theMenuBar->Selection = theMenuBar->NumberOfMenus;
               do
               {
                  (theMenuBar->Selection)--;
               } while (!theMenuBar->MenuIsActive[theMenuBar->Selection]);
            }
            InvertSelectedMenu(theMenuBar);
            DrawDescBar(theMenuBar);
            break;
         default:
#define CASEMASK 223 /* 11011111 */
            for (i=0; i<theMenuBar->NumberOfMenus; i++)
               if ((ch & CASEMASK)==(theMenuBar->MenuName[i][0] & CASEMASK))
               {
                  Choice = i;
                  if (theMenuBar->Selection != Choice)
                  {
                     InvertSelectedMenu(theMenuBar);
                     SelectMenu(theMenuBar,Choice);
                     InvertSelectedMenu(theMenuBar);
                     DrawDescBar(theMenuBar);
                  }
               }
               if (Choice < 0)
                  printf("%c",7);
            break;
      }
   } while (Choice<0);
   return(Choice);
}

int PushMenu(theMenuBar)
MENUBAR *theMenuBar;
{
   if (MenuTrail.StackTop < MENUDEPTH)
   {
      MenuTrail.StackTop++;
      MenuTrail.MenuBarPtr[MenuTrail.StackTop] = theMenuBar;
   }
   return;
}

MENUBAR *PopMenu()
{
   if (MenuTrail.StackTop > EMPTY)
      MenuTrail.StackTop--;
   return(MenuTrail.MenuBarPtr[MenuTrail.StackTop]);
}

int MenuService(theMenuBar)
```

```
MENUBAR *theMenuBar;
{
   int Choice,i,BackTrack;
   MENUBAR *CurrentMenu;
   int (*Service)();

   MenuTrail.StackTop = EMPTY;
   PushMenu(theMenuBar);
   CurrentMenu = theMenuBar;
   do
   {
      RefreshMenuBar(CurrentMenu);
      Choice = GetMenuChoice(CurrentMenu);
      switch (Choice)
      {
         case NOCHOICE:
            CurrentMenu->MenuChanged = TRUE;
            CurrentMenu = PopMenu();
            break;
         case SPECIAL:
            Service = CurrentMenu->KeyFunction;
            BackTrack = (*Service)();
            if (BackTrack < 0)
               MenuTrail.StackTop = EMPTY;
            else
               for (i=0; i<BackTrack; i++)
               {
                  CurrentMenu->MenuChanged = TRUE;
                  CurrentMenu = PopMenu();
               }
            break;
         default:
            if (CurrentMenu->MenuIsActive[Choice])
            {
               if (CurrentMenu->NextMenuBar[Choice] == NULL)
               {
                  Service = CurrentMenu->MenuFunction[Choice];
                  BackTrack = (*Service)();
                  if (BackTrack < 0)
                     MenuTrail.StackTop = EMPTY;
                  else
                     for (i=0; i<BackTrack; i++)
                     {
                        CurrentMenu->MenuChanged = TRUE;
                        CurrentMenu = PopMenu();
                     }
               }
               else
               {
                  CurrentMenu->MenuChanged = TRUE;
                  CurrentMenu = CurrentMenu->NextMenuBar[Choice];
                  PushMenu(CurrentMenu);
               }
            }
```

```
            else
               bell();
            break;
      }
   } while (MenuTrail.StackTop != EMPTY);
   return;
}

int PrintMenuRecord(theMenuBar,stream)
MENUBAR *theMenuBar;
FILE *stream;
{
   int i;
   fprintf(stream,"Menu record starting at %lx:\n",theMenuBar);
   fprintf(stream,"MenuChanged = %d\n",theMenuBar->MenuChanged);
   fprintf(stream,"BarX = %d, BarY = %d\nBarWid = %d\nNumberOfMenus = %d\n",
           theMenuBar->BarX,theMenuBar->BarY,theMenuBar->BarWid,
           theMenuBar->NumberOfMenus);
   fprintf(stream,"Selection = %d\n",theMenuBar->Selection);
   for (i=0; i<theMenuBar->NumberOfMenus; i++)
      fprintf(stream,"Name: %s\nNext: %lx\nDesc: %s\nAct?: %d\n\n",
              &(theMenuBar->MenuName[i][0]),
              theMenuBar->NextMenuBar[i],
              &(theMenuBar->MenuDescrip[i][0]),
              theMenuBar->MenuIsActive[i]);
   return;
}

int GetKey()
{
   int ch;
   ch = getch();
   if (!ch)
      ch = -getch();
   return(ch);
}

int DumpKeyBuffer()
{
   while (kbhit())
      getch();
   return;
}

/* ************************ SAVIDEFS.H include file ************************** */

#include <pathdef.h>

/* ---------------------- SAVI constant definitions ----------------------- */

/*                                MISCELLANEOUS                              */

/* SAVI version number */
#define SAVI_VERSION 2.2
```

```
#define DELAY 0
#define NONDET 1

#define USER 0
#define PROG 1

#define FALSE 0
#define TRUE  1

#define INTERIOR 0
#define EXTERIOR 1

#define SOURCE 0
#define DESTINATION 1

#define MORE_UP 0
#define MORE_DN 1

/* EMPTY is used to indicate an empty type/location pair */
#define EMPTY -1

/* CLEAR and DIRECT are choice settings for a COMP_SETTING choice field */
#define CLEAR -1
#define DIRECT -2

/* INFINITY is a delay component value indicating infinite delay */
#define INFINITY -2

/* INFINITE_VAL is the floating point value which represents infinite */
/* time delay (the maximum time delay supported in SAVI is 5999 secs) */
/* NOTE: this value must not exceed maxint                            */
#define INFINITE_VAL 10000.00

#define NOT_FOUND -1

/* INPUT_SEG, OUTPUT_SEG and EITHER_SEG indicate the location of  */
/* delay/detection components within a PE.  EITHER_SEG indicates  */
/* that the component may occur on either (or both) path segments */
/* in the PE traversal equations                                  */
#define INPUT_SEG 0
#define OUTPUT_SEG 1
#define EITHER_SEG 2

/* AUTH and UNAUTH refer to authorized and unauthorized traversal */
/* of a PE                                                        */
#define AUTH 0
#define UNAUTH 1

/* FORCE and MIXED refer to threat strategies                */
/* FORCE indicates that all PEs will be traversed by force */
/* MIXED indicates that the most vulnerable traversal      */
/* will be made by combining force and deceit tactics      */
#define FORCE 0
```

```
#define MIXED 1

/* ENTRY, EXIT and EXIT_DEP are traversal direction states */
/* ENTRY indicates traversal toward the target             */
/* EXIT indicates traversal away from the target           */
/* EXIT_DEP indicates traversal away from the target along */
/* a path segment which was previously traversed toward    */
/* the target (thus, exit-dependent)                       */
#define ENTRY 0
#define EXIT 1
#define EXIT_DEP 2

/* SEGMENT_LEVEL and COMPONENT_LEVEL refer to analysis types */
/* indicating where the path segment values are derived for  */
/* a call to the path analysis code                          */
#define SEGMENT_LEVEL 0
#define COMPONENT_LEVEL 1

/* MAX_LESS_THAN_MIN and RANGE_TOO_SMALL are RFT range errors */
#define MAX_LESS_THAN_MIN 1
#define RANGE_TOO_SMALL 2

/* default popup window dimensions */
#define POPUP_TOP 2
#define POPUP_HEIGHT 23
#define POPUP_WIDTH 38

/* popup window beginning column values for */
/* left and right placement of the window   */
/* to allow any PE to be visible            */
#define POPUP_RIGHT_COL 41
#define POPUP_LEFT_COL 1

