SOUTHWEST RESEARCH INSTITUTE
Post Office Drawer 28510, 6220 Culebra Road
San Antonio, Texas 78228-0510

CONTINUATION OF RESEARCH INTO SOFTWARE FOR SPACE OPERATIONS SUPPORT

FINAL REPORT
VOLUME II

NASA Grant No. NAG 9-388
SwRI Project No. 05-2984

Prepared by:

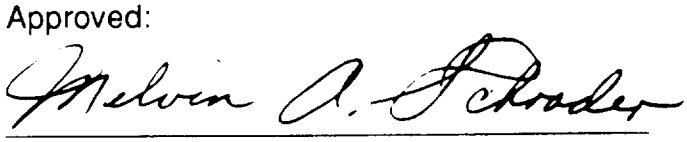
Mark D. Collier
Ronnie Killough
Nancy L. Martin

Prepared for:

NASA
Johnson Space Center
Houston, Texas

November 30, 1990

Approved:



Melvin A. Schrader, Director
Data Systems Department

SOUTHWEST RESEARCH INSTITUTE
Post Office Drawer 28510, 6220 Culebra Road
San Antonio, Texas 78228-0510

CONTINUATION OF RESEARCH IN SOFTWARE FOR SPACE OPERATIONS SUPPORT

CONVERSION OF THE DISPLAY MANAGER TO X WINDOWS/MOTIF

NASA Grant No. NAG 9-388
SwRI Project No. 05-2984

Prepared by:
Mark D. Collier
Nancy L. Martin
Ronnie Killough

Prepared for:
NASA
Johnson Space Center
Houston TX 77058

November 30, 1990

Table of Contents

1.0 INTRODUCTION

NASA is currently using a set of applications called the Display Builder and Display Manager. These applications allow users to build and drive real-time data displays. The two applications currently run on Concurrent systems and are heavily dependent on the Graphic Kernel System (GKS). When these applications were designed, GKS was the best choice for a graphics base due to portability concerns. At this time however, these two applications would more appropriately be developed in X Windows. This is true for the following reasons:

- GKS is primarily designed for drawing 2-dimensional primitives such as lines, polygons, and stroke text (GKS is not a bitmapped graphics systems). GKS also supports transformations, in which the user's coordinate system is transformed to the device coordinate system. This simplifies operations such as scaling and viewing. Unfortunately, GKS carries a large amount of overhead for support of transformations and in drawing of simple primitives. The result is that simple operations such as drawing of text are inefficient.
- GKS does not include any high-level functions which simplify user interface development. Developing a user interface in GKS is difficult because GKS only supports basic primitives such as lines, polygons, and text. Development of user interface objects such as fields, push buttons, and scrolling lists must be done from scratch.
- GKS, especially when integrated with X Windows, is not totally portable. There is no standard for the function calls which are required to interact with the surrounding X Windows system. In this case, GKS applications are not truly portable.

At the current time, the most appropriate graphics base for applications such as the Display Builder and Display Manager is X Windows, in which a low level X is used for all actual text and graphics display and a standard widget set (such as Motif) is used for the user interface. Use of X Windows will increase performance, improve the user interface, enhance portability, and improve reliability.

To demonstrate the viability of using X Windows and Motif for this type of application, SwRI developed an X Windows/Motif-based prototype of the existing Display Manager application. Note that only the Display Manager was prototyped, rather than both the Display Builder and Manager. This is true for the following reasons:

- The Display Manager represents a critical operational requirement and therefore demands maximum performance and reliability. Performance is less of a concern for the Display Builder, which is only used during development.
- While the Display Manager is large (about 100,000 lines of code), it is a manageable application for this research effort. The Display Builder is much larger and complex and would be too difficult to convert in the available time.

2.0 RESEARCH GOALS

Prototype an X Window/Motif-based Display Manager which provides the following advantages over a GKS-based application:

- Improved performance - by using low-level X Windows, display of graphic and text will be more efficient (faster and less resource-intensive).
- Improved user interface - by using Motif, the prototype will be easier to use.

- Improved portability - the prototype will execute on both Concurrent and Sun workstations.
- Improved reliability - use of only X Windows/Motif (as opposed to X and GKS) will improve reliability.

3.0 RESEARCH DETAILS

The goal of this research effort was to take the existing Display Manager application and convert all user interface and graphic display functionality to X Windows and Motif. The goal was not to redesign the Display Manager nor to add any new major functions. SwRI did not make any major changes to the Display Manager data structures, to shared memory, or in the area of available functionality. With the exception of one major change (explained below), the updates were focused upon the user interface and the display of data.

The original Display Manager application consisted of three separate processes. These processes and their responsibilities are:

- Data Handler - Retrieves data from the data acquisition process and makes the data available for display. Only one copy of the Data Handler executes on each workstation.
- Display Manager - Provides the user interface for a control of a single display. A separate copy of the Display Manager is required for each display.
- Data Displayer - Performs the actual data display. A separate copy of the Data Displayer is required for each display.

The prototype merged the functionality of the Display Manager and Data Displayer processes into one process. This approach was selected due to the large size of each program once the Motif library is loaded. The prototype uses one Data Handler process for a workstation and one Display Manager process for each display (with the Data Displayer functionality built into the Display Manager). SwRI began making changes that would allow one Display Manager process to support multiple displays. Much of the code to support this functionality exists at this point.

One problem with which SwRI was faced was that it was impossible to recreate the actual environment present at NASA. Although SwRI was supplied with a Concurrent system, it was not possible to configure all the necessary hardware and software needed to run the LAN support software, WEX, and data acquisition. This was especially true on the Sun, which as indicated was a target system. This prevented SwRI from creating a valid flow of real-time data, which is necessary for testing of the prototype. Therefore, it was decided that a stubbed version of the Data Handler would be developed. SwRI developed stubbed versions of the ***ds_connect***, ***ds_disconnect***, and ***ds_getkeys***, and ***ds_getparm*** functions which are used by the Data Handler to retrieve data from the data acquisition process. The stubbed versions supplied all the simulated data needed to test the converted functions.

The majority of the functions within the original Display Manager/Data Displayer processes which did not use GKS were not modified. The functions which did use GKS were converted to X and/or Motif one by one. All functionality within the original Display Manager process was converted (all the user interface). The majority of the functions within the original Data Displayer process were converted. The major functions which were not converted are those dealing with foreground graphics (clocks, meters, etc.). In addition, the zooming feature which is available in the prototype only affects plots. No scaling of normal background or foreground text is available.

The flow of control in the prototype is similar to the original. The major modifications which were made include the following:

- The user interface has been changed significantly. The prototype uses a single "control panel" window which provides access to all functions. When a display is created, a new independent window with horizontal and vertical scrollbars will appear. All forms, help text, and messages appear as popup windows. All functions which involve enabling or disabling a feature are controlled through toggle menu items which change to indicate the current state of the feature.
- The flow of events and control was changed to support X and Motif. In the new flow, the following events occur:
  - Menu item/selection of a function key - this type of event causes a callback which in turn calls the ***command*** function, which in turn calls the appropriate function.
  - Timer events - a cyclic timer event is used to trigger update of the display at a regular interval.
  - Expose events - an expose event occurs if the display window is scrolled or otherwise exposed.
  - Input events - an input event (selection of a button) occurs when a PBI is selected.
- The help text for each function is now stored in individual files. This eliminates the need for a separate function for each help screen.
- Because the two applications were merged, a number of redundant functions were eliminated.
- The original Display Manager used a number of functions for presentation of temporary menus. Because all menus are in the control panel window, these functions were eliminated.
- Because the user interface and display functions are now in one process, there is no need to communicate via flags through shared memory.
- Several new functions were added to the user interface:
  - A color editor - allows the colors used by the prototype to be interactively edited.
  - Enable/Disable messages - all messages generate an advisory. This feature controls whether or not messages are also displayed in the form of a popup window.
  - Pause display - this function allows the display to be paused and later restarted.
- For fields which allow entry of any of a display-specific set of values (such as MSIDs), a list of the valid values appears.
- The function which displayed the user-defined function keys is now built into the main control panel.
- Because so much code was changed, the original comment headers were removed. This was necessary because the time required to update the pseudocode was too great.
- Limited comment headers and in-line comments were added to all new and modified source files.

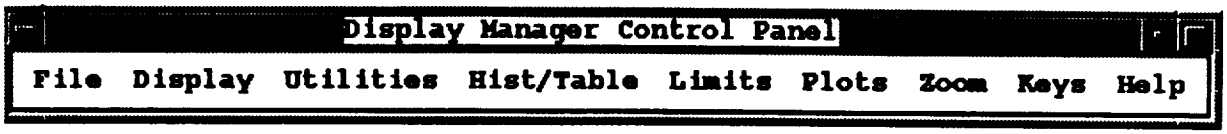
The prototype executes properly on both Concurrent and Sun systems. Most of the development was performed on the Sun, with specific performance tuning occurring on the Concurrent.

The following sections provide illustrations of the different windows presented by the prototype. Note that many of the original functions are built into the control panel menus in the form of

toggle menus which do not require a separate window. These menus are not presented in this document because no utility was available to capture a transient menu.

3.1 Main Control Window

This is the main control panel window for the Display Manager prototype. All functions may be accessed from this window.



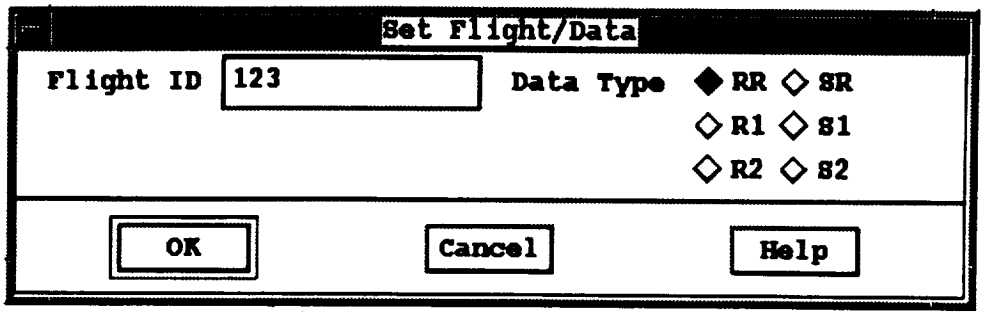
The pulldown menus and functions which are available from this window include the following (the functions followed by "..." involve display of an additional window):

- File:
  - Enable Messages
  - Set Flight/Data...
  - Screen Dump
  - Edit Colors...
  - Exit
- Display:
  - Select Display...
  - Remove Display
  - Freeze Display
- Utilities:
  - Change Update Rate...
  - Unlatch DDD MSID...
  - Unlatch ALL DDD's
  - Change GDR...
  - Enable Alarms
  - Enable PBIs
  - Enable Logging
  - Enable All Logging
- Hist/Table:
  - History Tables...
- Limits:
  - List Limits...
  - Change Limits...
- Plots:
  - List Plots...
  - Display Overlay...

  - Save Overlay...
  - Define Universal Plot...
- Zoom:
  - Zoom
  - Reset Zoom
  - Change Zoom Factor...
- Keys (List of available user-defined function keys)
- Help (specific help is also available from popup windows):
  - Enable/Disable Messages
  - Set Flight ID/Datatype
  - Screen Dump
  - Edit Colors
  - Exit
  - Select Display
  - Remove Display
  - Freeze Display
  - Change Update Rate
  - Unlatch DDD MSID
  - Unlatch All DDD's
  - Change GDR
  - History Tables
  - Enable/Disable Alarms
  - Enable/Disable PBI's
  - Enable/Disable Logging
  - Enable/Disable All Logging
  - List Limits
  - Change Limits
  - List Plots
  - Display Overlay
  - Save Overlay
  - Define Universal Plot
  - Zoom
  - Reset Zoom Factor
  - Set Zoom Factor
  - Show PF Keys

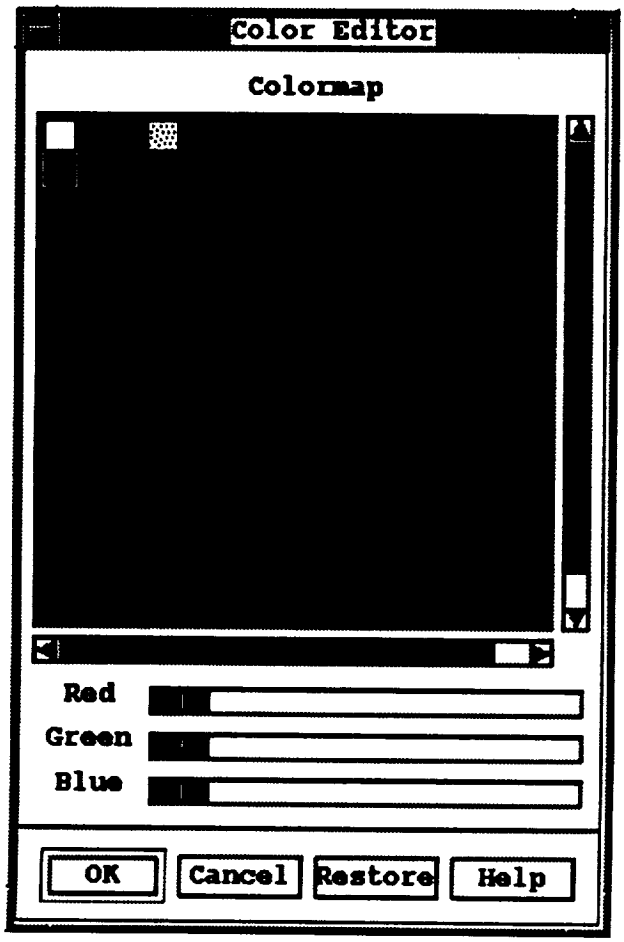
3.2 Set Flight/Data Type Window

This window allows the user to select the current flight and data type.



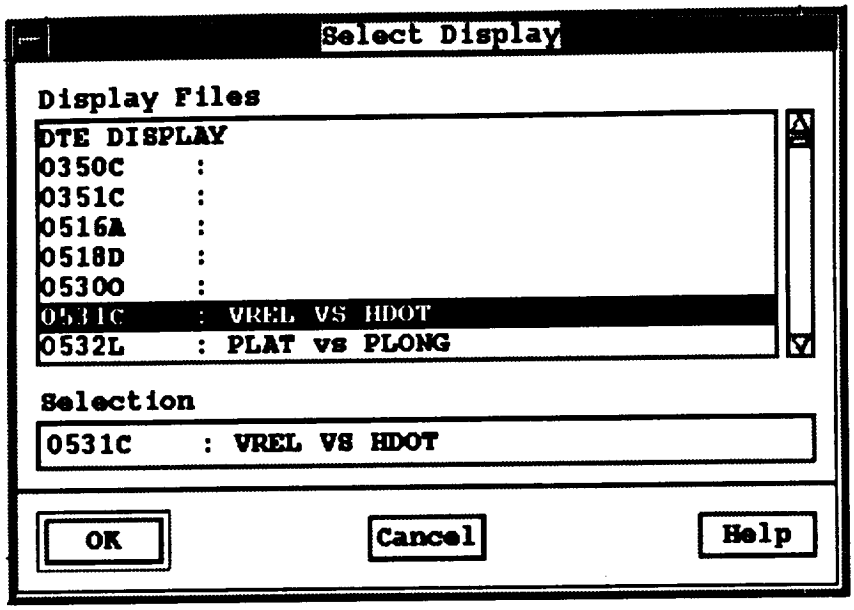
3.3 Color Editor Window

This window allows the user to edit the colors used in the display.



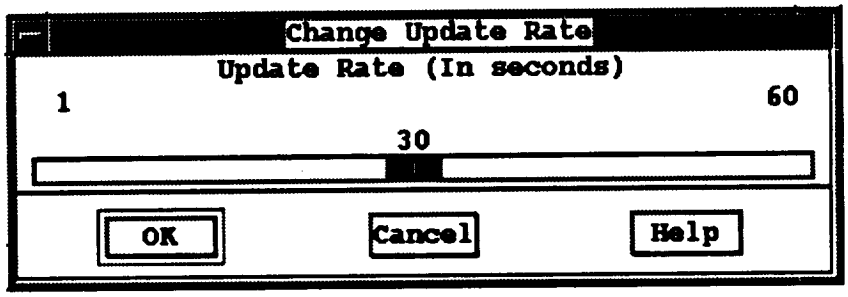
3.4 Select Display Window

This window is used to select a new display.



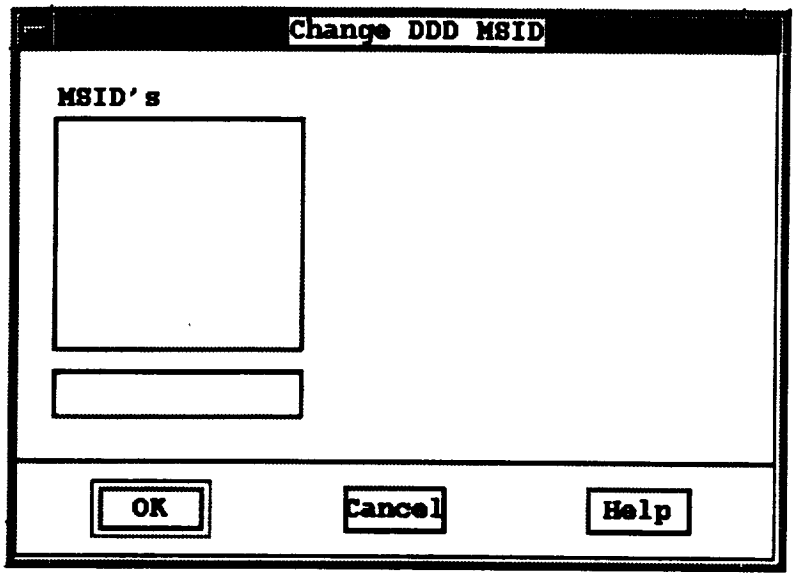
3.5 Change Update Rate Window

This window is used to change the update rate.



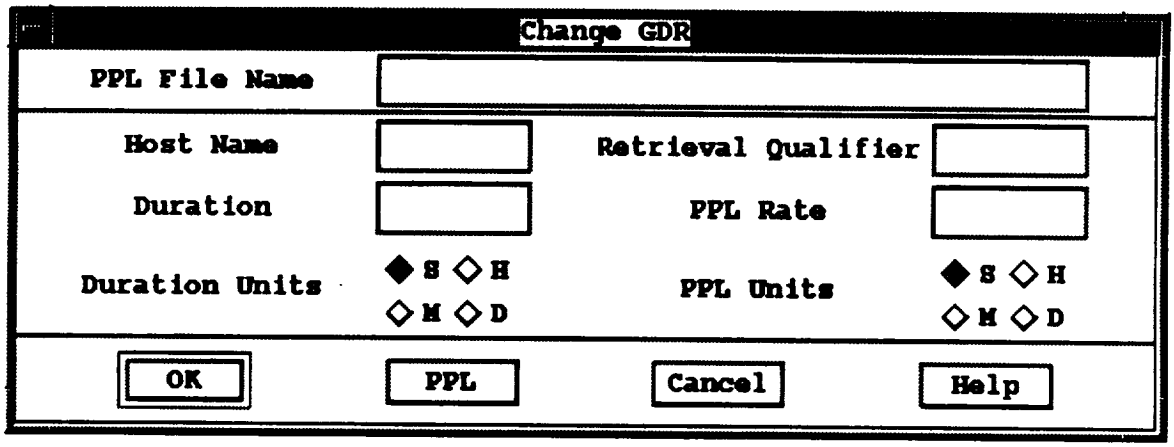
3.6 Change DDD MSID Window

This window is used to enable or disable a specific DDD MSID.



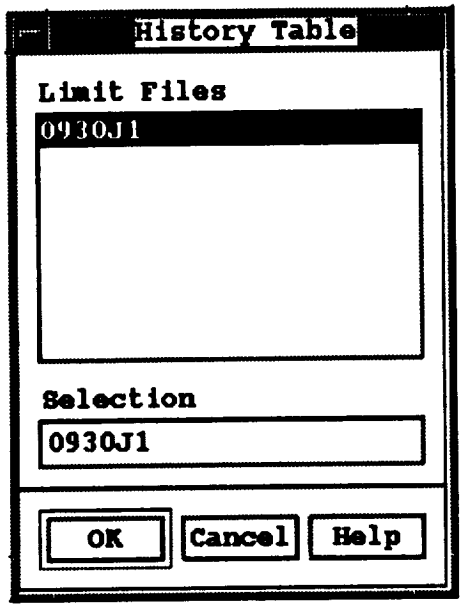
3.7 Change GDR Window

This window is used to change GDR data source characteristics.



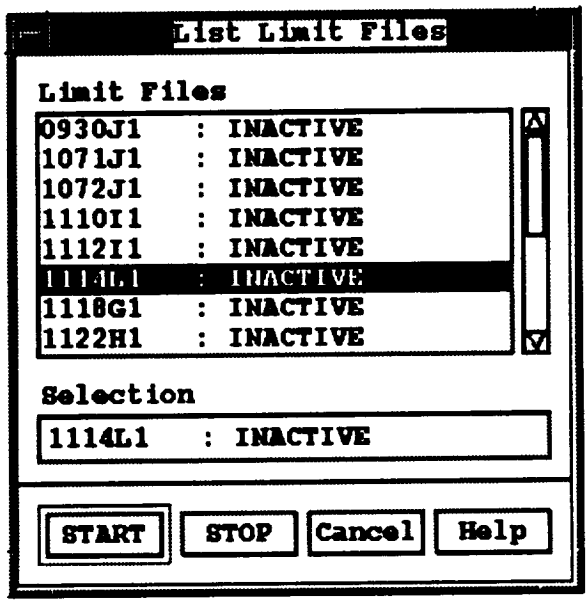
3.8 History Tables Window

This window allows display of History Table information.



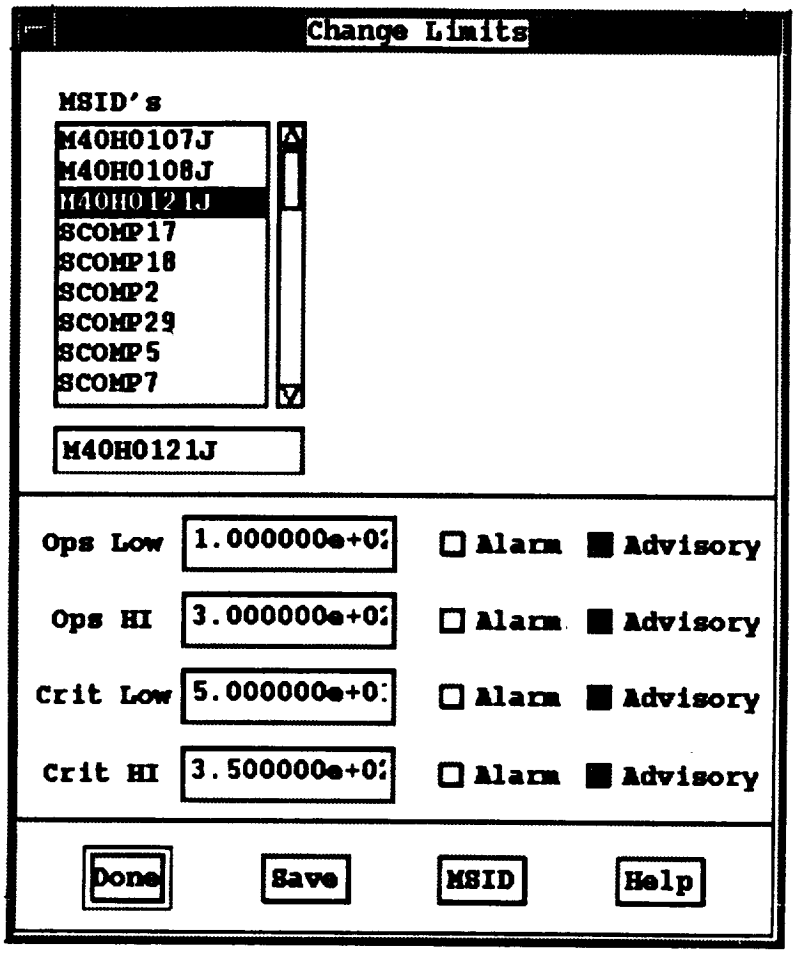
3.9 List Limit Files Window

This window allows the user to enable or disable a limit group for the display.



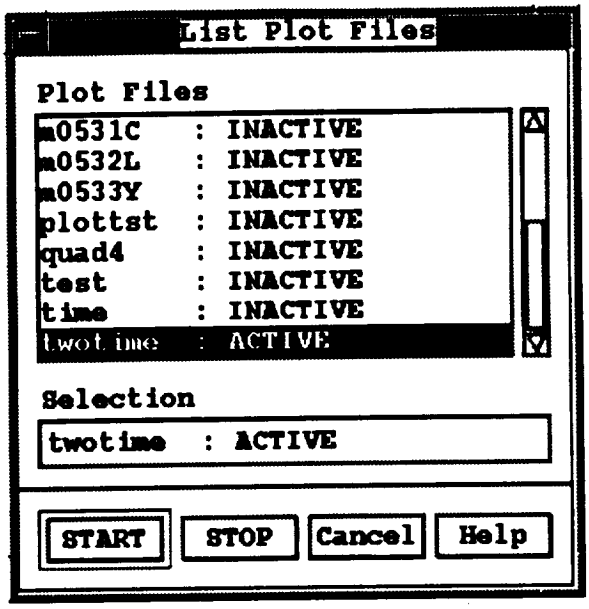
3.10 Change Limits Window

This window is used to alter characteristics of active limits.



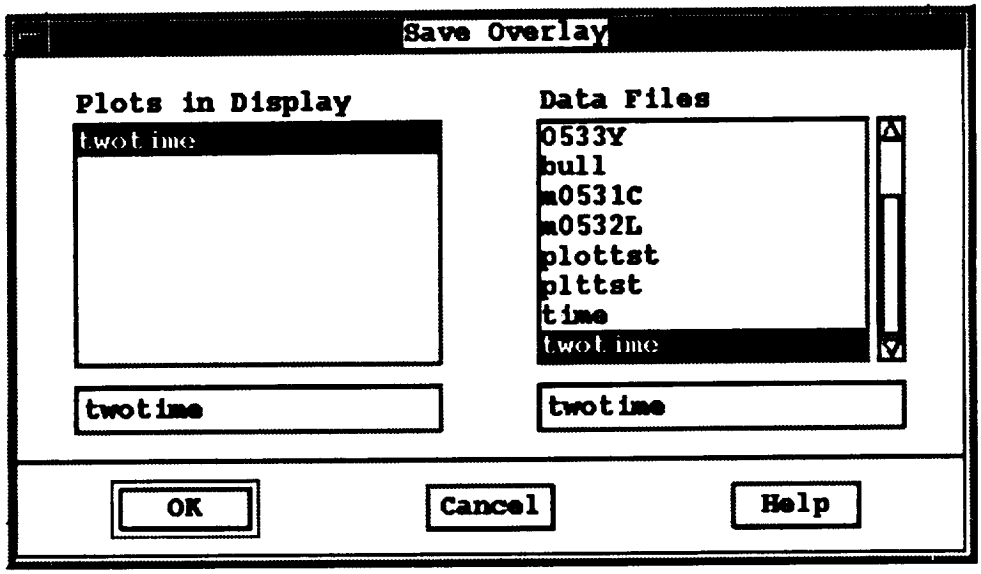
3.11 List Plot Files Window

This window allows the user to enable or disable any plot present in the display.



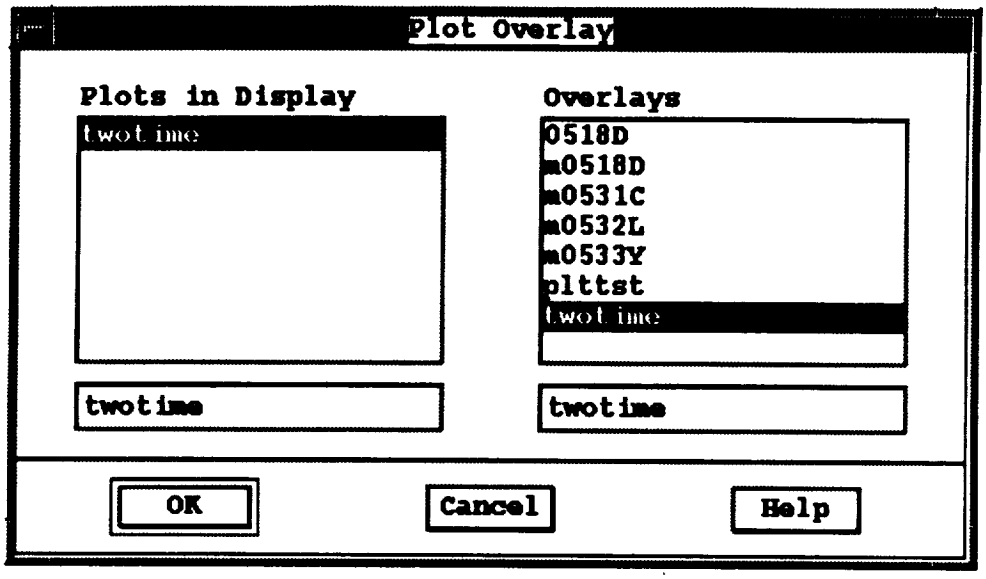
3.12 Save Overlay Window

This window allows the user to save a plot data file as an overlay.



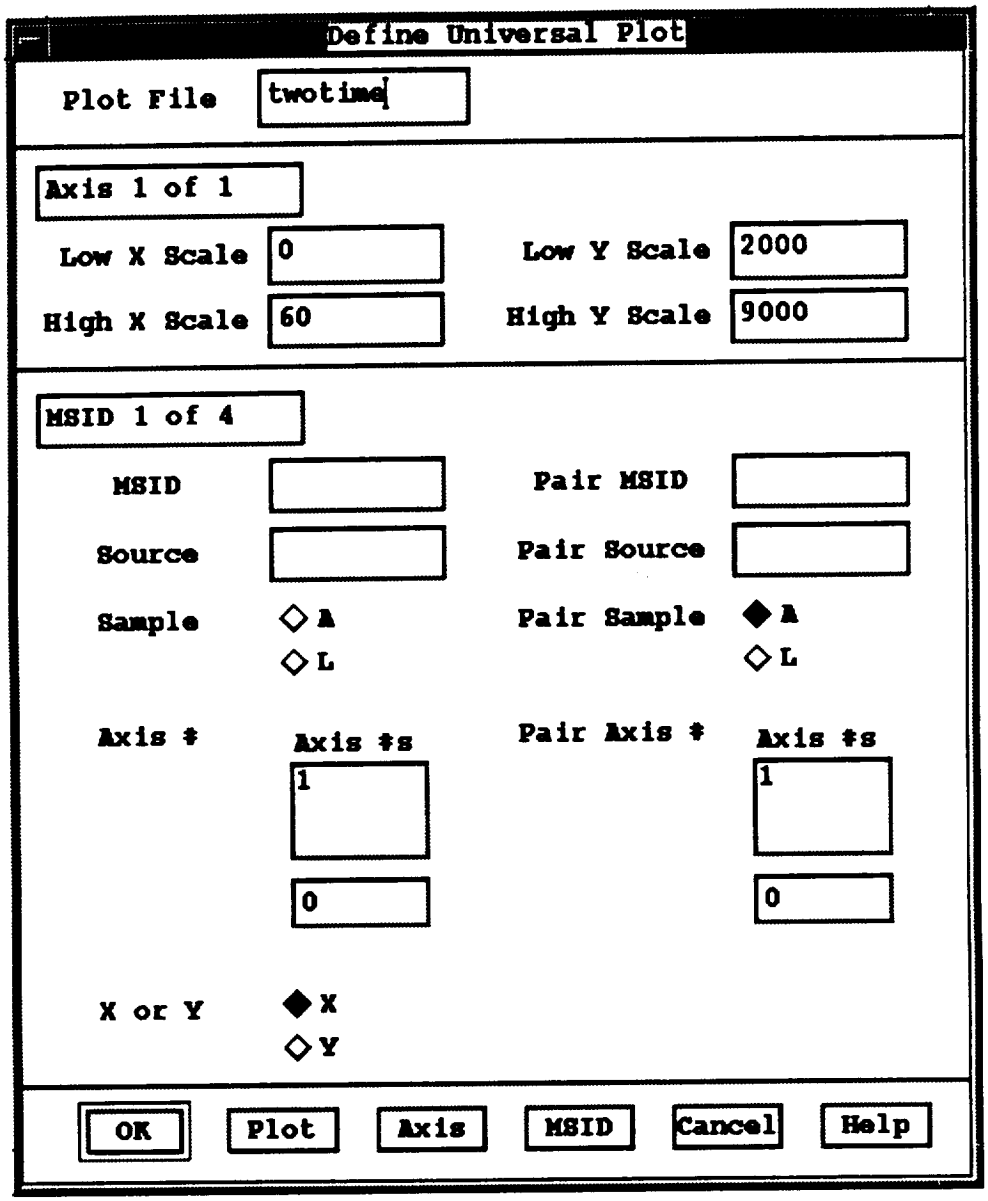
3.13 Plot Overlay Window

This window allows the user to display an overlay which was previously saved with the Save Overlay function.



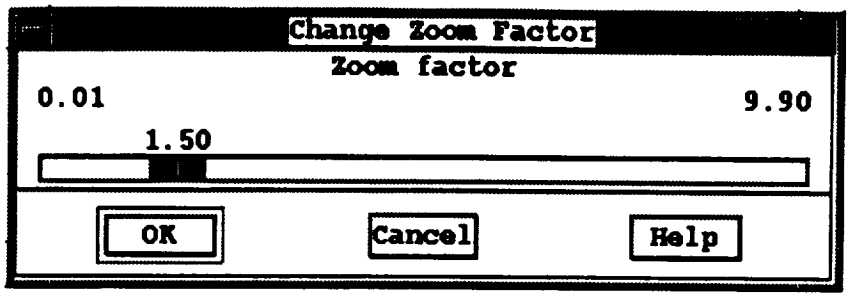
3.14 Define Universal Plot Window

This window allows the user to update plot values for a particular universal or normal plot file.



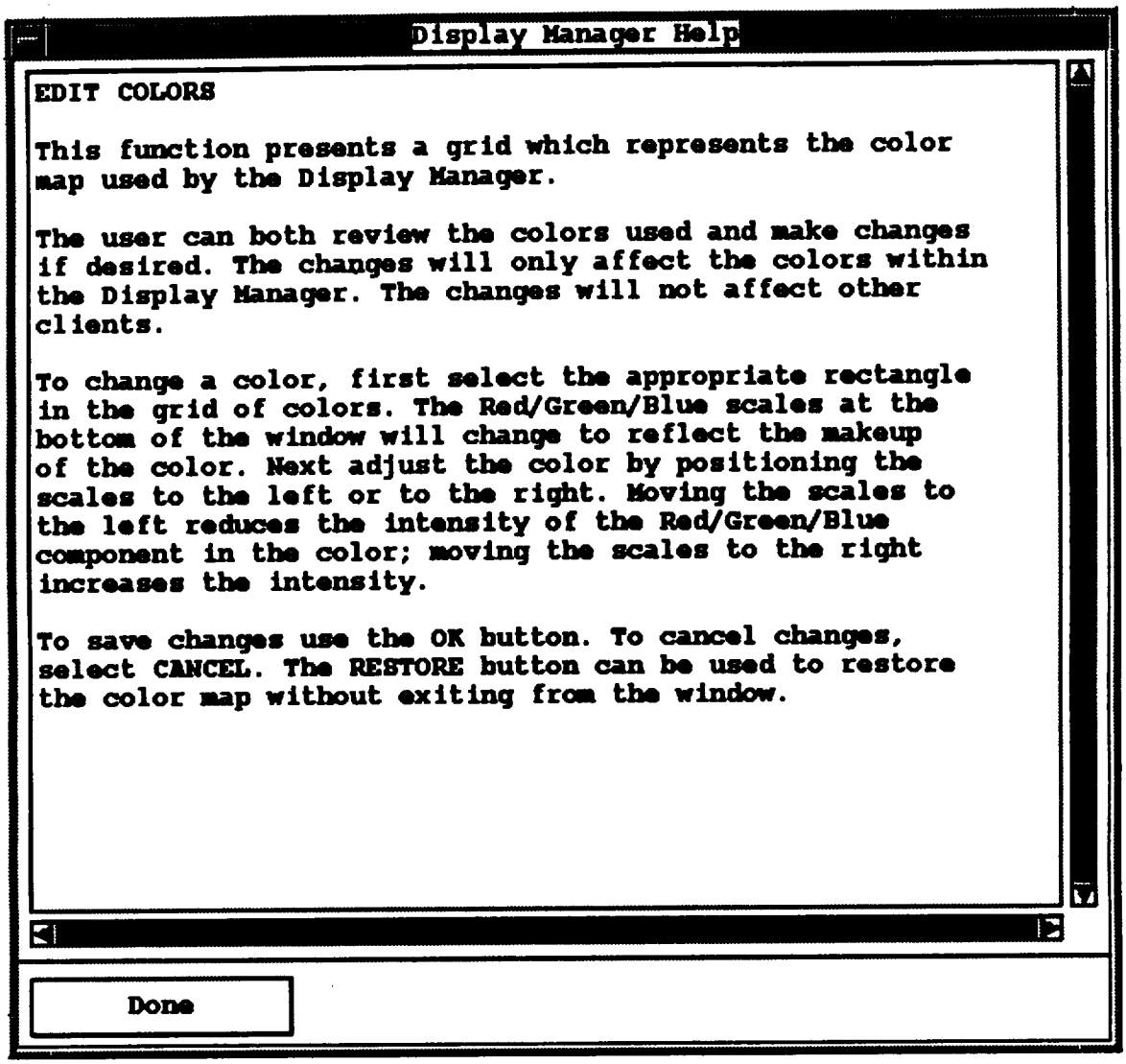
3.15 Change Zoom Factor Window

This window is used to change the zoom factor.

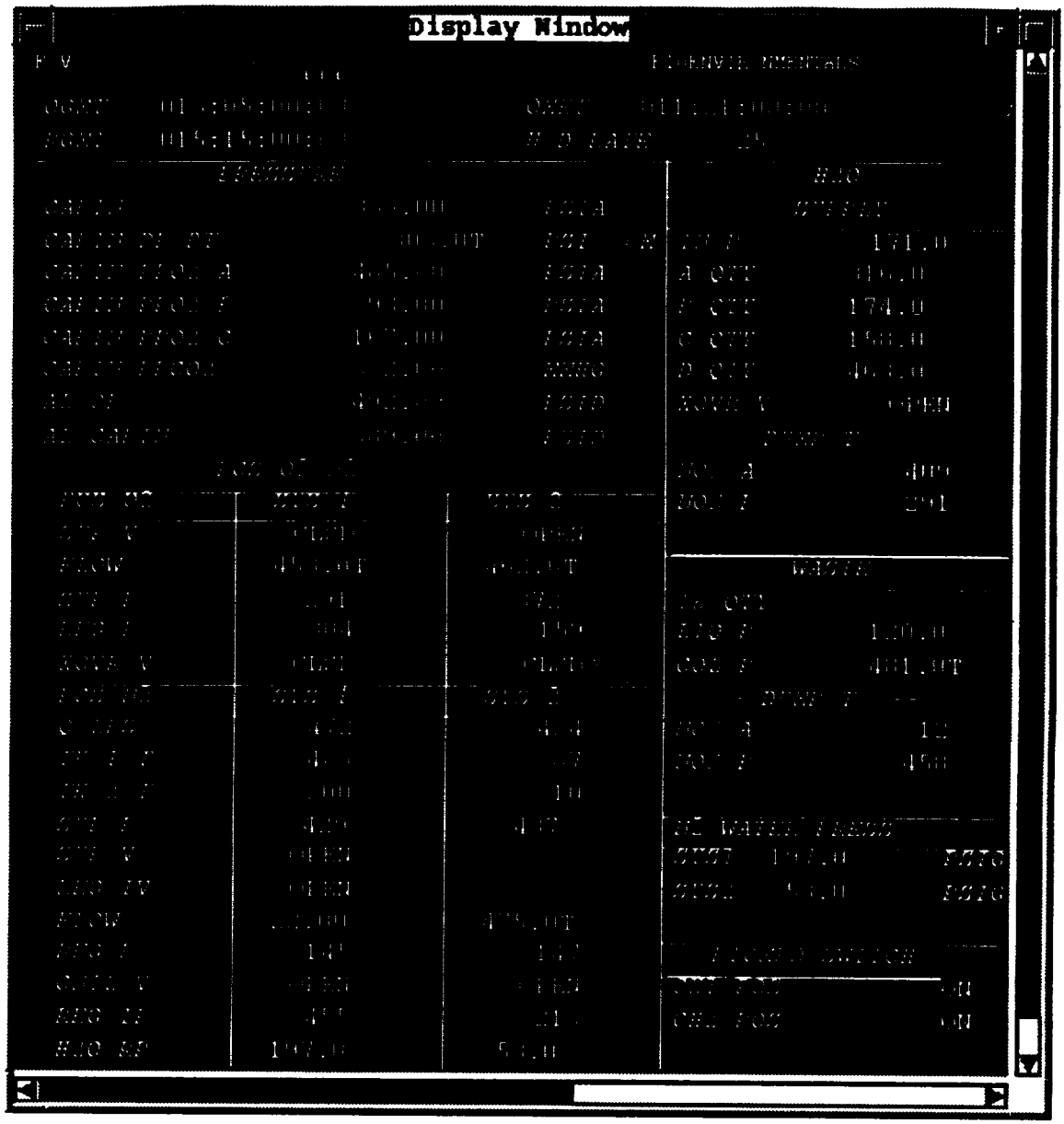


3.16 Help Window

This window shows an example of a help display. This window is independent of other windows and will remain displayed until removed.