/* UNKNOWN refers to unknown component selections */
#define UNKNOWN -1

/* NON_PRESENT is used to indicate a layer which does */
/* not have a PE on the path in question              */
#define NON_PRESENT -1

/* PATH_IN and PATH_OUT represent paths leading into and out of a PE */
#define PATH_IN 0
#define PATH_OUT 1

/* FOOT, TRUCK and HELICOPTER are threat transport modes */
#define FOOT 0
#define TRUCK 1
#define HELICOPTER 2

/* TIMELY_DETECTION_OK, NO_TIMELY_DETECTION, PATH_DELAY_DEFICIENCY and    */
/* TDP_OFFSITE are the values which are returned from the                 */
/* timely_detection_point function                                        */
#define TIMELY_DETECTION_OK 0
#define NO_TIMELY_DETECTION 1
```

```
#def ne PATH_DELAY_DEFICIENCY 2
#define TDP_OFFSITE 3

/* DRIVE_OK, DRIVE_NOT_READY, DISK_NOT_READABLE are the error values      */
/* which are returned from the check_drive function                       */
#define DRIVE_OK 0
#define DRIVE_NOT_READY 1
#define DISK_NOT_READABLE 2

/* IMPROVE_DELAY and IMPROVE_DETECTION are the constants for path segment */
/* improvement as suggested by the critical path analysis function        */
#define IMPROVE_DELAY     0
#define IMPROVE_DETECTION 1

/* these are the standard catalog and startup ASD file names */
#define PE_CATALOG_FILE_NAME "PE.CAT"
#define DELAY_CATALOG_FILE_NAME "DEL.CAT"
#define NONDET_CATALOG_FILE_NAME "DET.CAT"
#define STARTUP_ASD_FILE_NAME "STARTUP.ASD"
#define THREAT_CATALOG_FILE_NAME "THREAT.CAT"
#define GRAPHIC_FONT_NAME "SAVIFONT.FNT"

/*                              LABEL LENGTHS                             */

/* Note: these values include an extra character to hold the terminating  */
/*       null byte required at the end of strings.                        */

#define MAX_ASD_LABEL_LEN    49
#define MAX_AREA_LABEL_LEN   28
#define MAX_HALF_AREA_LABEL_LEN 14
#define MAX_PE_LABEL_LEN     35
#define MAX_PE_ACRONYM_LEN    4
#define MAX_COMP_LABEL_LEN   35
#define MAX_ERROR_STRING_LEN 38
#define MAX_THREAT_LABEL_LEN 45

/*                              ARRAY LIMITS                              */

#define MAX_PE_PER_LAYER         11
#define MAX_DISPLAY_PE_PER_LAYER 11

/* MAX_LAYERS_PER_ASD is the number of layers with PEs which are deletable */
/* within any ASD. Note that the Task PE is not deletable and is stored in */
/* its own field within the ASD structure. The Task PE can be thought of as */
/* residing on layer number MAX_LAYERS_PER_ASD                             */
#define MAX_LAYERS_PER_ASD 5

/* MAX_MODIFIABLE_AREAS is the number of areas which can have the label    */
/* modified by the user. Area 0 (OFFSITE) and the last Area (TARGET) cannot */
/* be modified                                                             */
#define MAX_MODIFIABLE_AREAS 5

/* MAX_COMP_PER_PE  is the maximum */
/* number of each component type   */
```

```
/* (delay or non-detection) per     */
/* protection element               */
#define MAX_COMP_PER_PE          20

#define MAX_EQNS_PER_SEGMENT     10

/* MAX_PE_CATALOG_ENTRIES is the maximum number of PE types      */
/* which can be entered into a single PE catalog  (two exist)    */
#define MAX_PE_CATALOG_ENTRIES   10

/* MAX_COMP_CATALOG_ENTRIES is the maximum number of components  */
/* which can be entered into a single component catalog          */
/* (two component catalogs exist for both delay and detection    */
#define MAX_COMP_CATALOG_ENTRIES 25

/* MAX_RFTS is the number of response force times that can be    */
/* specified for analysis within the analyze function            */
#define MAX_RFTS 10

/* TOP_VULNERABLE_PATHS is the number of paths which will have   */
/* vulnerability results displayed for a given analysis          */
#define TOP_VULNERABLE_PATHS 10

/* LAST_DEFINED_PE_TYPE is the highest number PE type which is   */
/* defined within the PE catalog. This number must not exceed    */
/* (2*MAX_PE_CATALOG_ENTRIES)-2 since the last PE type number    */
/* is reserved for the TASK type                                 */
#define LAST_DEFINED_PE_TYPE     14

#define LAST_DEFINED_DELAY_COMP 15
#define LAST_DEFINED_NONDET_COMP 30

/* TASK_PE_TYPE is the type number for the TASK PE. This should  */
/* be set to (2*MAX_PE_CATALOG_ENTRIES)-1 which is the last PE   */
/* entry number                                                  */
#define TASK_PE_TYPE             19

#define MAX_COMP_PER_EQN         10
#define MAX_CHOICES_PER_COMPTYPE 20
#define MAX_THREATS              10

/* NUM_VALID_THREATS indicates the number of threats which    */
/* are currently supported in SAVI  (Note: the valid threats  */
/* must be contiguous in the low end of the threat table)     */
#define NUM_VALID_THREATS        3

/* MAX_AREAS_PER_ASD should be MAX_LAYERS_PER_ASD + 2          */
#define MAX_AREAS_PER_ASD         7

/* ----------------------- SAVI Macro Definitions ----------------------- */

#define max(a,b) (((a) > (b)) ? (a) : (b))
#define min(a,b) (((a) < (b)) ? (a) : (b))
```

```
/* ---------------------- SAVI Structure Definitions ---------------------- */

/*                          Two Line Label Structure                        */

typedef struct {
               char first_line[MAX_HALF_AREA_LABEL_LEN],
                    second_line[MAX_HALF_AREA_LABEL_LEN];
               } TWO_LINE_LABEL;

/*                          Segment Equation Structure                      */

typedef struct {
               int comptype[MAX_COMP_PER_EQN];
               } SEGMENT_EQN;

/*                        Traversal Equations Structure                     */

typedef struct {
               SEGMENT_EQN entry_input_eqns[MAX_EQNS_PER_SEGMENT],
                           entry_output_eqns[MAX_EQNS_PER_SEGMENT],
                           exit_input_eqns[MAX_EQNS_PER_SEGMENT],
                           exit_output_eqns[MAX_EQNS_PER_SEGMENT];
               } TRAVERSAL_EQNS;

/*                  Component Type/Location Pair Structure                  */

typedef struct {
               int comptype,
                   location;
               } TYPE_LOC_PAIR;

/*                          Component Setting Structure                     */

typedef struct {
               int choice;
               float direct_setting;
               int panic_bar;
               } COMP_SETTING;

/*                          Protection Element Structure                    */

typedef struct {
               int present;
               int type;
               float user_delay_in,  user_nondet_in,
                     user_delay_out, user_nondet_out;
               COMP_SETTING del_comp_settings[MAX_COMP_PER_PE],
                            nondet_comp_settings[MAX_COMP_PER_PE];
               int input_area,
                   output_area;
               int transit_dist;
               } PROT_ELEM;

/*                     Adversary Sequence Diagram Structure                 */
```

```
typedef struct {
                float version;
                char label[MAX_ASD_LABEL_LEN];
                TWO_LINE_LABEL area_labels[MAX_MODIFIABLE_AREAS];
                int area_dist[MAX_MODIFIABLE_AREAS];
                int area_type[MAX_MODIFIABLE_AREAS];
                PROT_ELEM model[MAX_LAYERS_PER_ASD][MAX_PE_PER_LAYER];
                PROT_ELEM task_PE;
                int bypasses[MAX_LAYERS_PER_ASD];
                int analysis_state;
                int rft_table[MAX_RFTS];
                int num_rfts;
                float threshold_pi;
                int current_threat;
                int current_direction;
                int current_strategy;
                } ASD_STRUCT;

/*              Protection Element Specifications Structure                 */

typedef struct {
                char label[MAX_PE_LABEL_LEN];
                char acronym[MAX_PE_ACRONYM_LEN];
                int authorized;
                int ext_int;
                int helicopter;
                TYPE_LOC_PAIR del_comp_list[MAX_COMP_PER_PE],
                              nondet_comp_list[MAX_COMP_PER_PE];
                TRAVERSAL_EQNS unauthorized_del_eqns,
                               unauthorized_nondet_eqns,
                               authorized_del_eqns,
                               authorized_nondet_eqns;
                } PE_SPECS;

/*                    Component Specifications Structure                    */

typedef struct {
                char label[MAX_COMP_LABEL_LEN];
                float threat_values_table[MAX_CHOICES_PER_COMPTYPE]
                                         [MAX_THREATS];
                char comp_choice_labels[MAX_CHOICES_PER_COMPTYPE]
                                       [MAX_COMP_LABEL_LEN];
                int panic_bar;
                int snm;
                } COMP_SPECS;

/*                           Threat Catalog Structure                       */

typedef struct {
                char threat_labels[MAX_THREATS]
                                  [MAX_THREAT_LABEL_LEN];
                int threat_transport_modes[MAX_THREATS];
                } THREAT_CATALOG;
```

```
/* ----------------------- Function Declarations -------------------------- */

/* all functions which return non-integer results are declared here          */

PE_SPECS *get_PE_cat_spec_ptr();
PROT_ELEM *get_ASD_PE_ptr();
PROT_ELEM *get_ASD_task_PE_ptr();
COMP_SPECS *get_delay_comp_spec_ptr();
COMP_SPECS *get_nondet_comp_spec_ptr();
THREAT_CATALOG *get_threat_cat_ptr();
char *get_ASD_label();
char *get_full_ASD_area_label();
char *get_ASD_area_label_first_line();
char *get_ASD_area_label_second_line();
char *get_threat_label();
char *cardinal_number();
float get_ASD_threshold_pi();
float find_delay_comp_choice_val();
float find_nondet_comp_choice_val();
float convert_indep_delay_val();
float convert_indep_nondet_val();
float invert_det_nondet();
float get_prob_interruption();
float free_space_on_drive();
float trunc_float();
float get_transport_speed();
float combined_area_transit_time();
float area_transit_time();
float element_transit_time();
int *get_ASD_rft_table_ptr();

/* the following declarations are for all functions called from the menu     */

/* these functions are declared because the rd_menu function in Windows for  */
/* Data requires explicit declaration of the return type which is used to    */
/* control menu processing upon return                                       */

int savi_analyze();
int savi_analyze_results_vulnerability();
int savi_analyze_results_recommendations();
int savi_analyze_results_graphs();
int savi_analyze_results_quit();
int savi_analyze_print_vulnerability();
int savi_analyze_print_diagnostics();
int savi_analyze_print_recommendations();
int savi_analyze_print_quit();
int savi_analyze_quit();
int savi_threat_attributes();
int savi_threat_objective();
int savi_threat_strategy();
int savi_threat_quit();
int savi_define_segments();
int savi_define_component_settings();
```

```
int savi_define_component_copy();
int savi_define_component_quit();
int savi_define_distance_areas();
int savi_define_distance_elements();
int savi_define_distance_quit();
int savi_define_response();
int savi_define_quit();
int savi_quit_no();
int savi_quit_yes();
int savi_modify_title();
int savi_modify_area_labels();
int savi_modify_area_traversal();
int savi_modify_area_quit();
int savi_modify_element_presence();
int savi_modify_element_jumps();
int savi_modify_element_types();
int savi_modify_element_quit();
int savi_modify_quit();
int savi_print_diagram();
int savi_print_tables();
int savi_print_quit();
int savi_file_retrieve();
int savi_file_save();
int savi_file_directory();
int savi_file_quit();

/* ********************* end SAVIDEFS.H include file ********************** */
#define NODE_LIMIT          25
#define THRESH_NEVER_PASSED (-1)

#define OUT_OF_MEMORY       -1
#define BAD_NODE_NUM        -2     /* child num <= parent num     */
#define BAD_PARENT          -3     /* parent limit exceeded       */
#define SEAPATH_INITED      -4     /* free_tree before re-initing */
#define CHILD_REINSERTED    -5     /* this child already inserted */
#define READ_FAILURE        -6     /* failure in load_results     */

typedef struct PATH_INFO{
float cum_prod;   /* these are the accumulated costs         */
int cum_sum;      /* along this path                         */
int break_point;  /* node_num where cumulative_sum exceeded  */
                  /* threshold and cumulative_prod began     */
int number_of_nodes;    /* number of nodes along this path   */
int nodes[NODE_LIMIT];       /* the nodes array contains     */
                             /* the node_num for each        */
                             /* node along this path.        */
int user[NODE_LIMIT];   /* user array for use by quantify    */
} PATH_INFO;

float get_num_paths();
float get_valid_paths();
float turbo_SEApath();
PATH_INFO *get_path_ptr();
float *get_frequency_ptr();
```

```
#define LINT_ARGS       1

#define MAX_ARRAY_SIZE  20
#define MAX_RETURNED    100
#define MAX_THRESH      10
#define FREQ_BINS       20
#define INFINITY        32767
#define TRUE            1
#define FALSE           0
#define NIL             0
/* #define NIL             0L */
#define EMPTY           (-1)
#define OK              0
#define ABSENT          (-1)
#define CHECKLOOPS      100
#define ESC             27

static int allocated=FALSE;   /* flag = TRUE is tree/path space is allocated */
int midpoint;                 /*location of the target in the stack*/
int dependency;
int destination;
int origin;
int target;
int breakpoint;
int maxp;                      /*maximum number of paths in the paths array */
int maxt;                     /*maximum number of thresholds*/
int path_sum;
int in_headnodes;
int headnodes;
int n_thresholds;
int *threshold;
int index[MAX_THRESH*MAX_RETURNED];
float path_prod;
float valid_paths[MAX_THRESH];
float frequency[MAX_THRESH][FREQ_BINS];

PATH_INFO **path_array_ptr;
PATH_INFO *paths;
PATH_INFO *path_location;

typedef struct SEGMENT_INFO {
   int link_num;  /* unique ID of this segment...                        */
   int node_num;  /* node_num is the graph number of the node            */
                  /* that this path segment leads to. Note that          */
                  /* node_num must be greater than the number            */
                  /* of the node that this segment is leading            */
                  /* from (a child node must have a greater              */
                  /* node_num than its parent node).                     */
   float prod_entry,prod_exit,prod_ex_dep;   /* these values are the     */
   int sum_entry,sum_exit,sum_ex_dep;        /* segment costs            */
   struct SEGMENT_INFO *next_segment;  /* this is a pointer to the       */
                                       /* next path segment that leads   */
                                       /* from the same node that this   */
                                       /* segment leads from.            */
```

```
    int visited;   /* turbo_SEApath work field                          */
    int user_field;
    } SEGMENT_INFO  ;


SEGMENT_INFO **tree; /* pointer to the current tree structure */
int numlinks; /* number of links in the current tree structure */

struct  {
        SEGMENT_INFO *index[NODE_LIMIT];
        int stack_top;
        } STACK;