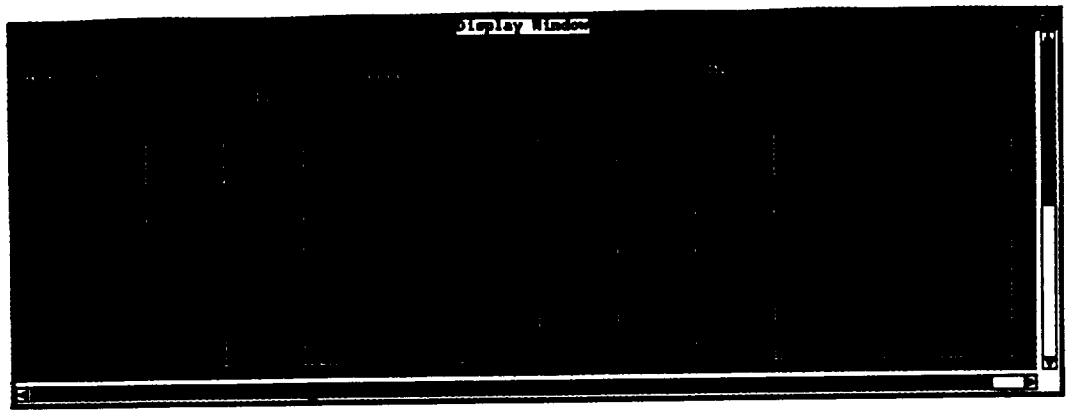


3.17 Sample Display 1

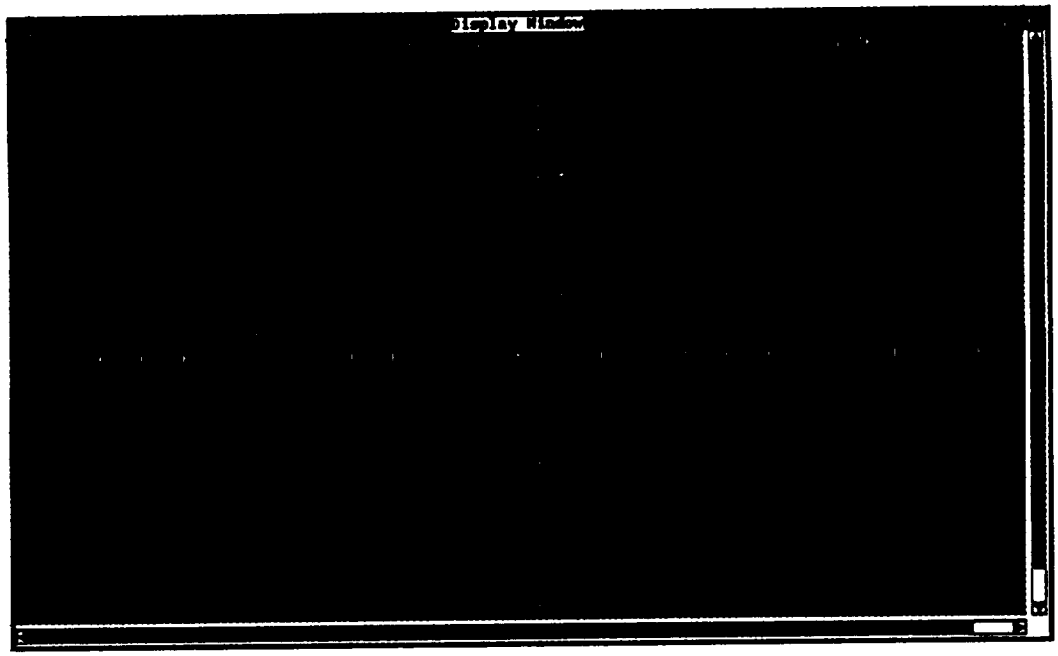


ORIGINAL PAGE IS
OF POOR QUALITY

3.18 Sample Display 2



3.19 Sample Display 3



ORIGINAL PAGE IS
OF POOR QUALITY

4.0 RESEARCH CONCLUSIONS

Several important conclusions were drawn from this research effort. These conclusions include the following:

- Use of X Windows/Motif for an application such as the Display Manager can result in an application which offers high performance, an improved user interface, high reliability, and increased portability.
- Additional effort is required to convert remaining functionality. Additional effort is also required for more performance tuning, elimination of unused data structures (a result of merging processes), and generation of internal documentation.
- A relatively small amount of effort is required to implement several new features, such as support of multiple displays with one Display Manager process.

5.0 ATTACHMENTS

The following pages contain the actual code for the converted Display Manager. The code and related files which are present include:

- User interface library Makefile and code.
- Data Handler stub code.
- Data Manager code.
- Utility programs.