/*global*/  int true_name(int );
/*global*/  struct SEGMENT_INFO *backtrack(void);
/*global*/  int load_stack(struct SEGMENT_INFO *);
/*global*/  int reset(struct SEGMENT_INFO *);
/*global*/  float get_num_paths(void);
/*global*/  float turbo_SEApath(int *,int );
/*global*/  int insert_link(int ,int ,int ,float ,int ,float ,int ,float ,int );
/*global*/  int DoInsert(int ,int ,int ,int ,float ,int ,float ,int ,
            float ,int );
/*global*/  int free_tree(void);
/*global*/  int turbo_init(int ,int ,int ,int ,int ,int );
/*global*/  int save_results(struct _iobuf *);
/*global*/  int get_results_size(void);
/*global*/  int load_results(struct _iobuf *);
/*global*/  struct PATH_INFO *get_path_ptr(int ,int );
/*global*/  float get_valid_paths(int );
/*global*/  int get_displayable_paths(int );
/*global*/  int quantify(void);
/*global*/  float *get_frequency_ptr(int );
/*global*/  int init_frequency_array(void);
/*global*/  int init_valid_paths_array(void);
#define LINT_ARGS       1

#define TRUE            1
#define FALSE           0
#define ENV_STACK_LIMIT 16
#define EMPTY           (-1)
#define OK              0
#define NOT_OK          1
#define XAXIS           0
#define YAXIS           1
#define TOPAXIS         0
#define BOTTOMAXIS      1
#define LEFTAXIS        2
#define RIGHTAXIS       3
#define LINTYPE         0
#define LOGTYPE         1
#define LEFT            0
#define MIDDLE          1
#define RIGHT           2
#define LABEL_LENGTH    10
```

```
#define INT_LABELS      0
#define FL1_LABELS      1
#define FL2_LABELS      2
#define FL3_LABELS      3
#define FL4_LABELS      4
#define LOG_LABELS      5
#define SCI_LABELS      6

typedef char LABEL[LABEL_LENGTH+1];

typedef struct GR_ENVIRONMENT{
   int ResX,ResY;               /* pixel resolution of the current port    */
   int PlotXMax,PlotYMax;       /* maximum pixel position of plotting area */
   int BorderLeft,BorderRight; /* % of screen devoted to side borders      */
   int BorderBottom,BorderTop; /* % of screen devoted to b/t borders       */
   int MaxCount;                /* max # of increments allowed on an axis  */
   int XType;                   /* x axis type (linear, log, etc.)         */
   float XMin,XMax,XInc;        /* x axis parameters                       */
   int XCount;                  /* # of increments on x axis               */
   int XMinor;                  /* # of minor tics/increment on x axis     */
   int YType;                   /* y axis type (linear, log, etc.)         */
   float YMin,YMax,YInc;        /* y axis parameters                       */
   int YCount;                  /* # of increments on y axis               */
   int YMinor;                  /* # of minor tics/increment on y axis     */
} GR_ENVIRONMENT;

#include "plotdecs.h"
/*global*/  int InitPlot(void);
/*global*/  int SetBorderArea(int ,int ,int ,int );
/*global*/  int ScalePlot(float ,float ,float ,float );
/*global*/  int SetAxis(int ,int );
/*global*/  int ScaledX(float );
/*global*/  int ScaledY(float );
/*global*/  int SetScale(float ,float ,int ,float *,float *,float *);
/*global*/  int SetCount(float ,float ,int ,int ,float *,float *,float *);
/*global*/  int SetMinor(int ,int );
/*global*/  int XTic(float ,float ,int ,int );
/*global*/  int YTic(float ,float ,int ,int );
/*global*/  int XTicMinor(int ,float ,float ,int ,int );
/*global*/  int YTicMinor(int ,float ,float ,int ,int );
/*global*/  int DrawAxis(int ,int ,int ,int ,int ,int );
/*global*/  int LabelAxis(int ,char (*)[11],int );
/*global*/  int SetLabels(int ,char (*)[11],int ,int ,float ,int ,float );
/*global*/  int TitleAxis(int ,char *,int );
/*global*/  int Bar(float ,float ,float ,int ,int );
/*global*/  int ClipScreen(void);
/*global*/  int ClipPlot(void);
/*global*/  int EraseScreen(void);
/*global*/  int ErasePlot(void);
/*global*/  int PushEnviron(void);
/*global*/  int PopEnviron(void);
/*global*/  int CopyEnviron(struct GR_ENVIRONMENT *);
/*global*/  int SetEnviron(struct GR_ENVIRONMENT *);
/*global*/  int SetMaxCount(int );
```

```
/*global*/  float GetXMin(void);
/*global*/  int SetXMin(float );
/*global*/  float GetXMax(void);
/*global*/  int SetXMax(float );
/*global*/  float GetYMin(void);
/*global*/  int SetYMin(float );
/*global*/  float GetYMax(void);
/*global*/  int SetYMax(float );
/*global*/  int GetXCount(void);
/*global*/  int SetXCount(int );
/*global*/  int GetYCount(void);
/*global*/  int SetYCount(int );
/*global*/  float GetXInc(void);
/*global*/  int SetXInc(float );
/*global*/  float GetYInc(void);
/*global*/  int SetYInc(float );
/*global*/  int GetXType(void);
/*global*/  int GetYType(void);
/*global*/  int cdecl matherr(struct exception *);
/*global*/  int Text(char *);
/*global*/  int SpecialText(char *);
/*global*/  int Swap(int *,int *);
/*global*/  int SetHiResColor(int );
/*global*/  struct MENUBAR *MakeNewMenuBar(void);
/*global*/  int InitMenuBar(struct MENUBAR *);
/*global*/  int HomeMenuBar(struct MENUBAR *);
/*global*/  int FreeMenuBar(struct MENUBAR *);
/*global*/  int SetMenuStyle(int );
/*global*/  int SetMenuSpacing(struct MENUBAR *,char *,char *);
/*global*/  int AddMenuField(struct MENUBAR *,char *,int (*)(),
            struct MENUBAR *,char *);
/*global*/  int EditMenuField(struct MENUBAR *,int ,char *,
            int (*)(),struct MENUBAR *,char *);
/*global*/  int SetSpecialKey(struct MENUBAR *,char ,int (*)());
/*global*/  int SelectMenu(struct MENUBAR *,int );
/*global*/  int ActivateMenu(struct MENUBAR *,int );
/*global*/  int DeactivateMenu(struct MENUBAR *,int );
/*global*/  int RefreshMenuBar(struct MENUBAR *);
/*global*/  int DrawMenuBar(struct MENUBAR *);
/*global*/  int EraseMenuBar(struct MENUBAR *);
/*global*/  int BlankMenuBar(struct MENUBAR *);
/*global*/  int DrawMainBar(struct MENUBAR *);
/*global*/  int DrawDescBar(struct MENUBAR *);
/*global*/  int InvertSelectedMenu(struct MENUBAR *);
/*global*/  int GrayMenu(struct MENUBAR *,int );
/*global*/  int GetMenuChoice(struct MENUBAR *);
/*global*/  int PushMenu(struct MENUBAR *);
/*global*/  struct MENUBAR *PopMenu(void);
/*global*/  int MenuService(struct MENUBAR *);
/*global*/  int PrintMenuRecord(struct MENUBAR *,struct _iobuf *);
/*global*/  int GetKey(void);
/*global*/  int DumpKeyBuffer(void);
#define MAX_CHARS_PER_PAD      4
#define MAX_CHARS_PER_DIVIDER  6
```

```
#define MAX_MENUS_PER_BAR       8
#define MAX_CHARS_PER_BAR       80
#define MAX_CHARS_PER_MENU      15
#define MAX_CHARS_PER_DESCRIP   80
#define MENUDEPTH               5
#define GONE_STYLE              0
#define GRAY_STYLE              1
#define GRAYPATTERN             3

typedef struct MENUBAR{
   int MenuChanged;
         /* flag denoting whether menu needs screen refreshing...          */
   int BarX,BarY;
         /* global screen coordinates of this menu bar                     */
   int BarWid;
         /* width in global pixels of this bar                             */
   int DeactiveStyle;
         /* style in which deactive menus are displayed                    */
   char SpecialKey;
         /* 'hidden' key for user-defined special cases                    */
   int (*KeyFunction)();
         /* function executed by SpecialKey                                */
   int NumberOfMenus;
         /* number of menus currently defined in this bar                  */
   int Selection;
         /* ID of the menu that is currently hilited in this bar           */
   char Padding[MAX_CHARS_PER_PAD+1];
         /* padding string placed before and after each menu               */
   char Divider[MAX_CHARS_PER_DIVIDER+1];
         /* string separating each menu name                               */
   char MenuName[MAX_MENUS_PER_BAR][MAX_CHARS_PER_MENU];
         /* array containing the names of all menus in this bar            */
   int (*MenuFunction[MAX_MENUS_PER_BAR])();
         /* array of pointers to menu service functions                    */
   struct MENUBAR *NextMenuBar[MAX_MENUS_PER_BAR];
         /* array of pointers to any submenus (NULL if menu at bottom)     */
   char MenuDescrip[MAX_MENUS_PER_BAR][MAX_CHARS_PER_DESCRIP];
         /* Array of string descriptors for each menu in bar               */
   int MenuIsActive[MAX_MENUS_PER_BAR];
         /* flags describing active status of each menu in bar             */
} MENUBAR;

typedef struct MENUSTACK{
int StackTop;                    /* menu stack pointer                     */
MENUBAR *MenuBarPtr[MENUDEPTH];  /* max depth of menu structure            */
} MENUSTACK;
#define FORMFEED            12
#define RETURN              13
#define ESC                 27
#define SPACE               32
#define HOME                -71
#define UPARROW             -72
#define PAGEUP              -73
#define LEFTARROW           -75
```

```
#define RIGHTARROW          -77
#define END                 -79
#define DNARROW             -80
#define DOWNARROW           -80
#define PAGEDOWN            -81
#define INS                 -82
#define DEL                 -83
#define TRUE                1
#define FALSE               0
#define OK                  0
#define NOT_OK              1
#define SINGLE_Y            0
#define NOCHOICE            0
#define VULNTYPE            1
#define SENSTYPE            2
#define DISTTYPE            3
#define MAX_GRAPH_TYPE      3
#define MAX_PATHS_RETURNED  10
#define MAX_RFTS            10
#define PATTERN1            1
#define PATTERN2            25
#define LOBLUE              1
#define LOGREEN             2
#define LOCYAN              3
#define LORED               4
#define HIBLUE              9
#define HIGREEN             10
#define HICYAN              11
#define HIRED               12
#define PRINT_SCREEN_INTERRUPT 5

#define MaxOf(a,b) (((a) > (b)) ? (a) : (b))

/*global*/  void DashArc(void);
/*global*/  void MultiDash(void);
/*global*/  void MultiFrame(void);
/*global*/  void Invect_Multi(void);
/*global*/  void Pnvect_EGA(void);
/*global*/  void StrokeText(void);
/*global*/  void SetEGA(void);
/*global*/  void ResetEGA(void);
/*global*/  void SetEGAMap(void);
/*global*/  void Invect_EGA(void);
/*global*/  void CopyEGABits(void);
/*global*/  void Invect_ColorBits(void);
/*global*/  void CopyPlanes(void);
/*global*/  int graph_service(int *,int ,int *);
/*global*/  int TypeService(void);
/*global*/  int DrawCurrentType(void);
/*global*/  int DrawVulnType(void);
/*global*/  int DrawSensType(void);
/*global*/  int DrawDistType(void);
/*global*/  int BuildGraph(void);
/*global*/  int BuildVuln(void);
```

```
/*global*/  int BuildSens(void);
/*global*/  int BuildDist(void);
/*global*/  int ResponseService(void);
/*global*/  int ScrollUpService(void);
/*global*/  int ScrollDownService(void);
/*global*/  int ZoomHomeService(void);
/*global*/  int ZoomHome(void);
/*global*/  int ZoomOutService(void);
/*global*/  int ZoomInService(void);
/*global*/  int ZoomQuitService(void);
/*global*/  int ScaleService(void);
/*global*/  int PrintService(void);
/*global*/  int PrintLogoService(void);
/*global*/  int QuitService(void);
/*global*/  int Legend(void);
/*global*/  int SetLog(struct GR_ENVIRONMENT *,struct GR_ENVIRONMENT *,
            float ,float ,float ,int );
/*global*/  int SetLin(struct GR_ENVIRONMENT *,struct GR_ENVIRONMENT *);
/*global*/  int LabelTwo(int ,int ,char (*)[11],char (*)[11],
            struct GR_ENVIRONMENT *,struct GR_ENVIRONMENT *);
/*global*/  int PushTwo(struct GR_ENVIRONMENT *,struct GR_ENVIRONMENT *);
/*global*/  int PopTwo(struct GR_ENVIRONMENT *,struct GR_ENVIRONMENT *);
/*global*/  int SetXStuff(float ,float ,float ,int );
/*global*/  int SetYStuff(float ,float ,float ,int );
/*global*/  int BuildMenuStructure(void);
/*global*/  int HomeMenuBars(void);
/*global*/  int DrawGraph(void);
/*global*/  int RedrawGraphAndTitle(void);
/*global*/  int RedrawGraph(void);
/*global*/  int ErasePage(void);
/*global*/  int DrawBar(float ,float ,float ,int ,int );
/*global*/  int PlotTitle(char *);
/*global*/  int Message(char *,int );
/*global*/  int zoom(void);
/*global*/  int ZoomBound(int *,int ,int ,int );
/*global*/  int ArrowUp(float ,float );
/*global*/  int ArrowDown(float ,float );
/*global*/  int PrintScreen(int );
/*global*/  int init_meta(void);
/*global*/  int PlotSaviLogo(int ,int ,float ,float ,int ,int );
/* int GetSaviPoly(int *,struct UNNAMED *,struct UNNAMED *,struct UNNAMED *); */
/*global*/  int SetRFT(void);
/* int DisplayCurrentRFT(struct UNNAMED *,int ); */
/*global*/  int bell(void);
/*global*/  int inputs(void);
/*global*/  int findpath(void);
/*global*/  int menu(char );
/*global*/  int save_asd(void);
/*global*/  int load_asd(void);
/*global*/  char *get_threat_label(int );
/*global*/  char *get_ASD_label(void);
/*global*/  int get_current_threat(void);
/*global*/  int get_current_direction(void);
/*global*/  int get_current_strategy(void);
```

```
/*global*/  int main(void);
/*global*/  int display_turbo_progress(int ,float );
/*global*/  int printer_ready(int );
/* wfd_defs.h -- definitions for Windows for Data

    ************* Copyright 1985 by Vermont Creative Software **************

      DATE:  January 11, 1986

*/

#ifndef ATT_LOGIC
#define ATT_LOGIC                       /*must use logical attributes        */
#endif

/*-------------------------------------------------------------------------*/
/*  Definitions for accesing standard form and window                      */
/*-------------------------------------------------------------------------*/
#define STDFORMP dformp                 /*standard form pointer              */
#define STDWNP dformp->wnp              /*standard form window pointer       */