ATTACHMENT 1 - User Interface Library

```
###############################################################################
# Makefile for the Display Manager user interface library.
###############################################################################

#
# Define the target which this file is to create.
#

TARGET      = libtui.a

#
# Initialize master, binary, library, and include directories.
#

MASTER  = /home/project/2984/db
BINDIR  = $(MASTER)
INCDIR = ../include
INCDIRS = -I. -I$(INCDIR)
LIBDIR  = .

#
# Define the compiler and linker flags.
#

CFLAGS      = -g $(INCDIRS) $(FLAGS)
LDFLAGS     = -g

#
# Define all objects which make up this target.
#

OBJS        =\
    tui_cr_as.o\
    tui_cr_cas.o\
    tui_cr_form.o\
    tui_cr_label.o\
    tui_cr_pb.o\
    tui_cr_rb.o\
    tui_cr_sel.o\
    tui_cr_sep.o\
    tui_cr_scale.o\
    tui_cr_text.o\
    tui_cr_tog.o\
    tui_cr_ts.o\
    tui_list.o\
    tui_msg.o\
    tui_prompt.o\
    tui_ques.o\
    tui_wait.o

#
# Define all header files required.
#

HDRS        =\
    $(INCDIR)/user_inter.h

#
# Make the target.
#

all:        $(TARGET)

$(TARGET):  $(OBJS)
```

```
        ar rv $(LIBDIR)/$@ $(OBJS)
        ranlib $(LIBDIR)/$@

$(OBJS):    $(HDRS)
```

```
###############################################################################
# Makefile for the Display Manager user interface library.
###############################################################################

#
# Define the target which this file is to create.
#

TARGET      = libtui.so.1.1

#
# Initialize master, binary, library, and include directories.
#

MASTER  = /home/project/2984/db
BINDIR  = $(MASTER)
INCDIR = ../include
INCDIRS = -I. -I$(INCDIR)
LIBDIR  = .

#
# Define the compiler and linker flags.
#

CFLAGS      = -g -pic $(INCDIRS) $(FLAGS)
LDFLAGS     = -g

#
# Define all objects which make up this target.
#

OBJS        =\
    tui_cr_as.o\
    tui_cr_cas.o\
    tui_cr_form.o\
    tui_cr_label.o\
    tui_cr_pb.o\
    tui_cr_rb.o\
    tui_cr_sel.o\
    tui_cr_sep.o\
    tui_cr_scale.o\
    tui_cr_text.o\
    tui_cr_tog.o\
    tui_cr_ts.o\
    tui_list.o\
    tui_msg.o\
    tui_prompt.o\
    tui_wait.o\
    tui_ques.o

#
# Define all header files required.
#

HDRS        =\
    $(INCDIR)/user_inter.h

#
# Make the target.
#

all:        $(TARGET)

$(TARGET):  $(OBJS)
```

```
	ld -o $(LIBDIR)/$@ -assert pure-text $(OBJS)

$(OBJS):	$(HDRS)
```

```
/*****************************************************************************
 * MODULE NAME: ( tui_create_app_shell )
 *
 *  This function creates an application shell widget which is used to create
 *  an indepedent window. Windows rooted with application shells may be iconi-
 *  fied independently from other windows and do not have their input auto-
 *  matically disabled when a popup is displayed.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <X11/Shell.h>
#include <Xm/Xm.h>
#include <Xm/mwm.h>


extern Widget       Top;        /* The top level widget which is the parent
                                 * of transient shells.
                                 */


Widget tui_create_app_shell ( name, colormap, arg_list, num_args )

    char            *name;      /* The name of the widget. This string will
                                 * appear in the title bar of the shell.
                                 */

    Colormap        colormap;   /* The colormap to associate with the appli-
                                 * cation shell.
                                 */

    Arg             arg_list[]; /* List of arguments to be used in this
                                 * instance of the widget.
                                 */

    int             num_args;   /* The number of arguments in the argument
                                 * list;
                                 */

{
    register int    i,
                    count = 0;

    Arg             args[10];

    Widget          widget;

/*
 *  Set up at most five arguments from arg_list.
 */

    if ( num_args > 5 )
        num_args = 5;
    for ( i = 0; i <= num_args-1; i++ ) {
        args[i] = arg_list[i];
```

```
        count++;
    }

/*
 *  Initialize the colormap argument and create the application shell widget.
 *  The colormap argument forces all child widgets to use the new colormap in-
 *  stead of the default root colormap.
 */

    XtSetArg ( args[count], XmNcolormap, colormap ); count++;
    widget = XtAppCreateShell ( name, "Display Manager",
                                applicationShellWidgetClass, XtDisplay ( Top ),
                                args, count );

/*
 *  Specify the functions which will be available in the Motif window manager
 *  menu area. Note that the CLOSE function is absent. Note also that this
 *  is done after creation, because this argument appears to be ignored at creation
 *  time.
 */

    count = 0;
    XtSetArg ( args[count], XmNmwmFunctions,
        MWM_FUNC_MOVE     | MWM_FUNC_RESIZE |
        MWM_FUNC_MINIMIZE | MWM_FUNC_MAXIMIZE ); count++;
    XtSetValues ( widget, args, count );

/*
 *  Return the created application shell widget.
 */

    return ( widget );
}
```

```
/*****************************************************************************
 * MODULE NAME: tui_create_cascade
 *
 *  This function creates a MOTIF cascade button gadget. A sub menu ID is passed in
 *  and the cascade gadget is attached to it.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <X11/Intrinsic.h>
#include <Xm/CascadeB.h>


Widget tui_create_cascade ( parent, name, submenu, arg_list, num_args )

    Widget          parent,     /* The parent widget which this cascade widget
                                 * will be attached.
                                 */

                    submenu;    /* The menu which will be activated when this
                                 * function is invoked.
                                 */

    char            *name;      /* The string which will be displayed on the menu.
                                 */

    Arg             arg_list[]; /* A list of arguments for expandability.
                                 */

    int             num_args;   /* Number of valid arguments in the arg_list.
                                 * Currently there are no valid arguments.
                                 */

{
    register int    count = 0,
                    i;

    Arg             args[10];

    Widget          widget;

/*
 *  Set up at most five arguments from the argument list.
 */

    if ( num_args > 5 )
        num_args = 5;
    for ( i = 0; i <= num_args-1; i++ ) {
       args[i] = arg_list[i];
       count++;
    }

/*
 *  Set argument which associates it with the pulldown menu which is to be activated
 *  when the cascade is selected.
 */
```

```
    XtSetArg ( args[count], XmNsubMenuId, submenu ); count++;

/*
 *  Create the cascade widget and return the widget pointer to calling function.
 */

    XtManageChild ( widget = XmCreateCascadeButton ( parent, name, args, count ) );

    return ( widget );
}
```

```
/*****************************************************************************
 * MODULE NAME: tui_create_form
 *
 *  This function creates a MOTIF form widget. If requested, a frame will be drawn
 *  around the form. This is normally only done for top level forms.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <X11/Intrinsic.h>
#include <Xm/Form.h>
#include <Xm/Frame.h>


Widget tui_create_form ( parent, name, frame_flag, arg_list, num_args )

    Widget          parent;     /* The parent widget to which the form widget will
                                 * be attached.
                                 */

    char            *name;      /* The name of the widget.  It uniquely
                                 * defines the widget.
                                 */

    Arg             arg_list[]; /* Optional argument list passed to the
                                 * createform widget.
                                 */

    int             frame_flag, /* If TRUE, draw a frame around the form.
                                 */
                    num_args;   /* Number of arguments in the arg_list.
                                 */
{
    Widget          widget;

/*
 *  If requested, draw a frame around the form widget.
 */

    if ( frame_flag )
        XtManageChild ( parent = XmCreateFrame ( parent, name, NULL, 0 ) );

/*
 *  Create and manage the form widget. Return the widget pointer to the
 *  calling function.
 */

    XtManageChild ( widget = XmCreateForm ( parent, name, arg_list, num_args ) );

    return ( widget );
}
```

```
/*****************************************************************************
 * MODULE NAME: tui_create_label
 *
 *  This function creates a MOTIF label gadget. The label string is passed in as a
 *  separate argument rather than being taken from the name.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <X11/Intrinsic.h>
#include <Xm/LabelG.h>


Widget tui_create_label ( parent, name, label, arg_list, num_args )

    Widget          parent;     /* The parent widget to which the label widget will
                                 * be attached.
                                 */

    char            *name,      /* The name of the widget.  It uniquely
                                 * defines the widget.
                                 */

                    *label;     /* The string which this label widget will display.
                                 */

    Arg             arg_list[]; /* List of arguments.
                                 */

    int             num_args;   /* Number of arguments in arg_list.
                                 */


{
    register int    count = 0,
                    i;

    Arg             args[10];

    Widget          widget;

    XmString        string;

/*
 *  Set up at most five arguments from arg_list.
 */

    if ( num_args > 5 )
        num_args = 5;
    for ( i = 0; i <= num_args-1; i++ ) {
        args[i] = arg_list[i];
        count++;
     }

/*
 *  Initialize a compound string and set in the argument list.
 */
```

```
    string = XmStringLtoRCreate ( label, XmSTRING_DEFAULT_CHARSET );

    XtSetArg ( args[count], XmNlabelType,   XmSTRING ); count++;
    XtSetArg ( args[count], XmNlabelString, string   ); count++;

/*
 *  Create and manage the widget. Free the space allocated for the compound string.
 *  Return the widget pointer to the calling function.
 */

    XtManageChild ( widget = XmCreateLabelGadget ( parent, name, args, count ) );
    XmStringFree ( string );

    return ( widget );
}
```

```
/*****************************************************************************
 * MOUDLE NAME: tui_create_pushbutton
 *
 *  This function is called to create a MOTIF Pushbutton gadget. The callback and
 *  closure values for the button callback are provided as arguments.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <X11/Intrinsic.h>
#include <Xm/PushBG.h>


Widget tui_create_pushbutton ( parent, name, callback, pointer, arg_list, num_args )

    Widget          parent;     /* The parent widget which this pushbutton widget
                                 * will be attached.
                                 */

    char            *name;      /* The instance of this widget. This uniquely
                                 * defines the widget.
                                 */

    XtCallbackProc  callback;   /* Specifies an array containing the list of func-
                                 * tions called upon command callback.  It may be
                                 * NULL if no functions are present.
                                 */

    caddr_t         pointer;    /* Pointer to the parameter passed to the callback.
                                 */

    Arg             arg_list[]; /* A list of arguments for expandability.
                                 */

    int             num_args;   /* Number of arguments in the arg_list.
                                 */
{
    register int    count = 0,
                    i;

    Arg             args[10];

    Widget          widget;

/*
 *  Set up at most 5 arguments from arg_list.
 */

    if ( num_args > 5 )
        num_args = 5;
    for ( i = 0; i <= num_args-1; i++ ) {
        args[i] = arg_list[i];
        count++;
    }

/*
 *  Create and manage the widget.
```

```
 */

    XtManageChild ( widget = XmCreatePushButtonGadget ( parent, name, args, count ) );

/*
 *  If the command has a callback, add it to the widget.
 */

    if ( callback )
        XtAddCallback ( widget, XmNactivateCallback, callback, pointer );

/*
 *  Return the widget.
 */

    return ( widget );
}
```

```
/*****************************************************************************
 * MODULE NAME: tui_create_toggle
 *
 *  This function creates a radio box widget and initializes it with a set of
 *  toggle button gadgets.
 *
 *
 * EXTERNAL FUNCTIONS:
 *
 *  o   tui_radio_get_value -   This function returns the current value (which
 *                              toggle is selected) from a selected radio box.
 *
 *  o   tui_radio_set_value -   This function sets the current value (which
 *                              toggle is set) for a selected radio box.
 *
 *
 * INTERNAL FUNCTIONS:
 *
 *  o   cb_radio            -   This function is called when a toggle is to
 *                              be destroyed. This function cleans up all
 *                              radio box resources.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <X11/Intrinsic.h>
#include <Xm/RowColumn.h>
#include <Xm/ToggleBG.h>


extern Widget   tui_create_toggle ();


struct radio_str {
    Widget  r_rb;
    int     r_num_labels;
    Widget  r_toggles[10];
    char    *r_labels[10];
} radios[10];

static num_radios = 0;


Widget tui_create_rb ( parent, name, labels, num_labels, def, arg_list, num_args )

    Widget          parent;     /* The parent widget to which the label
                                 * widget will be attached.
                                 */

    char            *name,      /* The name of the widget.  It uniquely
                                 * defines the widget.
                                 */
                    *labels[],  /* The strings which this widget will display.
                                 */
                    *def;       /* This string which is set (the toggle is enabled).
                                 */
```

```
    int             num_labels; /* The number of labels to display.
                                 */

    Arg             arg_list[]; /* List of arguments.
                                 */

    int             num_args;   /* Number of arguments in arg_list.
                                 */

{
    register int    count = 0,
                    i;

    Arg             args[10];

    Widget          widget,
                    widget1;

    XmString        string;

    XtCallbackProc  cb_radio();

    int             flag;

/*
 *  If the maximum number of managed radio boxes has been reached, output an error
 *  and return. Otherwise if the number of labels is > 10, set number to 10. MDC -
 *  fix to use a list.
 */

    if ( num_radios == 10 ) {
        printf ( "ERROR - Maximum number of managed radio boxes has been reached\n" );
        return ( NULL );
    } else if ( num_labels > 10 )
        num_labels = 10;

    num_radios++;

/*
 *  Set up at most five arguments from arg_list.
 */
    if ( num_args > 5 )
        num_args = 5;
    for ( i = 0; i <= num_args-1; i++ ) {
        args[i] = arg_list[i];
        count++;
    }

/*
 *  Create and manage the radio box widget and add the callback to the destroy
 *  function to insure that resources are cleaned up when the widget is destroyed.
 */

    XtManageChild ( widget = XmCreateRadioBox ( parent, name, args, count ));

    radios[num_radios-1].r_rb = widget;
    XtAddCallback ( widget, XmNdestroyCallback, cb_radio, 0 );

/*
 *  Create each of the toggle widgets to be placed in the radio box widget. If
 *  the widget is the one set, then enable the toggle.
 */

    radios[num_radios-1].r_num_labels = num_labels;
```

```
    count = 0;
    for ( i = 0; i < num_labels; i++ ) {
        flag = ( strcmp ( labels[i], def ) == 0 ) ? 1 : 0;
        widget1 = tui_create_toggle ( widget, "", labels[i], flag, NULL, 0, args, count );
        radios[num_radios-1].r_toggles[i] = widget1;
        radios[num_radios-1].r_labels [i] = labels[i];
    }

/*
 *  Return radio box widget.
 */

    return ( widget );
}
```

```
/*****************************************************************************
 * MODULE NAME: tui_radio_get_value
 *
 *  This function returns the current value of the specified radio box widget.
 *  The selected widget must have been created by the (tui_create_radio_box)
 *  function.
 *****************************************************************************/

char *tui_radio_get_value ( widget )

    Widget          widget;     /* The radio box from which to get a value.
                                 */
{
    register int    i,
                    r;

/*
 *  Search the array of radio boxes for a match with the specified widget. If
 *  no match is found, return an error.
 */

    for ( r = 0; r < num_radios; r++ )
        if ( radios[r].r_rb == widget )
            break;

    if ( r == num_radios )
        return ( (char *)-1 );

/*
 *  Scan each toggle in the radio box and if one is selected, return a pointer
 *  to the label corresponding to the toggle. Note that this pointer is only
 *  valid as long as the widget is created.
 */

    for ( i = 0; i < radios[r].r_num_labels; i++ )
        if ( XmToggleButtonGadgetGetState ( radios[r].r_toggles[i] ) )
            return ( radios[r].r_labels[i] );

/*
 *  Return NULL if no toggle is selected.
 */

    return ( (char *)NULL );
}
```

```
/*****************************************************************************
 * MODULE NAME: tui_radio_set_value
 *
 *  This function sets the value of the specified radio box widget.
 *  The selected widget must have been created by the (tui_create_radio_box)
 *  function.
 *****************************************************************************/

int tui_radio_set_value ( widget, value )

    Widget          widget;     /* The radio box to set the value on.
                                 */

    char            *value;     /* The value to set.
                                 */
{
    register int        i,
                        r;

/*
 *  Search the array of radio boxes for a match with the specified widget. If
 *  no match is found, return an error.
 */

    for ( r = 0; r < num_radios; r++ )
        if ( radios[r].r_rb == widget )
            break;

    if ( r == num_radios )
        return ( -1 );

/*
 *  Scan the array of labels for the radio box for a match with the passed string.
 *  If a match is found, set the state to TRUE. For each label not matched, set the
 *  state to FALSE. This is necessary because the radio box behavior is only enforced
 *  for interactive selection.
 */

    for ( i = 0; i < radios[r].r_num_labels; i++ )
        if ( strcmp ( radios[r].r_labels[i], value ) == 0 )
            XmToggleButtonGadgetSetState ( radios[r].r_toggles[i], TRUE  );
        else
            XmToggleButtonGadgetSetState ( radios[r].r_toggles[i], FALSE );

    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: cb_radio
 *
 *  This callback function is called when one of the created radio widgets is
 *  destroyed. It is responsible for removal of the widget resources.
 *****************************************************************************/

static XtCallbackProc cb_radio ( w, closure, calldata )

    Widget  w;                  /* Set to the widget which initiated this
                                 * callback function.
                                 */

    caddr_t closure,            /* Callback specific data.  This parameter
                                 * is set to button selected.
                                 */
            *calldata;
                                /* Specifies any callback-specific data the
                                 * widget needs to pass to the client.
                                 */
{
    register int    i;

/*
 *  Decrement the number of radio structures. If the one removed was the last,
 *  exit from this function. Note that there is no need to destroy the widget
 *  because it is already being destroyed.
 */

    num_radios--;

    if ( num_radios == 0 )
        return;

/*
 *  Otherwise there is more than radio widget, so find the one which is being
 *  destroyed and then copy the last structure on top of it.
 */

    for ( i = 0; i < num_radios; i++ )
        if ( radios[i].r_rb == w )
            memcpy ( (char *)&radios[i], (char *)&radios[num_radios],
                     sizeof ( struct radio_str ) );

/*
 *  Normal return.
 */

    return;
}
```

```
/*****************************************************************************
 * MODULE NAME: tui_create_scale
 *
 *  This function creates a MOTIF scale widget. This function creates a scale
 *  with a set of low and high limits and corresponding labels.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <X11/Intrinsic.h>
#include <Xm/Scale.h>


Widget tui_create_scale ( parent, name, min, max, def, labels, num_labels,
                          arg_list, num_args )

    Widget          parent;     /* The parent widget to which the scale widget will
                                 * be attached.
                                 */

    char            *name,      /* The name of the widget.  It uniquely defines the
                                 * widget.
                                 */
                    **labels;   /* The list of labels to place in the scale.
                                 */

    int             min,        /* The minimum value of the scale bar.
                                 */

                    max,        /* The maximum value of the scale bar.
                                 */

                    def,        /* The default (or current) value of the scale bar.
                                 */
                    num_labels; /* The number of labels to place inside the scale.
                                 */

    Arg             arg_list[]; /* List of arguments.
                                 */

    int             num_args;   /* Number of arguments in arg_list.
                                 */
{
    register int    count = 0,
                    i;

    Arg             args[10];

    Widget          widget;

/*
 *  Set up at most five arguments from arg_list.
 */

    if ( num_args > 5 )
        num_args = 5;
    for ( i = 0; i <= num_args-1; i++ ) {
```

```
        args[i] = arg_list[i];
        count++;
    }

/*
 *  Set the arguments for the minimum, maximum, and default values.
 */

    XtSetArg ( args[count], XmNminimum, min ); count++;
    XtSetArg ( args[count], XmNmaximum, max ); count++;
    XtSetArg ( args[count], XmNvalue,   def ); count++;

/*
 *  Create and manage the scale widget.
 */

    XtManageChild ( widget = XmCreateScale ( parent, name, args, count ) );

/*
 *  Add all labels to the scale widget.
 */

    for ( i = 0; i < num_labels; i++ )
        tui_create_label ( widget, "", *(labels+i), args, 0 );

/*
 *  Return the widget pointer.
 */

    return ( widget );
}
```

```
/*****************************************************************************
 * MODULE NAME: tui_create_sel
 *
 *  This function creates a selection list for use in a form. This function creates
 *  the list with a set of selections and the text widget which may be used to
 *  directly enter a selection.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include <Xm/mwm.h>
#include <Xm/SelectioB.h>


Widget tui_create_sel ( parent, name, list, num_items, label, arg_list, num_args )

    Widget          parent;     /* The parent widget which will contain the selection
                                 * widget.
                                 */

    Arg             arg_list[]; /* List of arguments to widget.
                                 */

    char            *name,      /* The name to give the widget. The name uniquely
                                 * identifies the widget.
                                 */
                    **list,     /* List of items (character strings) to put into
                                 * the list.
                                 */
                    *label;     /* The label to place above the list.
                                 */

    int             num_items,  /* The number of items in the list.
                                 */
                    num_args;   /* The number of arguments in arg_list.
                                 */
{
    register int    i,
                    count = 0;

    Arg             args[10];

    Widget          widget,
                    wlist;

    XmString        string1,
                    string2;

/*
 *  Set up at most five arguments from arg_list.
 */

    if ( num_args > 5 )
        num_args = 5;
    for ( i = 0; i <= num_args-1; i++ ) {
```

```
        args[i] = arg_list[i];
        count++;
    }

/*
 *  Set up all resource values. Prevent the popup from disappearing when a button
 *  is selected and initialize message title string.
 */

    XtSetArg ( args[count], XmNautoUnmanage, FALSE ); count++;

    string1 = XmStringLtoRCreate ( label, XmSTRING_DEFAULT_CHARSET );
    XtSetArg ( args[count], XmNlistLabelString, string1 ); count++;

/*
 *  Create the selection list widget.
 */

    widget = XmCreateSelectionBox ( parent, name, args, count );

/*
 *  Unmanage all unneeded buttons.
 */

    XtUnmanageChild ( XmSelectionBoxGetChild ( widget, XmDIALOG_CANCEL_BUTTON    ) );
    XtUnmanageChild ( XmSelectionBoxGetChild ( widget, XmDIALOG_OK_BUTTON        ) );
    XtUnmanageChild ( XmSelectionBoxGetChild ( widget, XmDIALOG_HELP_BUTTON      ) );
    XtUnmanageChild ( XmSelectionBoxGetChild ( widget, XmDIALOG_APPLY_BUTTON     ) );
    XtUnmanageChild ( XmSelectionBoxGetChild ( widget, XmDIALOG_SELECTION_LABEL ) );
    XtUnmanageChild ( XmSelectionBoxGetChild ( widget, XmDIALOG_SEPARATOR        ) );

/*
 *  Retrieve the actual list widget from the selection list and add each item to
 *  be displayed in the list.
 */

    wlist = XmSelectionBoxGetChild ( widget, XmDIALOG_LIST );

    for ( count = 0; count < num_items; count++ ) {
        string2 = XmStringCreateLtoR ( *(list+count), XmSTRING_DEFAULT_CHARSET );
        XmListAddItem ( wlist, string2, 0 );
        XmStringFree ( string2 );
    }

/*
 *  Manage the widget and free the string used for the label.
 */

    XtManageChild ( widget );

    XmStringFree ( string1 );

/*
 *  Return the text widget (as opposed to the selection list), so that the calling
 *  function can easily get the selection.
 */

    return ( XmSelectionBoxGetChild ( widget, XmDIALOG_TEXT ) );
}
```

```
/*****************************************************************************
 * MODULE NAME: ( tui_create_separator )
 *
 *  This function is used to create a MOTIF separator gadget. Separators are
 *  used in forms to separate logical areas of control and data entry.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Mark D. Collier -   Software Engineering Section
 *                      Data Systems Department
 *                      Automation and Data Systems Division
 *                      Southwest Research Institute
 *****************************************************************************/


#include <X11/Intrinsic.h>
#include <Xm/SeparatoG.h>


Widget tui_create_separator ( parent, name, arg_list, num_args )

    char            *name;      /* The instance name of the widget.  It uniquely
                                 * defines the widget.
                                 */

    Widget          parent;     /* The parent widget to which the separator
                                 * widget will be attached.
                                 */

    Arg             arg_list[]; /* A list of arguments to be used for this
                                 * instance of the widget.
                                 */

    int             num_args;   /* The number of args in arg_list.
                                 */
{
    int             count = 0,
                    i;

    Arg             args[10];

    Widget          widget;

/*
 *  Set up at most the first five arguments of arg_list.
 */

    if ( num_args > 5 )
        num_args = 5;
    for ( i = 0; i < num_args; i++ ) {
        args[i] = arg_list[i];
        count++;
    }

/*
 *  Create the separator widget and return widget pointer.
 */

    XtManageChild ( widget = XmCreateSeparatorGadget ( parent, name, args, count ) );

    return ( widget );
}
```

```
/*****************************************************************************
 * MODULE NAME: tui_create_text
 *
 *  This function creates a MOTIF text widget. The text widget created can be a
 *  single or multi-line field, and can be editable or read-only.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <X11/Intrinsic.h>
#include <Xm/Text.h>


Widget tui_create_text ( parent, name, text, max_length, mode, edit_flag,
                         arg_list, num_args )

    Widget          parent;     /* The parent widget to which the text widget will
                                 * be attached.
                                 */

    char            *name,      /* The name of the widget.  It uniquely
                                 * defines the widget.
                                 */

                    *text;      /* The ascii text which will be displayed in the
                                 * text widget.
                                 */

    int             max_length, /* Maximum allowable length of the text to be input
                                 * by the user.
                                 */
                    mode,       /* Indicates whether the widget will be single or
                                 * multiple lines:
                                 *
                                 *   XmSINGLE_LINE_EDIT
                                 *   XmMULTI_LINE_EDIT
                                 */

                    edit_flag;  /* Indicates whether or not the widget can be
                                 * edited.
                                 */

    Arg             arg_list[]; /* A list of arguments for expandability.
                                 */

    int             num_args;   /* Number of arguments in arg_list.
                                 */
{
    register int    count = 0,
                    i;

    Arg             args[10];

    Widget          widget;

/*
 *  Set up at most five arguments from arg_list.
```

```
*/

    if ( num_args > 5 )
        num_args = 5;

    for ( i = 0; i <= num_args-1; i++ ) {
        args[i] = arg_list[i];
        count++;
    }

/*
 *  Initialize the default text (not a compound string), the line size mode, the
 *  edit mode, and the maximum number of characters to be entered.
 */

    XtSetArg ( args[count], XmNvalue,     text       ); count++;
    XtSetArg ( args[count], XmNeditMode,  mode       ); count++;
    XtSetArg ( args[count], XmNeditable,  edit_flag  ); count++;
    XtSetArg ( args[count], XmNmaxLength, max_length ); count++;

/*
 *  Based on the (mode) flag, create the appropriate type of widget. Next manage
 *  the widget. Note that the instance name of a scrolled text widget is "instanceSW".
 */

    if ( mode == XmMULTI_LINE_EDIT )
        widget = XmCreateScrolledText ( parent, name, args, count );
    else
        widget = XmCreateText         ( parent, name, args, count );

    XtManageChild ( widget );

/*
 *  Return the widget pointer.
 */

    return ( widget );
}
```

```
/*****************************************************************************
 * MODULE NAME: tui_create_toggle
 *
 *  This function creates a MOTIF toggle button gadget. The toggle can be created
 *  in either a set or unset state.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <X11/Intrinsic.h>
#include <Xm/ToggleBG.h>


Widget tui_create_toggle ( parent, name, label, set, cb_name, cb_data, arg_list, num_args
)

    Widget          parent;     /* The parent widget to which the label widget will
                                 * be attached.
                                 */

    char            *name,      /* The name of the widget.  It uniquely
                                 * defines the widget.
                                 */

                    *label;     /* The string which this label widget
                                 * will display.
                                 */

    int             set;        /* If TRUE, then the toggle is set; if FALSE,
                                 * the toggle is unselected.
                                 */

    XtCallbackProc  cb_name;    /* Used to specify a callback function to call
                                 * if this toggle is selected by the user.
                                 */

    caddr_t         cb_data;    /* Value to passed to the callback function.
                                 */

    Arg             arg_list[]; /* List of arguments.
                                 */

    int             num_args;   /* Number of arguments in arg_list.
                                 */

{
    register int    count = 0,
                    i;

    Arg             args[10];

    Widget          widget;

    XmString        string;

/*
 *  Set up at most five arguments from arg_list.
```

```
 */

    if ( num_args > 5 )
        num_args = 5;
    for ( i = 0; i <= num_args-1; i++ ) {
        args[i] = arg_list[i];
        count++;
    }

/*
 *  Set the label for the toggle gadget.
 */

    string = XmStringLtoRCreate ( label, XmSTRING_DEFAULT_CHARSET );
    XtSetArg ( args[count], XmNlabelType,   XmSTRING ); count++;
    XtSetArg ( args[count], XmNlabelString, string   ); count++;

/*
 *  Create and manage the toggle widget.
 */

    XtManageChild ( widget = XmCreateToggleButtonGadget ( parent, name, args, count ) );

/*
 *  If a callback function was specified, add the appropriate callback to the
 *  toggle widget.
 */

    if ( cb_name )
        XtAddCallback ( widget, XmNarmCallback, cb_name, cb_data );

/*
 *  If the toggle is to be in a selected state, set the state of the toggle to
 *  TRUE.
 */

    if ( set == TRUE )
        XmToggleButtonGadgetSetState ( widget, TRUE, FALSE );

/*
 *  Free the compound string and return widget.
 */

    XmStringFree ( string );

    return ( widget );
}
```

```
/*****************************************************************************
 * MODULE NAME: ( tui_create_trans_shell )
 *
 *  This function creates a transient shell widget which is used to create a
 *  popup which is temporary in nature. This function is provided to simplify
 *  this process and to set the correct arguments so that the window is appli-
 *  cation modal (all other windows are disabled - except application shells)
 *  and so that it does not have a CLOSE function in the window manager menu.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <X11/Shell.h>
#include <Xm/Xm.h>
#include <Xm/mwm.h>


extern Widget       Top,        /* The top level widget which is the parent
                                 * of transient shells.
                                 */
                    Cur_shell;  /* Keeps track of the popup widget currently
                                 * displayed.
                                 */


Widget tui_create_trans_shell ( name, arg_list, num_args )

    char        *name;          /* The name of the widget. This string will
                                 * appear in the title bar of the shell.
                                 */

    Arg         arg_list[];     /* List of arguments to be used in this
                                 * instance of the widget.
                                 */

    int         num_args;       /* The number of arguments in the argument
                                 * list;
                                 */

{
    register int    i,
                    count = 0;

    Arg             args[10];

    Widget          widget;

    XtCallbackProc  cb_destroy_shell();

    int             x,
                    y;

/*
 *  Retrieve the location of the top level shell. The location will be used to place
 *  the new popup.
```

```
 */

    XtSetArg ( args[count], XmNx, &x ); count++;
    XtSetArg ( args[count], XmNy, &y ); count++;
    XtGetValues ( Top, args, count );

/*
 *  Set up at most five arguments from arg_list.
 */

    count = 0;
    if ( num_args > 5 )
        num_args = 5;
    for ( i = 0; i <= num_args-1; i++ ) {
        args[i] = arg_list[i];
        count++;
    }

/*
 *  Initialize arguments to place the popup on top of the control panel window.
 */

    XtSetArg ( args[count], XmNx, x ); count++;
    XtSetArg ( args[count], XmNy, y ); count++;

/*
 *  Create the transient shell widget.
 */

    widget = XtCreatePopupShell ( name, transientShellWidgetClass, Top, args, count );

/*
 *  Save the shell widget and record a callback so that the application knows when
 *  the shell is removed. This is necessary to change the cursor in the current
 *  popup window.
 */

    Cur_shell = widget;
    XtAddCallback ( widget, XmNdestroyCallback, cb_destroy_shell, 0 );

/*
 *  Force the popup to be application modal, which means that all other
 *  windows (except application shells) will refuse input. Specify the
 *  functions which will be available in the Motif window manager menu area.
 *  Note that the CLOSE function is absent. Note also that these arguments are
 *  set after creation, because they appear to be ignored at that time.
 */

    count = 0;
    XtSetArg ( args[count], XmNmwmInputMode,
        MWM_INPUT_APPLICATION_MODAL           ); count++;
    XtSetArg ( args[count], XmNmwmFunctions,
        MWM_FUNC_MOVE     | MWM_FUNC_RESIZE |
        MWM_FUNC_MINIMIZE | MWM_FUNC_MAXIMIZE ); count++;
    XtSetValues ( widget, args, count );

/*
 *  Return the created application shell widget.
 */

    return ( widget );
}
```

```
/*****************************************************************************
 * MODULE NAME: tui_get_list
 *
 *  This function displays a MOTIF scrolled list and returns when the user selects
 *  an item from the list. This popup can include either the normal OK button or
 *  a START and STOP buttons.
 *
 *
 * INTERNAL FUNCTIONS:
 *
 *  o   cb_list -   This is a callback function which is called when the OK, APPLY,
 *                  or CANCEL button is selected from the popup.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include <Xm/mwm.h>
#include <Xm/SelectioB.h>
#include <wex/EXmsg.h>


extern Widget   Cur_shell;          /* Keeps track of the popup widget currently
                                     * displayed.
                                     */


static Widget   widget;
static int      flag;
static char     *item_t = NULL;


int tui_get_list ( parent, list, num_items, item, title, label, start_stop, help_index,
                   arg_list, num_args )

    Widget          parent;     /* Main window of the application. Used to attach
                                 * the shell widget.
                                 */

    char            **list,     /* List of items to put into the list.
                                 */
                    *item,      /* Updated to the selected item.
                                 */
                    *title,     /* The title of the popup.
                                 */
                    *label;     /* The label for the list of selections.
                                 */

    int             num_items,  /* Number of items in the list.
                                 */
                    start_stop, /* Indicates the type of popup. If set, buttons
                                 * for START and STOP will be displayed. Otherwise,
                                 * only an OK button will be displayed.
                                 */
                    help_index, /* Indicates help text to be displayed for the
                                 * the popup.
```

```
                                */
                    num_args;   /* Number of arguments in arg_list.
                                */

    Arg             arg_list[]; /* List of arguments to widget.
                                */
{
    register int    count = 0,
                    i;

    Arg             args[15];

    Widget          wlist;

    XEvent          event;

    XmString        string,
                    string1,
                    string2,
                    string3,
                    string4;

    XtCallbackProc  cb_destroy_shell(),
                    cb_help         (),
                    cb_list         ();

/*
 *  Set up at most five arguments from arg_list.
 */

    if ( num_args > 5 )
        num_args = 5;
    for ( i = 0; i <= num_args-1; i++ ) {
        args[i] = arg_list[i];
        count++;
    }

/*
 *  Set up all resource values. Prevent the popup from disappearing when a button
 *  is selected, turn on application modal to lock out other windows, prevent the
 *  user from entering a string not in the list, and initialize message and title
 *  strings.
 */

    XtSetArg ( args[count], XmNautoUnmanage, FALSE                      ); count++;
    XtSetArg ( args[count], XmNdialogStyle,  XmDIALOG_APPLICATION_MODAL ); count++;
    XtSetArg ( args[count], XmNmustMatch,    TRUE                       ); count++;

    string1 = XmStringLtoRCreate ( label, XmSTRING_DEFAULT_CHARSET );
    XtSetArg ( args[count], XmNlistLabelString, string1 ); count++;

    string2 = XmStringLtoRCreate ( title, XmSTRING_DEFAULT_CHARSET );
    XtSetArg ( args[count], XmNdialogTitle, string2 ); count++;

/*
 *  If (start_stop) is set, set the labels for the OK and APPLY button to START and
 *  STOP.
 */

    if ( start_stop ) {
        string3 = XmStringLtoRCreate ( "START", XmSTRING_DEFAULT_CHARSET );
        XtSetArg ( args[count], XmNokLabelString,    string3 ); count++;
        string4 = XmStringLtoRCreate ( "STOP",  XmSTRING_DEFAULT_CHARSET );
        XtSetArg ( args[count], XmNapplyLabelString, string4 ); count++;
```

```
    }

/*
 *  Create the list popup.
 */

    widget = XmCreateSelectionDialog ( parent, "", args, count );

/*
 *  Save the shell widget and record a callback so that the application knows when
 *  the shell is removed. This is necessary to change the cursor in the current
 *  popup window.
 */

    Cur_shell = widget;
    XtAddCallback ( widget, XmNdestroyCallback, cb_destroy_shell, 0 );

/*
 *  Initialize all callbacks.
 */

    XtAddCallback ( widget, XmNcancelCallback,  cb_list, (caddr_t)0 );
    XtAddCallback ( widget, XmNokCallback,      cb_list, (caddr_t)1 );
    XtAddCallback ( widget, XmNnoMatchCallback, cb_list, (caddr_t)3 );
    XtAddCallback ( widget, XmNhelpCallback,    cb_help, help_index );

/*
 *  If (start_stop) is set, manage and add callback for the APPLY button.
 */

    if ( start_stop ) {
        XtAddCallback ( widget, XmNapplyCallback,  cb_list, (caddr_t)2 );
        XtManageChild ( XmSelectionBoxGetChild ( widget, XmDIALOG_APPLY_BUTTON ) );
    }

/*
 *  Update the list with the array of valid selections.
 */

    wlist = XmSelectionBoxGetChild ( widget, XmDIALOG_LIST );

    for ( count = 0; count < num_items; count++ ) {
        string = XmStringCreateLtoR ( *(list+count), XmSTRING_DEFAULT_CHARSET );
        XmListAddItem ( wlist, string, 0 );
        XmStringFree ( string );
    }

/*
 *  Manage the widget and all compound strings.
 */

    XtManageChild ( widget );

    XmStringFree ( string1 );
    XmStringFree ( string2 );
    if ( start_stop ) {
        XmStringFree ( string3 );
        XmStringFree ( string4 );
    }

    set_cmap ( XtParent ( widget ) );

/*
 *  Wait until a button is selected.
```

```
 */

    flag = -1;
    while ( flag == -1 ) {
        XtNextEvent     ( &event );
        XtDispatchEvent ( &event );
    }

/*
 *  If the user selected OK, copy the selection string to the parameter (item).
 */

    if ( flag )
        strcpy ( item, item_t );

/*
 *  Destroy the widget.
 */

    XtDestroyWidget ( widget );

/*
 *  Return the value selected by the user (0 is for no item selected and 1 is
 *  for an item selected).
 */

    return ( flag );
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( cb_list )
 *
 *  This callback function is called when the OK, APPLY, or CANCEL button is
 *  selected from the popup.
 *****************************************************************************/

static XtCallbackProc cb_list ( w, closure, sb )

    Widget  w;                  /* Set to the widget which initiated this
                                 * callback function.
                                 */

    caddr_t closure;            /* Callback specific data.  This parameter
                                 * is set to button selected.
                                 */
    XmSelectionBoxCallbackStruct *sb;
                                /* Specifies any callback-specific data the
                                 * widget needs to pass to the client.
                                 */
{
/*
 *  If the user selected either OK or APPLY (APPLY only available for START/STOP
 *  operation), retrieve the selected item. If an item was selected, set (flag) to
 *  the value of (closure), which will cause the popup to be removed.
 */

    if ( (int)closure == 1 || (int)closure == 2 ) {
        XmStringGetLtoR ( sb->value, XmSTRING_DEFAULT_CHARSET, &item_t );
        if ( *item_t )
            flag = (int)closure;

/*
 *  Otherwise, if (closure) is 3, the user entered a string which does not match
 *  any in the list, so generate a warning.
 */

    } else if ( (int)closure == 3 ) {
        tui_msg ( M_YELLOW, "Selection does not match any in list" );

/*
 *  Otherwise, the user selected CANCEL, so set (flag) to 0.
 */

    } else {
        flag = (int)closure;
    }

/*
 *  Normal return.
 */

    return;
}
```

```
/*****************************************************************************
 * MODULE NAME: tui_msg
 *
 *  This function displays different types of popups for different message types.
 *  It displays a non-modal popup which when acknowledged, is automatically
 *  removed. Note that this function also calls EXmsg. Note also that only one
 *  message of this type will ever be displayed because this function always re-
 *  moves the previous (if any) message.
 *
 *
 * EXTERNAL FUNCTIONS:
 *
 *  o   tui_msg_control -   This function allows the display of popup messages to
 *                          be enabled and disabled.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include <Xm/mwm.h>
#include <Xm/MessageB.h>
#include <wex/EXmsg.h>


extern Widget   Top;            /* The top level widget of the Display Manager
                                 * application.
                                 */

static int      popup_flag = FALSE;
                                /* Indicates whether or not popup messages should
                                 * be displayed.
                                 */


int tui_msg ( type, format, p1, p2, p3, p4, p5, p6, p7, p8 )

    int             type;       /* Type of the message. Used to determine the type
                                 * of popup displayed.
                                 */

    char            *format;    /* Format string for the sprintf.
                                 */

    char            *p1, *p2, *p3, *p4, *p5, *p6, *p7, *p8;
                                /* Parameter values for the string.
                                 */
{
    register int    count = 0;

    static Widget   widget = NULL;

    Arg             args[10];

    XmString        string;

    char            message[200];
```

```
/*
 *  Format the string to be displayed and call EXmsg to display it.
 */

    sprintf ( message, format, p1, p2, p3, p4, p5, p6, p7, p8 );
    EXmsg ( format, format, p1, p2, p3, p4, p5, p6, p7, p8 );

/*
 *  If popup messages are not to be displayed, return immediately.
 */

    if ( popup_flag == FALSE )
        return ( 0 );

/*
 *  If a popup was already defined, destroy it (it will have been unmanaged, but
 *  will still exist.
 */

    if ( widget )
        XtDestroyWidget ( widget );

/*
 *  Initialize the string to be displayed in the popup.
 */

    string = XmStringLtoRCreate ( message, XmSTRING_DEFAULT_CHARSET );
    XtSetArg ( args[count], XmNmessageString, string ); count++;

/*
 *  Based on the message type, create the appropriate popup type.
 */

    switch ( type ) {

    case    M_BLUE:
    case    M_WHITE:
    case    M_GREEN:
        widget = XmCreateInformationDialog ( Top, "Display Manager Message", args, count )
;
        break;
    case    M_YELLOW:
        widget = XmCreateWarningDialog     ( Top, "Display Manager Message", args, count )
;
        break;
    case    M_RED:
    case    M_CRITICAL:
        widget = XmCreateErrorDialog       ( Top, "Display Manager Message", args, count )
;
        break;
    default:
        break;
    }

/*
 *  Manage the widget and Free the string used for the compound string.
 */

    XtManageChild ( widget );

    XmStringFree ( string );

    set_cmap ( XtParent ( widget ) );
```

```
/*
 *  Unmanage the CANCEL and HELP push buttons as they have no function.
 */

    XtUnmanageChild ( XmMessageBoxGetChild ( widget, XmDIALOG_CANCEL_BUTTON ) );
    XtUnmanageChild ( XmMessageBoxGetChild ( widget, XmDIALOG_HELP_BUTTON   ) );

/*
 *  Normal return.
 */

    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: tui_msg_control
 *
 *  This function is called to turn on or off the display of popup messages.
 *****************************************************************************/

int tui_msg_control ( flag )

    int     flag;                   /* If set to TRUE, messages will be displayed; if
                                     * FALSE, messages will not be displayed.
                                     */
{
/*
 *  Set static flag to the value of parameter.
 */

    popup_flag = flag;

/*
 *  Normal return.
 */

    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: tui_get_prompt
 *
 *  This function displays a Motif popup which waits for the user to enter a string
 *  and select OK. The calling function specifies the function to call when the OK
 *  button is selected. If this function returns TRUE, this function will return to
 *  the calling function.
 *
 *
 * INTERNAL FUNCTIONS:
 *
 *  o   cb_func -   This function is called to process selection of the OK and
 *                  CANCEL buttons on the popup. This function will call the user-
 *                  defined function if OK is selected.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include <Xm/mwm.h>
#include <Xm/SelectioB.h>
#include <Xm/Text.h>


extern Widget   Cur_shell;          /* Keeps track of the popup widget currently
                                     * displayed.
                                     */


static Widget   widget;
static int      flag,
                (*user_func)();


int tui_get_prompt ( parent, title, label, value, help_index, func, arg_list, num_args )

    Widget              parent;     /* Main window of the application. Used to attach
                                     * the shell widget.
                                     */

    char                *title,     /* The title of the popup.
                                     */
                        *label,     /* Label for the text input widget.
                                     */
                        *value;     /* String set to the initial value.
                                     */

    Arg                 arg_list[]; /* List of arguments to widget.
                                     */

    int                 help_index, /* Index to the help text to be displayed for
                                     * this popup.
                                     */
                        num_args;   /* Number of arguments in arg_list.
                                     */
```

```
    int                 (*func)();  /* Function to call when the user selects the OK
                                     * button.
                                     */
{
    register int            count = 0,
                            i;

    static  XtCallbackRec   cb[] = {
        { (XtCallbackProc)NULL, (caddr_t)NULL, },
        { (XtCallbackProc)NULL, (caddr_t)NULL  }
    };

    Arg                     args[15];

    XEvent                  event;

    XmString                string1,
                            string2,
                            string3;

    XtCallbackProc          cb_destroy_shell(),
                            cb_func         (),
                            cb_help         ();

/*
 *  Save pointer to user function so that it can be used by the callback function.
 */

    user_func = func;

/*
 *  Set up at most five arguments from arg_list.
 */

    if ( num_args > 5 )
        num_args = 5;
    for ( i = 0; i <= num_args-1; i++ ) {
        args[i] = arg_list[i];
        count++;
    }

/*
 *  Set up all resource values. Prevent the popup from disappearing when a button
 *  is selected, turn on application modal to lock out other windows, and initialize
 *  message and title strings.
 */

    XtSetArg ( args[count], XmNautoUnmanage, FALSE                       ); count++;
    XtSetArg ( args[count], XmNdialogStyle,  XmDIALOG_APPLICATION_MODAL ); count++;

    string1 = XmStringLtoRCreate ( value, XmSTRING_DEFAULT_CHARSET );
    XtSetArg ( args[count], XmNtextString, string1 ); count++;

    string2 = XmStringLtoRCreate ( label, XmSTRING_DEFAULT_CHARSET );
    XtSetArg ( args[count], XmNselectionLabelString, string2 ); count++;

    string3 = XmStringLtoRCreate ( title, XmSTRING_DEFAULT_CHARSET );
    XtSetArg ( args[count], XmNdialogTitle, string3 ); count++;

/*
 *  Create the prompt popup.
 */

    widget = XmCreatePromptDialog ( parent, "", args, count );
```

```
/*
 *  Save the shell-widget and record a callback so that the application knows when
 *  the shell is removed. This is necessary to change the cursor in the current
 *  popup window.
 */

    Cur_shell = widget;
    XtAddCallback ( widget, XmNdestroyCallback, cb_destroy_shell, 0 );

/*
 *  Add callbacks for the OK, CANCEL, and HELP buttons.
 */

    XtAddCallback ( widget, XmNcancelCallback, cb_func, (caddr_t)0 );
    XtAddCallback ( widget, XmNokCallback,     cb_func, (caddr_t)1  );
    XtAddCallback ( widget, XmNhelpCallback,   cb_help, help_index  );

/*
 *  Manage the widget and free the strings used for the message, title, and value.
 */

    XtManageChild ( widget );

    XmStringFree ( string1 );
    XmStringFree ( string2 );
    XmStringFree ( string3 );

    set_cmap ( XtParent ( widget ) );

/*
 *  Wait until either OK or CANCEL is selected.
 */

    flag = -1;
    while ( flag == -1 ) {
        XtNextEvent     ( &event );
        XtDispatchEvent ( &event );
    }

/*
 *  Destroy the popup widget and return the status of the popup to the user (0 is for
 *  CANCEL, 1 is for OK).
 */

    XtDestroyWidget ( widget );

    return ( flag );
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( cb_func )
 *
 *  This callback is called when the user selects either OK or CANCEL from the
 *  popup.
 *****************************************************************************/

static XtCallbackProc cb_func ( w, closure, sb )

    Widget  w;                  /* Set to the widget which initiated this
                                 * callback function.
                                 */

    caddr_t closure;            /* Callback specific data.  This parameter
                                 * is set to button selected.
                                 */
    XmSelectionBoxCallbackStruct    *sb;
                                /* Specifies any callback-specific data the
                                 * widget needs to pass to the client.
                                 */
{
    char        *string;
/*
 *  If the user selected OK, retrieve the selection string and pass to the user-defined
 *  function. If this function returns TRUE, set flag to cause removal of the popup.
 */

    if ( (int)closure == 1 ) {
        XmStringGetLtoR ( sb->value, XmSTRING_DEFAULT_CHARSET, &string );
        if ( (*user_func)( string ) == 1 )
            flag = (int)closure;

/*
 *  Otherwise, if the user selected CANCEL, so set flag to 0.
 */

    } else
        flag = (int)closure;

/*
 *  Normal return.
 */

    return;
}
```

```
/*****************************************************************************
 * MODULE NAME: tui_display_question
 *
 *  This function displays a prompt and waits for the user to respond. This func-
 *  tion returns TRUE if the user selects OK and FALSE if the user selects CANCEL.
 *  This function will automatically call the help function to display help if the
 *  user selets the HELP button.
 *
 *
 *  INTERNAL FUNCTIONS:
 *
 *  o   cb_question -   This callback function processes selection of either the
 *                      OK or CANCEL button.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include <Xm/mwm.h>
#include <Xm/MessageB.h>


extern Widget       Cur_shell;  /* Keeps track of the popup widget currently
                                 * displayed.
                                 */


static Widget       widget;
static int          flag;


int tui_display_question ( parent, title, message_text, help_index, arg_list, num_args )

    Widget          parent;     /* Main window of the application. Used to attach
                                 * the shell widget.
                                 */

    char            *title,     /* The title of the popup.
                                 */
                    *message_text;
                                /* Message text to actually display.
                                 */

    Arg             arg_list[]; /* List of arguments to widget.
                                 */

    int             help_index, /* Index to the help text to be displayed for
                                 * this popup.
                                 */
                    num_args;   /* Number of arguments in arg_list.
                                 */
{
    register int    count = 0,
                    i;

    Arg             args[15];
```

```
    XEvent          event;

    XmString        string1,
                    string2;

    XtCallbackProc  cb_destroy_shell(),
                    cb_question     (),
                    cb_help         ();

/*
 *  Set up at most five arguments from arg_list.
 */

    if ( num_args > 5 )
        num_args = 5;
    for ( i = 0; i <= num_args-1; i++ ) {
        args[i] = arg_list[i];
        count++;
    }

/*
 *  Set up all resource values. Prevent the popup from disappearing when a button
 *  is selected, turn on application modal to lock out other windows, and initialize
 *  message and title strings.
 */

    XtSetArg ( args[count], XmNautoUnmanage, FALSE                        ); count++;
    XtSetArg ( args[count], XmNdialogStyle,  XmDIALOG_APPLICATION_MODAL ); count++;

    string1 = XmStringLtoRCreate ( message_text, XmSTRING_DEFAULT_CHARSET );
    XtSetArg ( args[count], XmNmessageString, string1 ); count++;

    string2 = XmStringLtoRCreate ( title, XmSTRING_DEFAULT_CHARSET );
    XtSetArg ( args[count], XmNdialogTitle, string2 ); count++;

/*
 *  Create the question dialog popup.
 */

    widget = XmCreateQuestionDialog ( parent, "Display Manager Verification",
                                      args, count );

/*
 *  Save the shell widget and record a callback so that the application knows when
 *  the shell is removed. This is necessary to change the cursor in the current
 *  popup window.
 */

    Cur_shell = widget;
    XtAddCallback ( widget, XmNdestroyCallback, cb_destroy_shell, 0 );

/*
 *  Add callbacks for the OK, CANCEL, and HELP buttons.
 */

    XtAddCallback ( widget, XmNokCallback,     cb_question, (caddr_t)1 );
    XtAddCallback ( widget, XmNcancelCallback, cb_question, (caddr_t)0 );
    XtAddCallback ( widget, XmNhelpCallback,   cb_help,     help_index );

/*
 *  Manage the widget and free the strings used for the message and title.
 */
```

```
    XtManageChild ( widget );

    XmStringFree ( string1 );
    XmStringFree ( string2 );

    set_cmap ( XtParent ( widget ) );

/*
 *  Wait until either OK or CANCEL is selected.
 */

    flag = -1;
    while ( flag == -1 ) {
        XtNextEvent     ( &event );
        XtDispatchEvent ( &event );
    }

/*
 *  Destroy the popup widget and return the status of the popup to the user (0 is for
 *  CANCEL, 1 is for OK).
 */

    XtDestroyWidget ( widget );

    return ( flag );
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( cb_question )
 *
 *  This callback is called when the user selects either OK or CANCEL from the
 *  popup.
 *****************************************************************************/

static XtCallbackProc cb_question ( widget, closure, calldata )

    Widget  widget;             /* Set to the widget which initiated this
                                 * callback function.
                                 */

    caddr_t closure,            /* Callback specific data.  This parameter
                                 * is set to button selected.
                                 */
            *calldata;
                                /* Specifies any callback-specific data the
                                 * widget needs to pass to the client.
                                 */
{
/*
 *  Set global flag based on the user's selection.
 */

    flag = (int)closure;

    return;
}
```

```
/*****************************************************************************
 * MODULE NAME: tui_start_wait
 *
 *  This function changes the cursor to an watch to indicate that a time-consuming
 *  operation is about to take place. The watch cursor will appear in all windows
 *  making up the application.
 *
 *
 * EXTERNAL FUNCTIONS:
 *
 *  o   tui_stop_wait       -   This function restores the cursor to the default.
 *
 *
 * INTERNAL FUNCTIONS:
 *
 *  o   cb_destroy_shell    -   This function is called whenever a shell is des-
 *                              troyed.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/cursorfont.h>
#include <constants.h>
#include <disp.h>


extern Widget               Top;            /* Top level widget of the main control
                                             * panel window.
                                             */

extern struct dm_shmemory   *Dm_Address;    /* Pointer to shared memory. Needed for
                                             * the shell of the current display
                                             * window.
                                             */

extern short                Disp_Num;       /* Index into the table of displays.
                                             */


Widget                      Cur_shell;      /* Maintains the current popup shell.
                                             * Needed to change the cursor in this
                                             * window.
                                             */


int tui_start_wait ( )
{
    static Cursor   cursor = NULL;

/*
 *  If called before any widgets have been initialized, return immediately.
 */

    if ( Top == NULL )
        return ( 0 );
```

```
/*
 *  If the watch cursor has not yet been defined, define it.
 */

    if ( cursor == NULL )
        cursor = XCreateFontCursor ( XtDisplay ( Top ), XC_watch );

/*
 *  Set the cursor on the top level, current popup, and display shells.
 */

    XDefineCursor ( XtDisplay ( Top ), XtWindow ( Top ), cursor );

    if ( Cur_shell )
        XDefineCursor ( XtDisplay ( Top ), XtWindow ( Cur_shell ), cursor );

    if ( Dm_Address->shell[Disp_Num] )
        XDefineCursor ( XtDisplay ( Top ), XtWindow ( Dm_Address->shell[Disp_Num] ),
                        cursor );

/*
 *  Synchronize the display to cause the new cursor to appear.
 */

    XSync ( XtDisplay ( Top ), FALSE );

/*
 *  Normal return.
 */

    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: tui_stop_wait
 *
 *  This function restores the default cursor on all shell windows. This function
 *  is called once the time-consuming operation is complete.
 *****************************************************************************/

int tui_stop_wait ( )
{
/*
 *  If called before any widgets have been initialized, return immediately.
 */

    if ( Top == NULL )
        return ( 0 );

/*
 *  Reset the cursor on the top level, current popup, and display shells.
 */

    XDefineCursor ( XtDisplay ( Top ), XtWindow ( Top ), None );

    if ( Cur_shell )
        XDefineCursor ( XtDisplay ( Top ), XtWindow ( Cur_shell ), None );

    if ( Dm_Address->shell[Disp_Num] )
        XDefineCursor ( XtDisplay ( Top ), XtWindow ( Dm_Address->shell[Disp_Num] ),
                        None );

/*
 *  Synchronize the display to cause the new cursor to appear.
 */

    XSync ( XtDisplay ( Top ), FALSE );

/*
 *  Normal return.
 */

    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( cb_destroy_shell )
 *
 *  This callback function is called whenever a shell widget is destroyed. This
 *  function clears the global variable (Cur_shell) which keeps track of the
 *  current shell.
 *****************************************************************************/

XtCallbackProc cb_destroy_shell ( widget, closure, calldata )

    Widget  widget;             /* Set to the widget which initiated this
                                 * callback function.
                                 */

    caddr_t closure,            /* Callback specific data.  This parameter
                                 * is set to button selected.
                                 */
            *calldata;          /* Specifies any callback-specific data the
                                 * widget needs to pass to the client.
                                 */
{
/*
 *  Clear the current shell variable.
 */

    Cur_shell = NULL;

/*
 *  Normal return.
 */

    return;
}
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Generic defaults.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

*topAttachment:             ATTACH_POSITION
*bottomAttachment:          ATTACH_POSITION
*leftAttachment:            ATTACH_POSITION
*rightAttachment:           ATTACH_POSITION
*fontList:                  -adobe-courier-bold-r-normal--14-140-75-75-m-90-iso8859-1
*topShadowColor:            white
*foreground:                black
*XmText.background:         lightblue
*XmList.background:         skyblue
*allowOverlap:              FALSE
*traversalOn:               TRUE

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Accelerators.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

*mp_file.Exit.accelerator:                              <Key>F1
*mp_file.Enable Message.accelerator:                    <Key>F2
*mp_file.Set Flight/Data.accelerator:                   <Key>F3
*mp_file.Screen Dump.accelerator:                       <Key>F4
*mp_disp.Select Display.accelerator:                    <Key>F5
*mp_disp.Remove Display.accelerator:                    <Key>F6
*mp_disp.Freeze Display.accelerator:                    <Key>F7
*mp_util.Enable PBIs.accelerator:                       <Key>F8
*mp_limits.List Limits.accelerator:                     <Key>F9
*mp_plot.List Plots.accelerator:                        <Key>F10

*mp_file.Exit.acceleratorText:                          F1
*mp_file.Enable Message.acceleratorText:                F2
*mp_file.Set Flight/Data.acceleratorText:               F3
*mp_file.Screen Dump.acceleratorText:                   F4
*mp_disp.Select Display.acceleratorText:                F5
*mp_disp.Remove Display.acceleratorText:                F6
*mp_disp.Freeze Display.acceleratorText:                F7
*mp_util.Enable PBIs.acceleratorText:                   F8
*mp_limits.List Limits.acceleratorText:                 F9
*mp_plot.List Plots.acceleratorText:                    F10

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Defaults for buttons.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

*OK.showAsDefault:                                      1
*OK.leftPosition:                                       9
*OK.rightPosition:                                      25
*OK.topPosition:                                        13
*OK.bottomPosition:                                     87
*Cancel.leftPosition:                                   43
*Cancel.rightPosition:                                  57
*Cancel.topPosition:                                    21
*Cancel.bottomPosition:                                 79
*Help.leftPosition:                                     76
*Help.rightPosition:                                    90
*Help.topPosition:                                      21
*Help.bottomPosition:                                   79

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Needed for the display window.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
*Display Window.allowShellResize:                    TRUE
*Display Window*scrollingPolicy:                     AUTOMATIC
*Display Window*scrollBarDisplayPolicy:              AS_NEEDED
*Display Window*borderWidth:                         0
*Display Window*scroll.width:                        400
*Display Window*scroll.height:                       400
*Display Window*draw.width:                          800
*Display Window*draw.height:                         800
*Display Window*draw.marginHeight:                   0
*Display Window*draw.marginWidth:                    0

###############################################################################
# Needed for the Help pop up.
###############################################################################