/*-------------------------------------------------------------------------*/
/*  Definitions for accessing current field and form                       */
/*-------------------------------------------------------------------------*/
#define CURFLDP _cfldndp->fldp
#define CURFORMP _curfrmp

/*-------------------------------------------------------------------------*/
/*  Definitions for controlling date format                                */
/*-------------------------------------------------------------------------*/
#define US 0                            /*US format for date       - mm/dd/yy */
#define EURO 1                          /*EUROPEAN format for date - dd/mm/yy */

/*-------------------------------------------------------------------------*/
/*  Definitions for field types                                            */
/*-------------------------------------------------------------------------*/
#define F_INT      0                    /*integer field type                 */
#define F_STRING   1                    /*string field type                  */
#define F_CHAR     2                    /*character field type               */
#define F_BOOL     3                    /*boolean field type                 */
#define F_LONG     4                    /*long integer field type            */
#define F_RDECIMAL 5                    /*right justified decimal field type */
#define F_LDECIMAL 6                    /*left justified decimal field type  */
#define F_DATE     7                    /*date field                         */
#define F_TIME     8                    /*time field                         */
#define F_MENU     9                    /*menu field type                    */
#define F_FLOAT   10                    /*floating point field type          */
#define F_11      11                    /*RESERVED for future use            */
#define F_12      12                    /*     ditto                         */
#define F_13      13                    /*     ditto                         */
#define F_14      14                    /*     ditto                         */
#define F_15      15                    /*     ditto                         */
#define F_16      16                    /*     ditto                         */
#define F_17      17                    /*     ditto                         */
```

```
#define F_18      18                    /*     ditto                          */
#define F_19      19                    /*     ditto                          */

/*-------------------------------------------------------------------------*/
/*  Definitions for accessing picture symbol table                         */
/*-------------------------------------------------------------------------*/
#define ANYASCII  0                     /*any ascii character                */
#define ANYALPHA  1                     /*any alpha character                */
#define DIGIT     2         /*digits 0-9, -, +                 */
#define NUMBER    3                     /*digits 0-9, -, +, ., E             */
#define ASCIITOUP 4                     /*any ascii to uppercase             */
#define SOFT_DEC  5                     /*soft decimal point for left decimal */

/*-------------------------------------------------------------------------*/
/*  Field terminator definitions                                           */
/*-------------------------------------------------------------------------*/
#define NEXT_FLD   1                    /*next field                         */
#define PREV_FLD   2                    /*previous field                     */
#define FIRST_FLD  3                    /*first field                        */
#define LAST_FLD   4                    /*last field                         */
#define SAME_FLD   5                    /*same field                         */
#define ABORT_FORM 6                    /*abort form                         */
#define EXIT_FORM  7                    /*exit form                          */

/*-------------------------------------------------------------------------*/
/*  Definitions for accessing data conversion functions                    */
/*-------------------------------------------------------------------------*/
#define FROM_STRING 0                   /*convert from a string to variable  */
#define TO_STRING   1                   /*convert variable to a string       */

/*-------------------------------------------------------------------------*/
/*  Definitions for accessing field bit flag                               */
/*-------------------------------------------------------------------------*/
#define RE_FLD     1                    /*field is right-to-left entry       */
#define NODISP_FLD 2                    /*no display of initial value        */
#define PURE_FLD   4                    /*field is pure - ins/del/bs allowed */
#define REQ_FLD    8                    /*entry is require for field         */
#define MFILL_FLD 16                    /*must fill field                    */
#define CLISTKEY  32                    /*clear field on first printable     */
                                        /*keystroke                          */
#define PROT_FLD  64                    /*protected field                    */

/*-------------------------------------------------------------------------*/
/*  Definitions for controlling form read environment                      */
/*-------------------------------------------------------------------------*/
#define ALL_FLDS 0                      /*read all fields at once            */
#define ONE_FLD  1                      /*read one field at a time           */

/*-------------------------------------------------------------------------*/
/*  Define some macros for convienence                                     */
/*-------------------------------------------------------------------------
#define se_autoexit(x)        (_autoxit = (x))
#define se_datef(x)           (_datefmt = (x))
#define se_automv(x)          (_automv = (x))
```

```
#define se_bell(x)              (_ring = (x))
#define se_verexit(x)           (_verxit = (x))
#define sf_location(r, c, p)    ((p)->fld_rb = (r), (p)->fld_cb = (c))
#define sf_name(n, p)           ((p)->fld_name = (n))
#define sf_atts(i, a, p)        ((p)->iact_att = (i), (p)->act_att = (a))
#define sf_message(m, p)        ((p)->message = (m))
#define sf_clear(p)             ((p)->fld_flag |= CLISTKEY)
#define sf_rtjust(p)            ((p)->fld_flag |= RE_FLD,\
                                 (p)->fld_flag |= CLISTKEY)
#define sf_required(p)          ((p)->fld_flag |= REQ_FLD)
#define sf_mustfill(p)          ((p)->fld_flag |= MFILL_FLD)
#define sf_protected(p)         ((p)->fld_flag |= PROT_FLD)
#define sf_noecho(p)            ((p)->act_att = LNODISPLAY,\
                                 (p)->iact_att = LNODISPLAY)
#define sf_nodisplay(p)         ((p)->fld_flag |= NODISP_FLD)
#define sf_help(h, p)           ((p)->help = (h))
#define sf_valid(f, p)          ((p)->val_func = (f))
#define sf_userp(u, p)          ((p)->fld_usrp = (u))

#define sd_helpfunc(p)          (_hlpfunc = (p))    /*install user help func    */
#define sd_hmsg(p)              (_dhlpmsg = (p))    /*set default help message  */
/* wfd_glob.h -- global variables for Windows for Data

    ************* Copyright 1985 by Vermont Creative Software **************

      DATE:  January 16, 1986

      COMMENTS

           Initial values are set for global variables.

           Place in main program following "wfd.h"

      NOTE: Modified by Brad Key for use in SAVI software.
            New Picture Control Characters were added and
            the Integer Field Picture Validation Function
            was removed.

*/

/*-------------------------------------------------------------------------*/
/*  Include the Windows for C global variables                             */
/*-------------------------------------------------------------------------*/
#include <window.h>

/*-------------------------------------------------------------------------*/
/*  Global variables for Windows for Data                                  */
/*-------------------------------------------------------------------------*/
WINDOW msg_wn;                          /*message window                   */
WINDOW mnu_msgwn;                       /*message window for menus         */
WINDOW err_wn;                          /*error window                     */
WINDOW help_wn;                         /*help window                      */
DFORMPTR dformp;                        /*pointer to default form          */
DFORMPTR _curfrmp;                      /*pointer to current form          */
```

```
FLDNODEPTR _cfldndp;                    /*pointer to current field node in    */
                                        /*linked list of fields               */
PICNODEPTR pichdndp;                    /*pointer to header node in linked    */
                                        /*list of picture strings             */
WINDOW _fld_wn;                         /*field window                        */

PFI _hlpfunc = NULLFP;                  /*user supplied help function         */
char *_dhlpmsg = "No help available for this field";    /*default help message*/

/*-------------------------------------------------------------------------*/
/*  To enable floating point you must #define the symbol WFD_FLOAT before     */
/**/you #include wfd_glob.h.
/*                 */
/*  This section declares the following integer functions so that they may    */
/**/be used as initializers.
/*-------------------------------------------------------------------------*/
extern int atoboolx(), atocx(), atodecx(), atoix(), atolx(),
           atostx(), booltoax(), ctoax(), dectoax(), itoax(),
           kdabort(), kdbacksp(), kdclfld(), kdcleof(), kddel(), kdexit(),
           kdfirst(), kdhelp(), kdins(), kdlast(), kdleft(), kdnfld(),
           kdpfld(), kdright(), kdundo(), ltoax(), picalpha(),
           picascii(), picdigit(), picnum(), pictoup(), picone(), picfive(),
           picdigit_only(), picfile_name(), sttoax(),
           val_bool(), val_date(), val_dec(), val_fill(), val_int(),
           val_long(), val_req(), val_time(), vpbool(), vpchar(), vpdate(),
           vpdec(), vpint(), vplong(), vpstr(), vptime();