*Display Manager Help.minWidth:                      600
*Display Manager Help.minHeight:                     550
*Display Manager Help*textSW.leftPosition:           1
*Display Manager Help*textSW.rightPosition:          99
*Display Manager Help*textSW.topPosition:            1
*Display Manager Help*textSW.bottomPosition:         90

*Display Manager Help*sep.leftPosition:              0
*Display Manager Help*sep.rightPosition:             100
*Display Manager Help*sep.topPosition:               91
*Display Manager Help*sep.bottomPosition:            92
*Display Manager Help*Done.leftPosition:             1
*Display Manager Help*Done.rightPosition:            25
*Display Manager Help*Done.topPosition:              93
*Display Manager Help*Done.bottomPosition:           99

###############################################################################
# Needed for the Set Flight ID/Data Type pop up.
###############################################################################

*Set Flight/Data.minWidth:                           500
*Set Flight/Data.minHeight:                          130
*Set Flight/Data*f_data.leftPosition:                1
*Set Flight/Data*f_data.rightPosition:               99
*Set Flight/Data*f_data.topPosition:                 1
*Set Flight/Data*f_data.bottomPosition:              59
*Set Flight/Data*sep0.leftPosition:                  0
*Set Flight/Data*sep0.rightPosition:                 100
*Set Flight/Data*sep0.topPosition:                   60
*Set Flight/Data*sep0.bottomPosition:                62
*Set Flight/Data*f_cmd.leftPosition:                 1
*Set Flight/Data*f_cmd.rightPosition:                99
*Set Flight/Data*f_cmd.topPosition:                  63
*Set Flight/Data*f_cmd.bottomPosition:               99

*Set Flight/Data*r_data.numColumns:                  2
*Set Flight/Data*l_fid.leftPosition:                 1
*Set Flight/Data*l_fid.rightPosition:                20
*Set Flight/Data*l_fid.topPosition:                  1
*Set Flight/Data*l_fid.bottomPosition:               40
*Set Flight/Data*t_fid.leftPosition:                 21
*Set Flight/Data*t_fid.rightPosition:                49
*Set Flight/Data*t_fid.topPosition:                  1
*Set Flight/Data*t_fid.bottomPosition:               40
*Set Flight/Data*l_data.leftPosition:                51
*Set Flight/Data*l_data.rightPosition:               70
*Set Flight/Data*l_data.topPosition:                 1
*Set Flight/Data*l_data.bottomPosition:              40
*Set Flight/Data*r_data.leftPosition:                71
```

```
*Set Flight/Data*r_data.rightPosition:                99
*Set Flight/Data*r_data.topPosition:                  1
*Set Flight/Data*r_data.bottomPosition:               99

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
# Needed for the Color Editor.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

*Color Editor.minHeight:                              465
*Color Editor*form*scrollingPolicy:                   AUTOMATIC
*Color Editor*form*scrollBarDisplayPolicy:            AS_NEEDED
*Color Editor*XmScale.orientation:                    HORIZONTAL
*Color Editor*XmScale.minimum:                        0
*Color Editor*XmScale.maximum:                        255
*Color Editor*XmScale.processingDirection:            MAX_ON_RIGHT
*Color Editor*f_clr.leftPosition:                     31
*Color Editor*f_clr.rightPosition:                    99
*Color Editor*f_clr.topPosition:                      1
*Color Editor*f_clr.bottomPosition:                   88
*Color Editor*sep.leftPosition:                       0
*Color Editor*sep.rightPosition:                      100
*Color Editor*sep.topPosition:                        89
*Color Editor*sep.bottomPosition:                     90
*Color Editor*f_cmd.leftPosition:                     1
*Color Editor*f_cmd.rightPosition:                    99
*Color Editor*f_cmd.topPosition:                      91
*Color Editor*f_cmd.bottomPosition:                   99

*Color Editor*l_colors.leftPosition:                  1
*Color Editor*l_colors.rightPosition:                 99
*Color Editor*l_colors.topPosition:                   1
*Color Editor*l_colors.bottomPosition:                6

*Color Editor*sw_colors.leftPosition:                 1
*Color Editor*sw_colors.rightPosition:                99
*Color Editor*sw_colors.topPosition:                  7
*Color Editor*sw_colors.bottomPosition:               80

*Color Editor*f_rgb.leftPosition:                     1
*Color Editor*f_rgb.rightPosition:                    99
*Color Editor*f_rgb.topPosition:                      81
*Color Editor*f_rgb.bottomPosition:                   99

*Color Editor*l_rgb_red.leftPosition:                 1
*Color Editor*l_rgb_red.rightPosition:                20
*Color Editor*l_rgb_red.topPosition:                  1
*Color Editor*l_rgb_red.bottomPosition:               32
*Color Editor*l_rgb_green.leftPosition:               1
*Color Editor*l_rgb_green.rightPosition:              20
*Color Editor*l_rgb_green.topPosition:                34
*Color Editor*l_rgb_green.bottomPosition:             65
*Color Editor*l_rgb_blue.leftPosition:                1
*Color Editor*l_rgb_blue.rightPosition:               20
*Color Editor*l_rgb_blue.topPosition:                 67
*Color Editor*l_rgb_blue.bottomPosition:              98

*Color Editor*sc_rgb_red.leftPosition:                21
*Color Editor*sc_rgb_red.rightPosition:               99
*Color Editor*sc_rgb_red.topPosition:                 1
*Color Editor*sc_rgb_red.bottomPosition:              32
*Color Editor*sc_rgb_green.leftPosition:              21
*Color Editor*sc_rgb_green.rightPosition:             99
*Color Editor*sc_rgb_green.topPosition:               34
*Color Editor*sc_rgb_green.bottomPosition:            65
```

```
*Color Editor*sc_rgb_blue.leftPosition:                21
*Color Editor*sc_rgb_blue.rightPosition:               99
*Color Editor*sc_rgb_blue.topPosition:                 67
*Color Editor*sc_rgb_blue.bottomPosition:              98

*Color Editor*OK.showAsDefault:                        1
*Color Editor*OK.leftPosition:                         3
*Color Editor*OK.rightPosition:                        25
*Color Editor*OK.topPosition:                          11
*Color Editor*OK.bottomPosition:                       89
*Color Editor*Cancel.leftPosition:                     27
*Color Editor*Cancel.rightPosition:                    49
*Color Editor*Cancel.topPosition:                      19
*Color Editor*Cancel.bottomPosition:                   81
*Color Editor*Restore.leftPosition:                    51
*Color Editor*Restore.rightPosition:                   73
*Color Editor*Restore.topPosition:                     19
*Color Editor*Restore.bottomPosition:                  81
*Color Editor*Help.leftPosition:                       75
*Color Editor*Help.rightPosition:                      97
*Color Editor*Help.topPosition:                        19
*Color Editor*Help.bottomPosition:                     81

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
# Needed for the Change Update Rate popup.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

*Change Update Rate*showValue:                         TRUE
*Change Update Rate*scale.orientation:                 HORIZONTAL
*Change Update Rate*scale.processingDirection:         MAX_ON_RIGHT
*Change Update Rate*minWidth:                          430
*Change Update Rate*minHeight:                         120
*Change Update Rate*f_data.leftPosition:               1
*Change Update Rate*f_data.rightPosition:              99
*Change Update Rate*f_data.topPosition:                1
*Change Update Rate*f_data.bottomPosition:             59
*Change Update Rate*sep0.leftPosition:                 0
*Change Update Rate*sep0.rightPosition:                100
*Change Update Rate*sep0.topPosition:                  60
*Change Update Rate*sep0.bottomPosition:               64
*Change Update Rate*f_cmd.leftPosition:                1
*Change Update Rate*f_cmd.rightPosition:               99
*Change Update Rate*f_cmd.topPosition:                 65
*Change Update Rate*f_cmd.bottomPosition:              99

*Change Update Rate*label.leftPosition:                1
*Change Update Rate*label.rightPosition:               99
*Change Update Rate*label.topPosition:                 1
*Change Update Rate*label.bottomPosition:              20
*Change Update Rate*scale.leftPosition:                1
*Change Update Rate*scale.rightPosition:               99
*Change Update Rate*scale.topPosition:                 21
*Change Update Rate*scale.bottomPosition:              99

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
# Needed for the Change Limits popup.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

*Change Limits*minWidth:                               400
*Change Limits*minHeight:                              470
*Change Limits*f_msid.leftPosition:                    1
*Change Limits*f_msid.rightPosition:                   99
*Change Limits*f_msid.topPosition:                     1
*Change Limits*f_msid.bottomPosition:                  48
```

```
*Change Limits*sep0.leftPosition:                0
*Change Limits*sep0.rightPosition:               100
*Change Limits*sep0.topPosition:                 49
*Change Limits*sep0.bottomPosition:              51
*Change Limits*f_data.leftPosition:              1
*Change Limits*f_data.rightPosition:             99
*Change Limits*f_data.topPosition:               52
*Change Limits*f_data.bottomPosition:            86
*Change Limits*sep1.leftPosition:                0
*Change Limits*sep1.rightPosition:               100
*Change Limits*sep1.topPosition:                 87
*Change Limits*sep1.bottomPosition:              88
*Change Limits*f_cmd.leftPosition:               1
*Change Limits*f_cmd.rightPosition:              99
*Change Limits*f_cmd.topPosition:                89
*Change Limits*f_cmd.bottomPosition:             99

*Change Limits*t_msid.leftPosition:              1
*Change Limits*t_msid.rightPosition:             40
*Change Limits*t_msid.topPosition:               1
*Change Limits*t_msid.bottomPosition:            99

*Change Limits*l_opslow.leftPosition:            1
*Change Limits*l_opslow.rightPosition:           20
*Change Limits*l_opslow.topPosition:             1
*Change Limits*l_opslow.bottomPosition:          20
*Change Limits*t_opslow.leftPosition:            21
*Change Limits*t_opslow.rightPosition:           50
*Change Limits*t_opslow.topPosition:             1
*Change Limits*t_opslow.bottomPosition:          20
*Change Limits*tg_opslow.leftPosition:           55
*Change Limits*tg_opslow.rightPosition:          73
*Change Limits*tg_opslow.topPosition:            1
*Change Limits*tg_opslow.bottomPosition:         20
*Change Limits*tg_opslow_a.leftPosition:         75
*Change Limits*tg_opslow_a.rightPosition:        99
*Change Limits*tg_opslow_a.topPosition:          1
*Change Limits*tg_opslow_a.bottomPosition:       20

*Change Limits*l_opshigh.leftPosition:           1
*Change Limits*l_opshigh.rightPosition:          20
*Change Limits*l_opshigh.topPosition:            27
*Change Limits*l_opshigh.bottomPosition:         46
*Change Limits*t_opshigh.leftPosition:           21
*Change Limits*t_opshigh.rightPosition:          50
*Change Limits*t_opshigh.topPosition:            27
*Change Limits*t_opshigh.bottomPosition:         46
*Change Limits*tg_opshigh.leftPosition:          55
*Change Limits*tg_opshigh.rightPosition:         73
*Change Limits*tg_opshigh.topPosition:           27
*Change Limits*tg_opshigh.bottomPosition:        46
*Change Limits*tg_opshigh_a.leftPosition:        75
*Change Limits*tg_opshigh_a.rightPosition:       99
*Change Limits*tg_opshigh_a.topPosition:         27
*Change Limits*tg_opshigh_a.bottomPosition:      46

*Change Limits*l_critlow.leftPosition:           1
*Change Limits*l_critlow.rightPosition:          20
*Change Limits*l_critlow.topPosition:            53
*Change Limits*l_critlow.bottomPosition:         72
*Change Limits*t_critlow.leftPosition:           21
*Change Limits*t_critlow.rightPosition:          50
*Change Limits*t_critlow.topPosition:            53
*Change Limits*t_critlow.bottomPosition:         72
```

```
*Change Limits*tg_critlow.leftPosition:                55
*Change Limits*tg_critlow.rightPosition:               73
*Change Limits*tg_critlow.topPosition:                 53
*Change Limits*tg_critlow.bottomPosition:              72
*Change Limits*tg_critlow_a.leftPosition:              75
*Change Limits*tg_critlow_a.rightPosition:             99
*Change Limits*tg_critlow_a.topPosition:               53
*Change Limits*tg_critlow_a.bottomPosition:            72

*Change Limits*l_crithigh.leftPosition:                1
*Change Limits*l_crithigh.rightPosition:               20
*Change Limits*l_crithigh.topPosition:                 79
*Change Limits*l_crithigh.bottomPosition:              98
*Change Limits*t_crithigh.leftPosition:                21
*Change Limits*t_crithigh.rightPosition:               50
*Change Limits*t_crithigh.topPosition:                 79
*Change Limits*t_crithigh.bottomPosition:              98
*Change Limits*tg_crithigh.leftPosition:               55
*Change Limits*tg_crithigh.rightPosition:              73
*Change Limits*tg_crithigh.topPosition:                79
*Change Limits*tg_crithigh.bottomPosition:             98
*Change Limits*tg_crithigh_a.leftPosition:             75
*Change Limits*tg_crithigh_a.rightPosition:            99
*Change Limits*tg_crithigh_a.topPosition:              79
*Change Limits*tg_crithigh_a.bottomPosition:           98

*Change Limits*Done.showAsDefault:                     1
*Change Limits*Done.leftPosition:                      8
*Change Limits*Done.rightPosition:                     20
*Change Limits*Done.topPosition:                       13
*Change Limits*Done.bottomPosition:                    87
*Change Limits*Save.leftPosition:                      32
*Change Limits*Save.rightPosition:                     44
*Change Limits*Save.topPosition:                       21
*Change Limits*Save.bottomPosition:                    79
*Change Limits*MSID.leftPosition:                      56
*Change Limits*MSID.rightPosition:                     68
*Change Limits*MSID.topPosition:                       21
*Change Limits*MSID.bottomPosition:                    79
*Change Limits*Help.leftPosition:                      80
*Change Limits*Help.rightPosition:                     92
*Change Limits*Help.topPosition:                       21
*Change Limits*Help.bottomPosition:                    79

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
# Needed for the Change DDD MSID popup.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

*Change DDD MSID*minWidth:                             400
*Change DDD MSID*minHeight:                            270
*Change DDD MSID*f_msid.leftPosition:                  1
*Change DDD MSID*f_msid.rightPosition:                 99
*Change DDD MSID*f_msid.topPosition:                   1
*Change DDD MSID*f_msid.bottomPosition:                80
*Change DDD MSID*sep0.leftPosition:                    0
*Change DDD MSID*sep0.rightPosition:                   100
*Change DDD MSID*sep0.topPosition:                     81
*Change DDD MSID*sep0.bottomPosition:                  82
*Change DDD MSID*f_cmd.leftPosition:                   1
*Change DDD MSID*f_cmd.rightPosition:                  99
*Change DDD MSID*f_cmd.topPosition:                    83
*Change DDD MSID*f_cmd.bottomPosition:                 99

*Change DDD MSID*t_msid.leftPosition:                  1
```

```
*Change DDD MSID*t_msid.rightPosition:                    40
*Change DDD MSID*t_msid.topPosition:                      1
*Change DDD MSID*t_msid.bottomPosition:                   99

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
# Needed for the Change GDR popup.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

*Change GDR*r_dur_units.numColumns:                       2
*Change GDR*r_ppl_units.numColumns:                       2
*Change GDR*minWidth:                                     600
*Change GDR*minHeight:                                    200
*Change GDR*f_ppl.leftPosition:                           1
*Change GDR*f_ppl.rightPosition:                          99
*Change GDR*f_ppl.topPosition:                            1
*Change GDR*f_ppl.bottomPosition:                         15
*Change GDR*sep0.leftPosition:                            0
*Change GDR*sep0.rightPosition:                           100
*Change GDR*sep0.topPosition:                             16
*Change GDR*sep0.bottomPosition:                          17
*Change GDR*f_data.leftPosition:                          1
*Change GDR*f_data.rightPosition:                         99
*Change GDR*f_data.topPosition:                           18
*Change GDR*f_data.bottomPosition:                        76
*Change GDR*sep1.leftPosition:                            0
*Change GDR*sep1.rightPosition:                           100
*Change GDR*sep1.topPosition:                             77
*Change GDR*sep1.bottomPosition:                          78
*Change GDR*f_cmd.leftPosition:                           1
*Change GDR*f_cmd.rightPosition:                          99
*Change GDR*f_cmd.topPosition:                            79
*Change GDR*f_cmd.bottomPosition:                         99

*Change GDR*l_ppl.leftPosition:                           1
*Change GDR*l_ppl.rightPosition:                          30
*Change GDR*l_ppl.topPosition:                            1
*Change GDR*l_ppl.bottomPosition:                         99
*Change GDR*t_ppl.leftPosition:                           31
*Change GDR*t_ppl.rightPosition:                          95
*Change GDR*t_ppl.topPosition:                            1
*Change GDR*t_ppl.bottomPosition:                         99

*Change GDR*l_host.leftPosition:                          1
*Change GDR*l_host.rightPosition:                         30
*Change GDR*l_host.topPosition:                           1
*Change GDR*l_host.bottomPosition:                        25
*Change GDR*t_host.leftPosition:                          31
*Change GDR*t_host.rightPosition:                         45
*Change GDR*t_host.topPosition:                           1
*Change GDR*t_host.bottomPosition:                        25
*Change GDR*l_rq.leftPosition:                            51
*Change GDR*l_rq.rightPosition:                           80
*Change GDR*l_rq.topPosition:                             1
*Change GDR*l_rq.bottomPosition:                          25
*Change GDR*t_rq.leftPosition:                            81
*Change GDR*t_rq.rightPosition:                           95
*Change GDR*t_rq.topPosition:                             1
*Change GDR*t_rq.bottomPosition:                          25

*Change GDR*l_duration.leftPosition:                      1
*Change GDR*l_duration.rightPosition:                     30
*Change GDR*l_duration.topPosition:                       30
*Change GDR*l_duration.bottomPosition:                    54
*Change GDR*t_duration.leftPosition:                      31
```

```
*Change GDR*t_duration.rightPosition:                45
*Change GDR*t_duration.topPosition:                  30
*Change GDR*t_duration.bottomPosition:               54
*Change GDR*l_ppl_rate.leftPosition:                 51
*Change GDR*l_ppl_rate.rightPosition:                80
*Change GDR*l_ppl_rate.topPosition:                  30
*Change GDR*l_ppl_rate.bottomPosition:               54
*Change GDR*t_ppl_rate.leftPosition:                 81
*Change GDR*t_ppl_rate.rightPosition:                95
*Change GDR*t_ppl_rate.topPosition:                  30
*Change GDR*t_ppl_rate.bottomPosition:               54

*Change GDR*l_dur_units.leftPosition:                1
*Change GDR*l_dur_units.rightPosition:               30
*Change GDR*l_dur_units.topPosition:                 60
*Change GDR*l_dur_units.bottomPosition:              99
*Change GDR*r_dur_units.leftPosition:                31
*Change GDR*r_dur_units.rightPosition:               45
*Change GDR*r_dur_units.topPosition:                 60
*Change GDR*r_dur_units.bottomPosition:              99
*Change GDR*l_ppl_units.leftPosition:                51
*Change GDR*l_ppl_units.rightPosition:               80
*Change GDR*l_ppl_units.topPosition:                 60
*Change GDR*l_ppl_units.bottomPosition:              99
*Change GDR*r_ppl_units.leftPosition:                81
*Change GDR*r_ppl_units.rightPosition:               95
*Change GDR*r_ppl_units.topPosition:                 60
*Change GDR*r_ppl_units.bottomPosition:              99

*Change GDR*OK.leftPosition:                         8
*Change GDR*OK.rightPosition:                        20
*Change GDR*OK.topPosition:                          13
*Change GDR*OK.bottomPosition:                       87
*Change GDR*PPL.leftPosition:                        32
*Change GDR*PPL.rightPosition:                       44
*Change GDR*PPL.topPosition:                         21
*Change GDR*PPL.bottomPosition:                      79
*Change GDR*Cancel.leftPosition:                     56
*Change GDR*Cancel.rightPosition:                    68
*Change GDR*Cancel.topPosition:                      21
*Change GDR*Cancel.bottomPosition:                   79
*Change GDR*Help.leftPosition:                       80
*Change GDR*Help.rightPosition:                      92
*Change GDR*Help.topPosition:                        21
*Change GDR*Help.bottomPosition:                     79

#############################################################################
# Needed for the Plot/Save overlay popup.
#############################################################################

*Plot/Save Overlay*minWidth:                         500
*Plot/Save Overlay*minHeight:                        270
*Plot/Save Overlay*f_data.leftPosition:              1
*Plot/Save Overlay*f_data.rightPosition:             99
*Plot/Save Overlay*f_data.topPosition:               1
*Plot/Save Overlay*f_data.bottomPosition:            80
*Plot/Save Overlay*sep0.leftPosition:                0
*Plot/Save Overlay*sep0.rightPosition:               100
*Plot/Save Overlay*sep0.topPosition:                 81
*Plot/Save Overlay*sep0.bottomPosition:              82
*Plot/Save Overlay*f_cmd.leftPosition:               1
*Plot/Save Overlay*f_cmd.rightPosition:              99
*Plot/Save Overlay*f_cmd.topPosition:                83
*Plot/Save Overlay*f_cmd.bottomPosition:             99
```

```
*Plot/Save Overlay*t_plot.leftPosition:                3
*Plot/Save Overlay*t_plot.rightPosition:               47
*Plot/Save Overlay*t_plot.topPosition:                 1
*Plot/Save Overlay*t_plot.bottomPosition:              99
*Plot/Save Overlay*t_ovl.leftPosition:                 53
*Plot/Save Overlay*t_ovl.rightPosition:                97
*Plot/Save Overlay*t_ovl.topPosition:                  1
*Plot/Save Overlay*t_ovl.bottomPosition:               99

##########################################################################################
# Needed for the Define Universal plot popup.
##########################################################################################

*Define Universal Plot*minWidth:                       500
*Define Universal Plot*minHeight:                      600
*Define Universal Plot*f_plot.leftPosition:            1
*Define Universal Plot*f_plot.rightPosition:           95
*Define Universal Plot*f_plot.topPosition:             1
*Define Universal Plot*f_plot.bottomPosition:          6
*Define Universal Plot*sep0.leftPosition:              0
*Define Universal Plot*sep0.rightPosition:             100
*Define Universal Plot*sep0.topPosition:               7
*Define Universal Plot*sep0.bottomPosition:            8
*Define Universal Plot*f_xy.leftPosition:              1
*Define Universal Plot*f_xy.rightPosition:             99
*Define Universal Plot*f_xy.topPosition:               9
*Define Universal Plot*f_xy.bottomPosition:            26
*Define Universal Plot*sep1.leftPosition:              0
*Define Universal Plot*sep1.rightPosition:             100
*Define Universal Plot*sep1.topPosition:               27
*Define Universal Plot*sep1.bottomPosition:            28
*Define Universal Plot*f_msid.leftPosition:            1
*Define Universal Plot*f_msid.rightPosition:           99
*Define Universal Plot*f_msid.topPosition:             29
*Define Universal Plot*f_msid.bottomPosition:          90
*Define Universal Plot*sep2.leftPosition:              0
*Define Universal Plot*sep2.rightPosition:             100
*Define Universal Plot*sep2.topPosition:               91
*Define Universal Plot*sep2.bottomPosition:            92
*Define Universal Plot*f_cmd.leftPosition:             1
*Define Universal Plot*f_cmd.rightPosition:            99
*Define Universal Plot*f_cmd.topPosition:              92
*Define Universal Plot*f_cmd.bottomPosition:           99

*Define Universal Plot*l_plot.leftPosition:            1
*Define Universal Plot*l_plot.rightPosition:           25
*Define Universal Plot*l_plot.topPosition:             1
*Define Universal Plot*l_plot.bottomPosition:          99
*Define Universal Plot*t_plot.leftPosition:            26
*Define Universal Plot*t_plot.rightPosition:           50
*Define Universal Plot*t_plot.topPosition:             1
*Define Universal Plot*t_plot.bottomPosition:          99

*Define Universal Plot*t_xy_id.leftPosition:           1
*Define Universal Plot*t_xy_id.rightPosition:          30
*Define Universal Plot*t_xy_id.topPosition:            1
*Define Universal Plot*t_xy_id.bottomPosition:         30

*Define Universal Plot*l_xlow.leftPosition:            1
*Define Universal Plot*l_xlow.rightPosition:           25
*Define Universal Plot*l_xlow.topPosition:             35
*Define Universal Plot*l_xlow.bottomPosition:          65
*Define Universal Plot*t_xlow.leftPosition:            26
```

```
*Define Universal Plot*t_xlow.rightPosition:          45
*Define Universal Plot*t_xlow.topPosition:            35
*Define Universal Plot*t_xlow.bottomPosition:         65
*Define Universal Plot*l_xhigh.leftPosition:          1
*Define Universal Plot*l_xhigh.rightPosition:         25
*Define Universal Plot*l_xhigh.topPosition:           70
*Define Universal Plot*l_xhigh.bottomPosition:        99
*Define Universal Plot*t_xhigh.leftPosition:          26
*Define Universal Plot*t_xhigh.rightPosition:         45
*Define Universal Plot*t_xhigh.topPosition:           70
*Define Universal Plot*t_xhigh.bottomPosition:        99

*Define Universal Plot*l_ylow.leftPosition:           51
*Define Universal Plot*l_ylow.rightPosition:          75
*Define Universal Plot*l_ylow.topPosition:            35
*Define Universal Plot*l_ylow.bottomPosition:         65
*Define Universal Plot*t_ylow.leftPosition:           76
*Define Universal Plot*t_ylow.rightPosition:          95
*Define Universal Plot*t_ylow.topPosition:            35
*Define Universal Plot*t_ylow.bottomPosition:         65
*Define Universal Plot*l_yhigh.leftPosition:          51
*Define Universal Plot*l_yhigh.rightPosition:         75
*Define Universal Plot*l_yhigh.topPosition:           70
*Define Universal Plot*l_yhigh.bottomPosition:        99
*Define Universal Plot*t_yhigh.leftPosition:          76
*Define Universal Plot*t_yhigh.rightPosition:         95
*Define Universal Plot*t_yhigh.topPosition:           70
*Define Universal Plot*t_yhigh.bottomPosition:        99

*Define Universal Plot*t_msid_id.leftPosition:        1
*Define Universal Plot*t_msid_id.rightPosition:       30
*Define Universal Plot*t_msid_id.topPosition:         1
*Define Universal Plot*t_msid_id.bottomPosition:      9

*Define Universal Plot*l_msid.leftPosition:           1
*Define Universal Plot*l_msid.rightPosition:          25
*Define Universal Plot*l_msid.topPosition:            11
*Define Universal Plot*l_msid.bottomPosition:         19
*Define Universal Plot*t_msid.leftPosition:           26
*Define Universal Plot*t_msid.rightPosition:          45
*Define Universal Plot*t_msid.topPosition:            11
*Define Universal Plot*t_msid.bottomPosition:         19

*Define Universal Plot*l_msid_p.leftPosition:         51
*Define Universal Plot*l_msid_p.rightPosition:        75
*Define Universal Plot*l_msid_p.topPosition:          11
*Define Universal Plot*l_msid_p.bottomPosition:       19
*Define Universal Plot*t_msid_p.leftPosition:         76
*Define Universal Plot*t_msid_p.rightPosition:        95
*Define Universal Plot*t_msid_p.topPosition:          11
*Define Universal Plot*t_msid_p.bottomPosition:       19

*Define Universal Plot*l_src.leftPosition:            1
*Define Universal Plot*l_src.rightPosition:           25
*Define Universal Plot*l_src.topPosition:             21
*Define Universal Plot*l_src.bottomPosition:          29
*Define Universal Plot*t_src.leftPosition:            26
*Define Universal Plot*t_src.rightPosition:           45
*Define Universal Plot*t_src.topPosition:             21
*Define Universal Plot*t_src.bottomPosition:          29

*Define Universal Plot*l_src_p.leftPosition:          51
*Define Universal Plot*l_src_p.rightPosition:         75
*Define Universal Plot*l_src_p.topPosition:           21
```

```
*Define Universal Plot*l_src_p.bottomPosition:          29
*Define Universal Plot*t_src_p.leftPosition:            76
*Define Universal Plot*t_src_p.rightPosition:           95
*Define Universal Plot*t_src_p.topPosition:             21
*Define Universal Plot*t_src_p.bottomPosition:          29

*Define Universal Plot*l_sample.leftPosition:           1
*Define Universal Plot*l_sample.rightPosition:          25
*Define Universal Plot*l_sample.topPosition:            31
*Define Universal Plot*l_sample.bottomPosition:         39
*Define Universal Plot*r_sample.leftPosition:           26
*Define Universal Plot*r_sample.rightPosition:          45
*Define Universal Plot*r_sample.topPosition:            31
*Define Universal Plot*r_sample.bottomPosition:         45

*Define Universal Plot*l_sample_p.leftPosition:         51
*Define Universal Plot*l_sample_p.rightPosition:        75
*Define Universal Plot*l_sample_p.topPosition:          31
*Define Universal Plot*l_sample_p.bottomPosition:       39
*Define Universal Plot*r_sample_p.leftPosition:         76
*Define Universal Plot*r_sample_p.rightPosition:        95
*Define Universal Plot*r_sample_p.topPosition:          31
*Define Universal Plot*r_sample_p.bottomPosition:       45

*Define Universal Plot*s_axis_no.listVisibleItemCount:  3
*Define Universal Plot*l_axis_no.leftPosition:          1
*Define Universal Plot*l_axis_no.rightPosition:         25
*Define Universal Plot*l_axis_no.topPosition:           47
*Define Universal Plot*l_axis_no.bottomPosition:        56
*Define Universal Plot*s_axis_no.leftPosition:          26
*Define Universal Plot*s_axis_no.rightPosition:         45
*Define Universal Plot*s_axis_no.topPosition:           47
*Define Universal Plot*s_axis_no.bottomPosition:        85

*Define Universal Plot*s_axis_no_p.listVisibleItemCount:3
*Define Universal Plot*l_axis_no_p.leftPosition:        51
*Define Universal Plot*l_axis_no_p.rightPosition:       75
*Define Universal Plot*l_axis_no_p.topPosition:         47
*Define Universal Plot*l_axis_no_p.bottomPosition:      56
*Define Universal Plot*s_axis_no_p.leftPosition:        76
*Define Universal Plot*s_axis_no_p.rightPosition:       95
*Define Universal Plot*s_axis_no_p.topPosition:         47
*Define Universal Plot*s_axis_no_p.bottomPosition:      85

*Define Universal Plot*l_xory.leftPosition:             1
*Define Universal Plot*l_xory.rightPosition:            25
*Define Universal Plot*l_xory.topPosition:              87
*Define Universal Plot*l_xory.bottomPosition:           96
*Define Universal Plot*r_xory.leftPosition:             26
*Define Universal Plot*r_xory.rightPosition:            45
*Define Universal Plot*r_xory.topPosition:              87
*Define Universal Plot*r_xory.bottomPosition:           99

*Define Universal Plot*OK.leftPosition:                 5
*Define Universal Plot*OK.rightPosition:                17
*Define Universal Plot*OK.topPosition:                  13
*Define Universal Plot*OK.bottomPosition:               87
*Define Universal Plot*Plot.leftPosition:               21
*Define Universal Plot*Plot.rightPosition:              33
*Define Universal Plot*Plot.topPosition:                21
*Define Universal Plot*Plot.bottomPosition:             79
*Define Universal Plot*Axis.leftPosition:               37
*Define Universal Plot*Axis.rightPosition:              49
*Define Universal Plot*Axis.topPosition:                21
```

```
*Define Universal Plot*Axis.bottomPosition:          79
*Define Universal Plot*MSID.leftPosition:            53
*Define Universal Plot*MSID.rightPosition:           65
*Define Universal Plot*MSID.topPosition:             21
*Define Universal Plot*MSID.bottomPosition:          79
*Define Universal Plot*Cancel.leftPosition:          69
*Define Universal Plot*Cancel.rightPosition:         81
*Define Universal Plot*Cancel.topPosition:           21
*Define Universal Plot*Cancel.bottomPosition:        79
*Define Universal Plot*Help.leftPosition:            85
*Define Universal Plot*Help.rightPosition:           97
*Define Universal Plot*Help.topPosition:             21
*Define Universal Plot*Help.bottomPosition:          79

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
# Needed for the Change Zoom Factor popup.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

*Change Zoom Factor*showValue:                       TRUE
*Change Zoom Factor*scale.orientation:               HORIZONTAL
*Change Zoom Factor*scale.processingDirection:       MAX_ON_RIGHT
*Change Zoom Factor*scale.decimalPoints:             2
*Change Zoom Factor*minWidth:                        430
*Change Zoom Factor*minHeight:                       120
*Change Zoom Factor*f_data.leftPosition:             1
*Change Zoom Factor*f_data.rightPosition:            99
*Change Zoom Factor*f_data.topPosition:              1
*Change Zoom Factor*f_data.bottomPosition:           59
*Change Zoom Factor*sep0.leftPosition:               0
*Change Zoom Factor*sep0.rightPosition:              100
*Change Zoom Factor*sep0.topPosition:                60
*Change Zoom Factor*sep0.bottomPosition:             64
*Change Zoom Factor*f_cmd.leftPosition:              1
*Change Zoom Factor*f_cmd.rightPosition:             99
*Change Zoom Factor*f_cmd.topPosition:               65
*Change Zoom Factor*f_cmd.bottomPosition:            99

*Change Zoom Factor*label.leftPosition:              1
*Change Zoom Factor*label.rightPosition:             99
*Change Zoom Factor*label.topPosition:               1
*Change Zoom Factor*label.bottomPosition:            20
*Change Zoom Factor*scale.leftPosition:              1
*Change Zoom Factor*scale.rightPosition:             99
*Change Zoom Factor*scale.topPosition:               21
*Change Zoom Factor*scale.bottomPosition:            99
```

GDR CHANGE RETRIEVAL

This function allows the user to change the Generalized Data Retrieval source on the workstation. The user specifies a PPL filename and then is allowed to make updates.

CHANGE LIMIT VALUES

This function allows the user to change limit sense values for an MSID in real-time. When a limit is changed, the new limit is effective for all displays within the same workstation, flight, data type, and position ID.

DEFINE UNIVERSAL PLOT

This function allows the user to define a universal plot. This enables the user to define in real-time certain fields in a plot definition file that were build during display build time.

DISPLAY OVERLAY

This function allows the user to display an overlay file.

EDIT COLORS

This function presents a grid which represents the color map used by the Display Manager.

The user can both review the colors used and make changes if desired. The changes will only affect the colors within the Display Manager. The changes will not affect other clients.

To change a color, first select the appropriate rectangle in the grid of colors. The Red/Green/Blue scales at the bottom of the window will change to reflect the makeup of the color. Next adjust the color by positioning the scales to the left or to the right. Moving the scales to the left reduces the intensity of the Red/Green/Blue component in the color; moving the scales to the right increases the intensity.

To save changes use the OK button. To cancel changes, select CANCEL. The RESTORE button can be used to restore the color map without exiting from the window.

ENABLE/DISABLE ALARMS

This toggle function allows the user to enable and disable alarms.

ENABLE/DISABLE ALL LOGGING

This function allows the user to enable and disable use of all log files.

ENABLE/DISABLE LOGGING

This function allows the user to enable and disable use of a log file.

ENABLE/DISABLE MESSAGES

This toggle function allows the user to enable or disable the display of popup messages.

All messages generated by the Display Manager are routed to the Advisory system. If popup messages are enabled, all messages will also be displayed via a popup window. A popup message window will remain displayed until removed by the user. The popup will not lock out input for other windows.

Only one popup message will be displayed at a time. If a popup message is not removed and another message is pending, the first will be replaced by the new popup.

ENABLE/DISABLE PBI'S

This toggle function allows the user to enable or disable input to defined PBI's.

EXIT

This function terminates the current display and causes the Display Manager to exit. This action will not halt other displays running on different IGP's.

SET FLIGHT ID AND DATA TYPE

This function allows the user to select the flight ID and the data stream type.

The flight ID is any valid flight active on the workstation. The data stream type is any of the valid types displayed in the forms radio box.

FREEZE/RESTART DISPLAY

This toggle function allows the user to freeze or restart the current display.

SHOW FUNCTION KEYS

This function displays a menu which lists all the defined function keys.

LIST LIMITS

This function lists all available limit files and their current status and allows the user to start or stop a limit group (file).

LIST PLOTS

This function lists all available plot files and their current status and allows the user to start or stop a plot.

REMOVE DISPLAY

This function removes the current display.

The display is removed and the user is allowed to select another display or use functions which do not require a display to be initialized. Note that this function does not cause the Display Manager to terminate.

RESET ZOOM

This function allows the user to set the zoom factor back to the default of 1.0

SAVE DISPLAY OVERLAY

This function allows the user to save a plot data file as a display overlay file to be used at some future date.

SCREENDUMP

This function is used to generate a dump of the current screen.

The contents will be dumped to the hardcopy device defined for the workstation. Note that for best results, first freeze the display with the desired information and then select the screen dump function. This is not required, but will yield better results.

SELECT DISPLAY

This function presents a list of the available display files and allows a display to be selected.

The selected display will be started and will replace any display already initialized. This is the normal manner of initializing a display.

Note that the list of files contained in this window is saved during the first use. Subsequent uses of this window will require much less initialization time. Note also that if a new display is added after this list is build, the display will not appear in the list unless the user exits and restarts the Display Manager.

SET ZOOM

This function allows the user to set the zoom factor to a new value.

UNLATCH ALL DDD's

This function allows the user to unlatch all Digital Display Driver (DDD) MSID's.

UNLATCH DDD MSID

This function allows the user to unlatch a specific Digital Display Driver (DDD) MSID.

UPDATE RATE

This function allows the user to set the display update rate.

ZOOM

This function allows the user to select the area which will be zoomed.