#ifdef WFD_FLOAT
extern int atofx(), ftoax(), vpfloat(), val_float();
#endif

PFI _dp_func[][2] =                     /*Array of Data Processing functions  */
{                                       /*Each field type has 2 functions     */
                                        /*     Convert from string            */
                                        /*     Convert to string              */
                                        /*We reserve positions 0 - 19.        */
                                        /*Positions 20-29 available to user.  */
                                        /*The order of this array must        */
                                        /*conform to the #defines for field   */
                                        /*types.                              */
    atoix,      itoax,                  /*INTEGER field                       */
    atostx,     sttoax,                 /*STRING field                        */
    atocx,      ctoax,                  /*CHARACTER field                     */
    atoboolx,   booltoax,               /*BOOLEAN field                       */
    atolx,      ltoax,                  /*LONG INTEGER field                  */
    atodecx,    dectoax,                /*RIGHT ENTRY DECIMAL field           */
    atodecx,    dectoax,                /*LEFT ENTRY DECIMAL field            */
    atostx,     sttoax,                 /*DATE field                          */
    atostx,     sttoax,                 /*TIME field                          */
    atostx,     sttoax,                 /*MENU field                          */
#ifdef WFD_FLOAT
    atofx,      ftoax                   /*FLOATING POINT field                */
#else
    NULLFP,     NULLFP
```

```
#endif
/*  NULLFP,     NULLFP,                 RESERVED for future use             */
/*  NULLFP,     NULLFP,                        ditto                        */
/*  NULLFP,     NULLFP,                        ditto                        */
/*  NULLFP,     NULLFP,                        ditto                        */
/*  NULLFP,     NULLFP,                        ditto                        */
/*  NULLFP,     NULLFP,                        ditto                        */
/*  NULLFP,     NULLFP,                        ditto                        */
/*  NULLFP,     NULLFP,                        ditto                        */
);

PFI _valfunc[] =                        /*Field Validation Functions        */
{                                       /*We reserve positions 0 - 19.      */
                                        /*Positions 20-29 available to user.*/
                                        /*The order of this array must      */
                                        /*conform to the #defines for field */
                                        /*types.                            */
    val_int,                            /*INTEGER field                     */
    NULLFP,                             /*STRING field                      */
    NULLFP,                             /*CHARACTER field                   */
    val_bool,                           /*BOOLEAN field                     */
    val_long,                           /*LONG INTEGER field                */
    val_dec,                            /*RIGHT ENTRY DECIMAL field         */
    val_dec,                            /*LEFT ENTRY DECIMAL field          */
    val_date,                           /*DATE field                        */
    val_time,                           /*TIME field                        */
    NULLFP,                             /*MENU field - no validation needed */
#ifdef WFD_FLOAT
    val_float                           /*FLOAT field                       */
#else
    NULLFP
#endif
/*  NULLFP,                             RESERVED for future use             */
/*  NULLFP,                                    ditto                        */
/*  NULLFP,                                    ditto                        */
/*  NULLFP,                                    ditto                        */
/*  NULLFP,                                    ditto                        */
/*  NULLFP,                                    ditto                        */
/*  NULLFP,                                    ditto                        */
/*  NULLFP,                                    ditto                        */
/*  NULLFP,                                    ditto                        */
);


PFI _pv_func[] =                        /*Picture Validation Functions      */
{                                       /*We reserve positions 0 - 19.      */
                                        /*Positions 20-29 available to user.*/
                                        /*The order of this array must      */
                                        /*conform to the #defines for field */
                                        /*types.                            */
    NULLFP,                             /*INTEGER field                     */
    vpstr,                              /*STRING field                      */
    vpchar,                             /*CHARACTER field                   */
```

```
    vpbool,                             /*BOOLEAN field                          */
    vplong,                             /*LONG INTEGER field                     */
    NULLFP,                             /*RIGHT ENTRY DECIMAL field              */
    NULLFP,                             /*LEFT ENTRY DECIMAL field               */
    vpdate,                             /*DATE field                             */
    vptime,                             /*TIME field                             */
    NULLFP,                             /*MENU field -- no picture val needed */
#ifdef WFD_FLOAT
    vpfloat                             /*FLOAT field                            */
#else
    NULLFP
#endif
/*  NULLFP,                             RESERVED for future use                */
/*  NULLFP,                                    ditto                           */
/*  NULLFP,                                    ditto                           */
/*  NULLFP,                                    ditto                           */
/*  NULLFP,                                    ditto                           */
/*  NULLFP,                                    ditto                           */
/*  NULLFP,                                    ditto                           */
/*  NULLFP,                                    ditto                           */
/*  NULLFP,                                    ditto                           */
};

DKEYDEF _dkeytbl[] =                    /*Editor Key Definition Table          */
{                                       /*The order here is unimportant.       *
    {0, kdhelp},                        /*invoke help key                      *,
    {0, kdundo},                        /*restore field to original value      */
    {0, kdclfld},                       /*clear field key                      */
    {0, kdcleof},                       /*clear to end of field                */
    {K_ENTER, kdexit},                  /*exit data form key                   */
    {K_ESC, kdabort},                   /*abort data form key                  */
    {-K_DN, kdnfld},                    /*move to next field key          */
    {-K_UP, kdpfld},                    /*move to previous field key           */
    {-K_PGUP, kdfirst},                 /*move to first field in form          */
    {-K_PGDN, kdlast},                  /*move to last field in form           */
    {-K_LEFT, kdleft},                  /*left arrow key                       */
    {-K_RIGHT, kdright},                /*right arrow key                      */
    {-K_INS, kdins},                    /*insert key                           */
    {-K_DEL, kddel},                    /*delete key                           */
    {K_BACK, kdbacksp}                  /*backspace key                        */
};
int _qdKeys = sizeof(_dkeytbl) / sizeof(DKEYDEF);  /*number of keys defined   */

DPICDEF _dpictbl[] =                    /*Data Picture Symbol Definition Table*/
{                                       /*The order of this array must         */
                                        /*conform to the #defines for picture */
                  /*symbol types.                       */
    {'X', picascii},                    /*any ascii character                  */
    {'A', picalpha},                    /*any alpha character                  */
    {'-', picdigit},                    /*any digit 0-9, +, -                  */
    {'#', picnum},                      /*any number 0-9, +, -, ., E
    {'!', pictoup},                     /*convert any ascii to uppercase       */
    {'@', picnum},                      /*soft decimal point for left          */
                                        /*justified decimal field              */
```

```
    {'F', picfile_name},                /*any valid file name character       */
    {'1', picone},                      /*any digit 0-1                       */
    {'5', picfive},                     /*any digit 0-5                       */
    {'9', picdigit_only}                /*any digit 0-9                       */
};
int _qdpic = sizeof(_dpictbl) / sizeof(DPICDEF); /*number of pic symbols def  */

int fld_term;                           /*field terminator code               */
int _ring = ON;                         /*global variable to control bell     */
int _clform = ON;                       /*controls clearing of field          */
                                        /*definitions on exit from data form  */
int _clfld = ON;                        /*controls clearing of field on first */
                                        /*printable character in first        */
                                        /*position of field                   */
int _automv = ON;                       /*controls auto-move to next field on */
                                        /*full field                          */
int _autoxit = ON;                      /*controls auto-exit from form when   */
                                        /*last field is filled                */
int _verxit = ON;                       /*controls if form exit verification  */
                                        /*is invoked                          */
int _datefmt = US;                      /*controls date format - US mm/dd/yy  */
                                        /*                       EURO dd/mm/yy*/
int _insmode = OFF;                     /*insert mode status                  */
char _wfd_buf[CO_QMAX + 1];             /*private global working buffer       */
/* att_glob.h --  contains default definitions and declarations of logical
                  attribute tables for use in Windows for C.  Nests in
                  window.h.  The #defines for logical attributes are in
                  def_att.h, which is nested in bios.h
```

********** Copyright 1985 by Vermont Creative Software **********

IMPLEMENTING LOGICAL OR PHYSICAL ATTRIBUTES

Logical attributes will be implemented if ATT_LOGIC is defined. This is the default condition set in bios.h.

To use physical attributes, #undef ATT_LOGIC here or in an include file that precedes this one. Alternatively, the #define ATT_LOGIC statement in bios.h could be commented out. Generally, the include file for a specific application would set ATT_LOGIC.

THE LOGICAL-ATTRIBUTE TABLE

The physical attributes associated with each logical attribute are specified in a logical-attribute table (two dimensional array), datt_tbl[][]. There is a row for each logical attribute; the number of rows in the table equals LATTQ. The number of columns equals PATTQ, the number of physical attributes for each logical attribute. LATTQ and PATTQ are #defined in def_att.h

The system, as supplied has two physical attributes for each logical attribute: a monochrome attribute, and a color attribute; thus PATTQ is set to two in att_def.h. You can add more physical attributes by increasing the value of PATTQ and adding more columns to the logical attribute table.

```
*/

#ifdef ATT_LOGIC                        /*LOGICAL ATTRIBUTES in use          */
int _lattsw = 1;
char latt[LATTQ];                       /*array of logical attributes        */

char  datt_tbl [LATTQ] [PATTQ] =
{
#ifdef MSDOS
   {NORMAL, c_att(WHITE, BLACK)},                                    /*LDOS     */
   {NORMAL, c_att(GREEN, BLACK)},                                    /*LNORMAL  */
   {REVERSE, c_att(BLACK, WHITE)},                                   /*LREVERSE */
   {REVERSE, c_att(WHITE + LIGHT, BLUE)},                            /*LHIGHLITE*/
   {NORMAL + HIGH_INT + BLINK, c_att(RED + BLINK, BLACK)},           /*LURGENT  */
   {NORMAL + HIGH_INT, c_att(WHITE, BLUE)},                          /*LHELP    */
   {REVERSE, c_att(RED, BLACK)},                                     /*LERROR   */
   {REVERSE, c_att(BLUE, CYAN)},                                     /*MESSAGE  */
   {NORMAL, c_att(BLACK, BLUE)},                                     /*LFIELDI  */
   {REVERSE, c_att(CYAN + LIGHT, BLUE)},                             /*LFIELDA  */
   {REVERSE, c_att(BLUE, WHITE)},                                    /*LMARK    */
   {0, c_att(BLACK, BLACK)},                                         /*LNODISPLA*/
   {NORMAL + HIGH_INT, c_att(CYAN + LIGHT, BLACK)},                  /*LCYANHI  */
   {NORMAL, c_att(BLUE, BLACK)},                                     /*LBLUE    */
   {NORMAL,  c_att(GREEN, BLACK)},                                   /*LGREEN   *
   {NORMAL, c_att(CYAN, BLACK)},                                     /*LCYAN    *,
   {REVERSE, c_att(RED, BLACK)},                                     /*LRED     */
   {REVERSE, c_att(WHITE + LIGHT, BLACK)},                           /*LWHITEHI */
   {NORMAL + HIGH_INT, c_att(GREEN + LIGHT, BLACK)},                 /*LGREENHI */
   {NORMAL, c_att(WHITE, BLACK)},                                    /*LWHITE   */
   {NORMAL, c_att(WHITE, BLUE)},                                     /*LWHITBLUE*/
   {REVERSE, c_att(RED + LIGHT, BLACK)},                             /*LREDHI   */
   {NORMAL + HIGH_INT + BLINK, c_att(WHITE+LIGHT+BLINK, BLACK)},     /*LWHITBLNK*/
   {0, 0}, {0, 0},                                                   /*RESERVED */
   {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0},                           /*RESERVED */
   {0, 0}, {0, 0}                                                    /*RESERVED */
#endif

#ifdef UNIX
   {NORMAL, COLOR9},                                                 /*LDOS     */
   {NORMAL, COLOR10},                                                /*LNORMAL  */
   {REVERSE, COLOR11},                                               /*LREVERSE */
   {HIGH_INT, COLOR12},                                              /*LHIGHLITE*/
   {BLINK, COLOR13},                                                 /*LURGENT  */
   {HIGH_INT, COLOR11},                                              /*LHELP    */
   {REVERSE, COLOR14},                                               /*LERROR   */
   {HIGH_INT, COLOR12},                                              /*MESSAGE  */
   {HIGH_INT, COLOR15},                                              /*LFIELDI  */
   {REVERSE, COLOR16},                                               /*LFIELDA  */
   {REVERSE, COLOR11},                                               /*LMARK    */
   {INVISIBLE, INVISIBLE},                                           /*LNODISPLA
   {REVERSE, COLOR1},                                                /*LBLACK   */
   {REVERSE, COLOR2},                                                /*LBLUE    */
   {NORMAL,  COLOR3},                                                /*LGREEN   */
```

```
    (NORMAL, COLOR4),                                                   /*LCYAN     */
    (REVERSE, COLOR5),                                                  /*LRED      */
    (REVERSE, COLOR6),                                                  /*LMAGENTA  */
    (NORMAL, COLOR7),                                                   /*LBROWN    */
    (NORMAL, COLOR8),                                                   /*LWHITE    */
    (0, 0), (0, 0), (0, 0), (0, 0), (0, 0),                             /*RESERVED  */
    (0, 0), (0, 0), (0, 0), (0, 0), (0, 0),                             /*RESERVED  */
    (0, 0), (0, 0)                                                      /*RESERVED  */
#endif
);
int _attrowq = LATTQ;                       /*number of rows in attribute table   */

#else                                       /*PHYSICAL ATTRIBUTES in use          */

int _lattsw = 0;
char latt[1];                               /*array of logical attributes         */
char datt_tbl [1][PATTQ];
int _attrowq = 1;

#endif

int _attcolq = PATTQ;                       /*number of columns in attribute table*/
```

DISTRIBUTION:

1 Tom Cousins
DP-341.3
U.S. Department of Energy
19900 Germantown Rd.
Germantown, MD 20874

1 Rokaya Al-Ayat
NSS/Safeguards Program Manager
Lawrence Livermore National Laboratory
7000 East Ave.
Mail Stop L-195
Livermore, CA 94550

1 Science & Engineering Associates, Inc.
Attn: Richard DeFuria
6100 Uptown Blvd., N.E.
Albuquerque, NM 87110

1	2614	A. A. Elsbernd
5	3141	S. A. Landenberger
8	3141-1	C. L. Ward
3	3151	W. I. Klein
1	5210	C. C. Hartwigsen
1	5211	E. R. Hoover
1	5211	W. K. Paulus
1	5211	M. K. Snell
1	5211	A. E. Winblad
1	5212	J. C. Matter
5	5212	S. E. Jordan
1	5240	D. S. Miyoshi
1	8524	J. A. Wackerly

DO NOT MICROFILM
THIS PAGE