ATTACHMENT 2 - Data Handler Stub Code

```
/*****************************************************************************
 * ds_connect -     STUB VERSION.  Returns the source # in the env_key for
 *                  use by ds_getkeys to open the proper mdef file.
 *
 * RLK 8/27/90
 *****************************************************************************/

/*****************************************************************************
 *  Include files
 *****************************************************************************/

#include <stdio.h>
#include <ds/ds.h>
#include <ds_stub.h>
#include <FCcplus.h>                    /* defined constants */
#include <FCdebug.h>                    /* debug macros */
#include <wex/wex.h>


/*****************************************************************************
 *  Externals
 *****************************************************************************/

extern  int errno;


/*****************************************************************************
 *  Function
 *****************************************************************************/

int ds_connect(source,option)
    char *source;               /* data source for stream */
    char *option[];             /* list of stream options */
{
D(fprintf(stderr, "ds_connect: source = %s, flight = %s, stream type = %s, options = %s\n"
, source, option[0], option[1], option[2]));


    if (IS_EVN(source))
        return(EVN);

    if (IS_MTM(source))
        return(MTM);

/*
    if (IS_GDR(source)) {
        fprintf(stderr, "ds_connect: GDR source not supported\n");
        errno = E_DSINVSRC;
        return(INVALID);
    }

    if (IS_USR(source)) {
        fprintf(stderr, "ds_connect: USR source not supported\n");
        errno = E_DSINVSRC;
        return(INVALID);
    }
*/

    if (IS_GDR(source))
        return(GDR);

    if (IS_USR(source))
        return(USR);
```

```
        if (IS_NDM(source))
            return(NDM);

        if (IS_PTM(source))
            return(PTM);

        if (IS_PPM(source))
            return(PPM);

        fprintf(stderr, "ds_connect: unknown source not supported\n");
        errno = E_DSINVSRC;
        return(INVALID);

    }
```

```
#include <stdio.h>

int ds_discon(env_key)
    int env_key;
{
    return(0);
}
```

```
/*****************************************************************************
 * ds_getkeys -     STUB VERSION.
 *                  (1) create decom entry for each msid
 *                  (2) create a keylist of data sizes in bytes
 *                  (3) set key_count to #msids
 *
 *      The decom entry will be generated in the following way:
 *
 *          length - (bits) read from msid.def
 *          size - (bytes)  (length + status (32 bits))/8 (if numeric)
 *                          length + 4 (if text)
 *          offset - accumulated as msids are processed, offset += size
 *          num_samps - set to 1 (check to assure <= msid_list->sample_cnt)
 *          error - may be set if requested sample too large, etc
 *          attribute - from msid.def
 *
 *      The keylist will have 3 entries per msid:
 *          length of one sample in bytes (decom->size - 4)
 *          low scale value     These values used by ds_getparms
 *          high scale value    to try to generate meaningful random data.
 *
 *      The data will be generated in the following way:
 *
 *          For each numeric msid, a random # will be calculated
 *          which is <= the maximum value representable in the # of
 *          bits in the length.
 *
 *          For textual msids, a random index into an array of strings
 *          (defined in #define) will be generated, and <length> chars
 *          copied to the file.
 *
 * This program is part of the Data Acquisition stub for DB/DM enhancement
 * at SwRI.
 *
 * Ronnie Killough 8/28/90
 *
 *****************************************************************************/

/*****************************************************************************
 *  Include files
 *****************************************************************************/

#include <stdio.h>
#include <ds/ds.h>
#include <ds_stub.h>
#include <FCcplus.h>                    /* defined constants */
/* #include <FCdebug.h>                    /* debug macros */
#include <wex/wex.h>

/*****************************************************************************
 *  Globals
 *****************************************************************************/

extern  int errno;

/*****************************************************************************
 *  Functions
 *****************************************************************************/

int ds_getkeys(environment_key, input_msids, meta_keylist, meta_decom,
               meta_data_values)

int environment_key;        /* Environment key returned from ds_connect */
p_input input_msids;        /* File or buffer input list of MSIDs */
```

```
int **meta_keylist;          /* Place to store pointer to key list */
p_decom **meta_decom;        /* Place to store pointer to decom table */
char **meta_data_values;     /* Place to store pointer to data value buffer*/
{
    FILE *tp;
        /* pointer to MSID files */
    FILE *mdp;
        /* ptr to mdef file */

    parm_ent *msid_list, *input_msid_list;
        /* pointer to msid parm_ent list */
    p_decom *decom;
        /* local ptr to decom tables */
    struct mdef_node md[MAX_DEF];
        /* list of msid.def records */

    int num_md;
        /* number of msid.def entries */
    int key_count;
        /* num msids for this source */
    int *keylist;
        /* list of data widths for each msid */
    int i, j;
        /* local temps */
    int offset;
        /* local offset */
    int found;
        /* boolean */

    char *data_values;
        /* local ptr to dv buffer */
    char tfn[15];
        /* diag filename */
    char fn[10];
        /* file name holder for mdef file */


    D(fprintf(stderr, "START ds_getkeys\n"));

/*
 *  Check for valid key
 */

    switch (environment_key) {
        case GDR:
            key_count = ds_rdppl(input_msids.parmin, &input_msid_list);
            break;
        case USR:
            fprintf(stderr, "ds_getkeys: invalid environment key %d\n", environment_key);
            return(-1);
            break;
        default:
            input_msid_list = (parm_ent *) input_msids.parmin;
            break;
    }

/*
 *  Output intro line to display
 */

   D(fprintf(stderr, "ds_getkeys: environment_key = %d, parmsrc = %c\n", environment_key,
 input_msids.parmsrc));

/*
```

```
 *  Output MSIDs to file for diagnostics
 */

#ifdef DEBUG

    sprintf(tfn, "%s/%s.%d", DATA_DIR, "msids", environment_key);
    if (!(tp = fopen(tfn, "w")))
        fprintf(stderr, "Unable to open diagnostic output file %s.\n",tfn);
    else {
        for (msid_list = input_msid_list; *(msid_list->name); msid_list++)
            fprintf(tp, "%s\n", msid_list->name);
        fclose(tp);
    }

#endif

/*
 *  Count the number of msids for key count
 */

    key_count = 0;

    for (msid_list = input_msid_list; *(msid_list->name); msid_list++)
        key_count++;

/*
 *  Allocate memory and copy pointers to calling parms
 */

    keylist = (int *) malloc((key_count * 3) * sizeof(int));
    decom = (p_decom *) malloc(key_count * sizeof(p_decom));

    if ((keylist == NULL) || (decom == NULL))
        return(-1);

    *meta_keylist = keylist;
    *meta_decom = decom;

/*
 *  Read in the MSID data file for this source
 */

    sprintf(fn, "%s/%s.%d", DATA_DIR, md_fn, environment_key);

    if (!(mdp = fopen(fn, "r"))) {
        fprintf(stderr, "Unable to open %s. Run mk_mdef on fg file.\n",fn);
        return(-1);
    }

    num_md = 0;
    fscanf(mdp, "%s %ld %d %d %d\n", md[num_md].msid, &(md[num_md].length), &(md[num_md].a
ttribute), &(md[num_md].low_scale), &(md[num_md].high_scale));
    num_md++;

    while (!feof(mdp) && num_md < MAX_DEF) {
        fscanf(mdp, "%s %ld %d %d %d\n", md[num_md].msid, &(md[num_md].length), &(md[num_m
d].attribute), &(md[num_md].low_scale), &(md[num_md].high_scale));
        num_md++;
    }

    num_md = num_md - 1;

    fclose(mdp);
```

```
/*
 *  Loop through MSIDs, find in mdef list, fill in decom record
 */

    /* Set up pointers, counters */

    offset = 0;
    msid_list = input_msid_list;

    /* Loop through */

    for (i=0; i<key_count; i++) {

        /* find this msid record */

        found = 0;
        for (j=0; j<=num_md; j++) {
            if (!strncmp(md[j].msid, msid_list->name, MSID_LEN)) {
                found = 1;
                break;
            }
        }

        if (!found && environment_key != GDR) {
            fprintf(stderr, "ds_getkeys: msid %s not found...run mk_mdef for this display.
\n", msid_list->name);
            sleep(5);
            return(-1);
        }

        /* create decom entry */

        decom->length = (int) md[j].length;
        decom->attribute = (char) md[j].attribute;

        if (md[j].attribute == 'A')              /* text...length is in bytes */
            decom->size = decom->length + 4;
        else                                     /* num...length is in bits */
            decom->size = (decom->length + 32) / 8;

        decom->num_samps = 1;
        if (decom->num_samps < msid_list->sample_cnt)
            decom->error = SAMP2LG;
        else
            decom->error = NULL;

        /* place data width and low/high scale values in the keylist */

        *keylist = decom->size - 4;      /* keylist = byte size of sample */
        keylist++;
        *keylist = md[j].low_scale;
        keylist++;
        *keylist = md[j].high_scale;

        /* copy offset to decom */
        decom->offset = offset;         /* offset is offset into dv buffer */
        offset += decom->size;

        /* increment the pointers */

        D(fprintf(stderr, "ds_getkeys: msid %s length %ld size %ld attribute %c key_count
%d\n", msid_list->name, decom->length, decom->size, decom->attribute, i));

        keylist++;
```

```
            decom++;
            msid_list++;
        }

    /*
     *  Allocate memory for the data value buffer and copy address to parm
     */

        data_values = (char *) malloc((unsigned int) offset);

        D(fprintf(stderr, "address is %ld\n", (unsigned long) data_values));

        *meta_data_values = data_values;

    /*
     *  Return key count
     */

        D(fprintf(stderr, "END ds_getkeys\n"));

        return(key_count);

    }
```

```
/*****************************************************************************
 * ds_getparms -    STUB VERSION.  Generates <key_count> data of length
 *                  <keylist> and places in the data value buffer.
 *
 *      The data will be generated in the following way:
 *
 *          For each numeric msid, a random # will be calculated
 *          which is <= the maximum value representable in the # of
 *          bits in the length.
 *
 *          For textual msids, a random index into an array of strings
 *          (defined in #define) will be generated, and <length> chars
 *          copied to the file.
 *
 * This program is part of the Data Acquisition stub for DB/DM enhancement
 * at SwRI.
 *
 * Ronnie Killough 8/28/90
 *
 *****************************************************************************/


/*****************************************************************************
 *  Include files
 *****************************************************************************/

#include <stdio.h>      /* standard I/O declarations */
#include <ds/ds.h>         /* DA constant definitions */
#include <ds_stub.h>
#include <constants.h>
#include <wex/wex.h>


int ds_getparms(env_key, key_count, keylist, decom, buf_ptr)
    int     env_key;            /* environment key entry number */
    int     key_count;          /* number of keys in keylist */
    int     *keylist;           /* pointer to array of keys */
    p_decom *decom;             /* pointer to decom array */
    char    *buf_ptr;           /* pointer to data value buffer */
/* int     cycle_check;        /* option to indicate cycle retrieval */
                                /* not sent if FAC set */
{
    char *dvb;
    p_decom *dcm;
    long status_selector;
    int *key;
    int cnt;
    static float *e_val;
    static long *f_val;
    static double *d_val;
    static double msid_value = 0.0;
    char *t_val = "This is a default string which doesn't do anything";
    static long *status;
    static int first_call = 1;
    int len, low, high;     /* extracted keylist parms */

    static long stat[7] = {
            DEAD_DATA, MISSING_DATA, STATIC_DATA, LIMIT_HIGH, LIMIT_LOW,
            CRITICAL_HIGH, CRITICAL_LOW };

    static char def_text[5][50] = {
        "AAAAAAAAAAAAAAAAABBBBBBBBBBBBBCCCCCCCCCCDDDDDDDDD",
        "aaaaaaaaaaaaaabbbbbbbbbbbbbbbccccccccccddddddddddd",
        "AAAAAAAAAAAAAAbbbbbbbbbbbbbbCCCCCCCCCCDDDDDDDDDD",
```

```
        "aaaaaaaaaaaaaaaBBBBBBBBBBBBBBBccccccccccccDDDDDDDDD",
        "EEEEEEEEEEEEEEEFFFFFFFFFFFFFGGGGGGGGGGGGGHHHHHHHHHHH"
    };

    D(fprintf(stderr,"START ds_getparms, source %d\n", env_key));

/*
 *  Allocate memory, if first call
 */

    if (first_call) {
        f_val = (long *) malloc(sizeof(long));
        e_val = (float *) malloc(sizeof(float));
        d_val = (double *) malloc(sizeof(double));

        status = (long *) malloc(sizeof(long));

        first_call = 0;
    }

/*
 *  Cycle through keylist, generating random data
 */

    /* initialize pointers and counters */

    dvb = buf_ptr;
    key = keylist;
    dcm = decom;


    for (cnt=0; cnt<key_count; cnt++) {

    D(fprintf(stderr, "cnt = %d  key = %ld  offset = %ld length = %ld  size = %ld  attribu
te = %c\n", cnt, *key, dcm->offset, dcm->length, dcm->size, dcm->attribute));

    /* for each msid, generate the random data and put in buffer */

        dvb = buf_ptr + dcm->offset;

/*
 *      Generate random status character
 */

        *status = 0;
/*
        status_selector = random() % 7;
        *status |= stat[status_selector];
*/

        memcpy(dvb, status, 4);

        len = *key;
        key++;
        low = *key;
        key++;
        high = *key;

        D(fprintf(stderr, "**** len %d low %d high %d\n", len, low, high));

        switch (dcm->attribute) {
            case 'A':                   /* text */
                        *f_val = random() % 5;
```

```
                    strncpy(t_val, def_text[*f_val], len);
                    memcpy(dvb+4, t_val, len);
                    D(fprintf(stderr, "%ld%s\n", *(long *)(dvb), (char *)(dvb+4)));
                    break;
            case 'F':                   /* long */
                    *f_val = (long) ((random() % (high - low + 1)) + low);
                    memcpy(dvb+4, f_val, len);
                    D(fprintf(stderr, "%ld%ld\n", *(long *)(dvb), *(long *)(dvb+4)));
                    break;
            case 'E':                   /* float */
                    *e_val = (float) (random() % (high - low + 1) + low);
                    memcpy(dvb+4, e_val, len);
                    D(fprintf(stderr, "%ld%f\n", *(long *)(dvb), *(float *)(dvb+4)));
                    break;
            case 'D':                   /* double */
                    *d_val = (double) (random() % (high - low + 1) + low);
                    memcpy(dvb+4, d_val, len);
                    D(fprintf(stderr, "%ld%f\n", *(long *)(dvb), *(double *)(dvb+4)));
                    break;
            default:
                    *d_val = (double) (random() % (high - low + 1) + low);
                    memcpy(dvb+4, d_val, len);
                    D(fprintf(stderr, "%ld%f\n", *(long *)(dvb), *(double *)(dvb+4)));
                    break;
        }

        dcm++;
        key++;

    }

    D(fprintf(stderr, "END ds_getparms\n"));

}
```

```
/*****************************************************************************
 * ds_rdppl
 *
 * This is a part of the STUB version of data acquisition.
 * It reads the data for the GDR data sources from the ppl file into a
 * parm_ent structure for use by the stub version of ds_getkeys to build
 * decom entry for each.
 *
 * set the sample count to 1 since is blank in ppl file
 *****************************************************************************/

#include <stdio.h>
#include <ds/ds.h>
#include <ds_stub.h>
#include <wex/wex.h>

int ds_rdppl(pfn, msids)
    char *pfn;
    parm_ent    **msids;
{
    char *get_token();
    char *calloc();
    parm_ent    *msid;

    FILE    *fp;
    int key_count;
    char buffer[256];
    char *bptr;

    D(fprintf(stderr, "START rdppl\n"));

/*
 *  Open ppl file
 */

    fp = fopen(pfn, "r");
    if (fp == NULL) {
        fprintf(stderr, "Unable to open PPL file %s\n", pfn);
        return(-1);
    }

/*
 *  Count # records (so can allocate contiguous memory for all)
 */


    fscanf(fp, "%s\n", buffer);
    key_count = 1;

    while (!feof(fp)) {
        fscanf(fp, "%s\n", buffer);
        key_count++;
    }

    D(fprintf(stderr, "key_count is %d\n", key_count));

/*
 *  Allocate memory for the msids
 */

    *msids = (parm_ent *) calloc(key_count, sizeof(parm_ent));
    msid = *msids;

/*
```

```
 *  Read GDR (ppl) records
 */

    rewind(fp);

    bptr = buffer;

    while (!feof(fp)) {
        fscanf(fp, "%s\n", buffer);
        bptr = buffer;
        strcpy(msid->name, get_token(&bptr));
        msid++;
    }

    fclose(fp);

    D(fprintf(stderr, "END rdppl\n"));

    return(key_count);
}


char *get_token(bptr)
    char **bptr;
{
    char *token = "aaaaaaaaaaaaaaaaaaaa";
    char *start;

    start = token;

    while (**bptr != ',') {
        *token = **bptr;
        token++;
        *bptr = *bptr + 1;
    }

    *token = '\0';

    return(start);

}
```

ATTACHMENT 3 - Display Manager Code

```
################################################################################
# Makefile for the Display Manager.
################################################################################

#
# Initialize master, binary, library, and include directories.
#

MASTER  = /home/project/2984/db
BINDIR  = /WEX/Exec
INCDIR  = $(MASTER)/include
INCDIRS = -I. -I$(INCDIR)
OBJDIR  = ./masscomp

#
# Define the target which this file is to create.
#

TARGET      = $(BINDIR)/dm_mass

#
# Define the libraries to search.
#

LIBRARIES   = -ltui -lXm -lXt -lX11 -lm -lg

#
# Define the compiler and linker flags.
#

#CFLAGS     = -g $(INCDIRS) $(FLAGS) -DFAC
#CFLAGS     = -g $(INCDIRS) $(FLAGS) -DDEBUG -DFAC -DSTUB
CFLAGS      = -g $(INCDIRS) $(FLAGS) -DFAC -DSTUB
#LINTFLAGS  = -axz -DLINT $(INCDIRS) $(FLAGS) -DEBUG -DFAC -DSUN -DSTUB
LINTFLAGS   = -axz -DLINT $(INCDIRS) $(FLAGS) -DFAC -DSUN -DSTUB

#
# Define all objects which make up this target.
#

OBJS        =\
    $(OBJDIR)/DDpbi_updt.o\
    $(OBJDIR)/Dbdata.o\
    $(OBJDIR)/add_pt.o\
    $(OBJDIR)/cb_cmd.o\
    $(OBJDIR)/cb_exp_plot.o\
    $(OBJDIR)/cb_expose.o\
    $(OBJDIR)/cb_help.o\
    $(OBJDIR)/cb_pbi.o\
    $(OBJDIR)/cb_zoom.o\
    $(OBJDIR)/chg_gdr.o\
    $(OBJDIR)/chg_lim.o\
    $(OBJDIR)/chg_zoom.o\
    $(OBJDIR)/chk_flg.o\
    $(OBJDIR)/chk_flt.o\
    $(OBJDIR)/chk_res.o\
    $(OBJDIR)/cleanup.o\
    $(OBJDIR)/clear.o\
    $(OBJDIR)/clr_disp.o\
    $(OBJDIR)/colorpal.o\
    $(OBJDIR)/colors.o\
    $(OBJDIR)/command.o\
    $(OBJDIR)/date_chek.o\
    $(OBJDIR)/dcm_ent.o\
```

```
$(OBJDIR)/ddd.o\
$(OBJDIR)/ddd_msid.o\
$(OBJDIR)/dec_val.o\
$(OBJDIR)/draw_axs.o\
$(OBJDIR)/draw_ovl.o\
$(OBJDIR)/draw_plt.o\
$(OBJDIR)/edit_colors.o\
$(OBJDIR)/ex_msgsnd.o\
$(OBJDIR)/exit_disp.o\
$(OBJDIR)/extract.o\
$(OBJDIR)/first_proc.o\
$(OBJDIR)/flt_data.o\
$(OBJDIR)/font_num.o\
$(OBJDIR)/gdr_next.o\
$(OBJDIR)/get_disp.o\
$(OBJDIR)/get_fn.o\
$(OBJDIR)/get_plot.o\
$(OBJDIR)/globals.o\
$(OBJDIR)/hist_tab.o\
$(OBJDIR)/ht_init.o\
$(OBJDIR)/init.o\
$(OBJDIR)/init_disp.o\
$(OBJDIR)/init_fg.o\
$(OBJDIR)/init_label.o\
$(OBJDIR)/int_ln.o\
$(OBJDIR)/lim_grp.o\
$(OBJDIR)/lim_ln.o\
$(OBJDIR)/limit_val.o\
$(OBJDIR)/list_files.o\
$(OBJDIR)/main.o\
$(OBJDIR)/new_disp.o\
$(OBJDIR)/org_file.o\
$(OBJDIR)/p_atimei.o\
$(OBJDIR)/p_dataval.o\
$(OBJDIR)/p_itimea.o\
$(OBJDIR)/parse_cmd.o\
$(OBJDIR)/pbi_cmd.o\
$(OBJDIR)/pbi_config.o\
$(OBJDIR)/pbi_free.o\
$(OBJDIR)/pbi_host.o\
$(OBJDIR)/pbi_hot.o\
$(OBJDIR)/pbi_local.o\
$(OBJDIR)/pbi_setup.o\
$(OBJDIR)/pbi_updt.o\
$(OBJDIR)/pf_chk.o\
$(OBJDIR)/plot_msid.o\
$(OBJDIR)/plot_ovl.o\
$(OBJDIR)/proc_plt.o\
$(OBJDIR)/read_disp.o\
$(OBJDIR)/read_fgr.o\
$(OBJDIR)/read_files.o\
$(OBJDIR)/read_ovls.o\
$(OBJDIR)/read_pf.o\
$(OBJDIR)/read_plt.o\
$(OBJDIR)/readbg.o\
$(OBJDIR)/readfg.o\
$(OBJDIR)/read_pbi.o\
$(OBJDIR)/redraw.o\
$(OBJDIR)/redwbg.o\
$(OBJDIR)/redwfg.o\
$(OBJDIR)/sel_disp.o\
$(OBJDIR)/set_cmap.o\
$(OBJDIR)/set_gc.o\
$(OBJDIR)/set_label.o\
```

```
        $(OBJDIR)/set_timer.o\
        $(OBJDIR)/shm_creat.o\
        $(OBJDIR)/sort_msid.o\
        $(OBJDIR)/stat_col.o\
        $(OBJDIR)/tick_mk.o\
        $(OBJDIR)/time_val.o\
        $(OBJDIR)/tmr_update.o\
        $(OBJDIR)/ui_init.o\
        $(OBJDIR)/unlatch.o\
        $(OBJDIR)/unv_plot.o\
        $(OBJDIR)/upd_rate.o\
        $(OBJDIR)/update.o\
        $(OBJDIR)/updtbg.o\
        $(OBJDIR)/updtfg.o\
        $(OBJDIR)/updtht.o\
        $(OBJDIR)/val_dt.o\
        $(OBJDIR)/val_fn.o\
        $(OBJDIR)/val_msid.o\
        $(OBJDIR)/val_ppl.o\
        $(OBJDIR)/val_src.o\
        $(OBJDIR)/valmsid.o\
        $(OBJDIR)/wex.o\
        $(OBJDIR)/zoom.o

#
# Make the target.
#

all:        $(TARGET)

$(TARGET):  $(OBJS)
    $(CC) -o $(TARGET) $(OBJS) $(LIBRARIES) $(LDFLAGS)

lint:
    $(LINT) $(LINTFLAGS) *.c $(LINTLIBS)

lintlib:
    $(LINT) -I../include -Cdm dm_lint.c

#
# Individual file dependencies.
#

$(OBJDIR)/DDpbi_updt.o: DDpbi_updt.c
    cc -c DDpbi_updt.c $(CFLAGS)
    mv DDpbi_updt.o $(OBJDIR)

$(OBJDIR)/Dbdata.o: Dbdata.c
    cc -c Dbdata.c $(CFLAGS)
    mv Dbdata.o $(OBJDIR)
#:n

$(OBJDIR)/add_pt.o: add_pt.c
    cc -c add_pt.c $(CFLAGS)
    mv add_pt.o $(OBJDIR)

$(OBJDIR)/cb_cmd.o: cb_cmd.c
    cc -c cb_cmd.c $(CFLAGS)
    mv cb_cmd.o $(OBJDIR)

$(OBJDIR)/cb_exp_plot.o: cb_exp_plot.c
    cc -c cb_exp_plot.c $(CFLAGS)
    mv cb_exp_plot.o $(OBJDIR)
```

```
$(OBJDIR)/cb_expose.o: cb_expose.c
    cc -c cb_expose.c $(CFLAGS)
    mv cb_expose.o $(OBJDIR)

$(OBJDIR)/cb_help.o: cb_help.c
    cc -c cb_help.c $(CFLAGS)
    mv cb_help.o $(OBJDIR)

$(OBJDIR)/cb_pbi.o: cb_pbi.c
    cc -c cb_pbi.c $(CFLAGS)
    mv cb_pbi.o $(OBJDIR)

$(OBJDIR)/cb_zoom.o: cb_zoom.c
    cc -c cb_zoom.c $(CFLAGS)
    mv cb_zoom.o $(OBJDIR)

$(OBJDIR)/chg_gdr.o: chg_gdr.c
    cc -c chg_gdr.c $(CFLAGS)
    mv chg_gdr.o $(OBJDIR)

$(OBJDIR)/chg_lim.o: chg_lim.c
    cc -c chg_lim.c $(CFLAGS)
    mv chg_lim.o $(OBJDIR)

$(OBJDIR)/chg_zoom.o: chg_zoom.c
    cc -c chg_zoom.c $(CFLAGS)
    mv chg_zoom.o $(OBJDIR)

$(OBJDIR)/chk_flg.o: chk_flg.c
    cc -c chk_flg.c $(CFLAGS)
    mv chk_flg.o $(OBJDIR)

$(OBJDIR)/chk_flt.o: chk_flt.c
    cc -c chk_flt.c $(CFLAGS)
    mv chk_flt.o $(OBJDIR)

$(OBJDIR)/chk_res.o: chk_res.c
    cc -c chk_res.c $(CFLAGS)
    mv chk_res.o $(OBJDIR)

$(OBJDIR)/cleanup.o: cleanup.c
    cc -c cleanup.c $(CFLAGS)
    mv cleanup.o $(OBJDIR)

$(OBJDIR)/clear.o: clear.c
    cc -c clear.c $(CFLAGS)
    mv clear.o $(OBJDIR)

$(OBJDIR)/clr_disp.o: clr_disp.c
    cc -c clr_disp.c $(CFLAGS)
    mv clr_disp.o $(OBJDIR)

$(OBJDIR)/colorpal.o: colorpal.c
    cc -c colorpal.c $(CFLAGS)
    mv colorpal.o $(OBJDIR)

$(OBJDIR)/colors.o: colors.c
    cc -c colors.c $(CFLAGS)
    mv colors.o $(OBJDIR)

$(OBJDIR)/command.o: command.c
    cc -c command.c $(CFLAGS)
    mv command.o $(OBJDIR)
```

```
$(OBJDIR)/date_chek.o: date_chek.c
    cc -c date_chek.c $(CFLAGS)
    mv date_chek.o $(OBJDIR)

$(OBJDIR)/dcm_ent.o: dcm_ent.c
    cc -c dcm_ent.c $(CFLAGS)
    mv dcm_ent.o $(OBJDIR)

$(OBJDIR)/ddd.o: ddd.c
    cc -c ddd.c $(CFLAGS)
    mv ddd.o $(OBJDIR)

$(OBJDIR)/ddd_msid.o: ddd_msid.c
    cc -c ddd_msid.c $(CFLAGS)
    mv ddd_msid.o $(OBJDIR)

$(OBJDIR)/dec_val.o: dec_val.c
    cc -c dec_val.c $(CFLAGS)
    mv dec_val.o $(OBJDIR)

$(OBJDIR)/draw_axs.o: draw_axs.c
    cc -c draw_axs.c $(CFLAGS)
    mv draw_axs.o $(OBJDIR)

$(OBJDIR)/draw_ovl.o: draw_ovl.c
    cc -c draw_ovl.c $(CFLAGS)
    mv draw_ovl.o $(OBJDIR)

$(OBJDIR)/draw_plt.o: draw_plt.c
    cc -c draw_plt.c $(CFLAGS)
    mv draw_plt.o $(OBJDIR)

$(OBJDIR)/edit_colors.o: edit_colors.c
    cc -c edit_colors.c $(CFLAGS)
    mv edit_colors.o $(OBJDIR)

$(OBJDIR)/ex_msgsnd.o: ex_msgsnd.c
    cc -c ex_msgsnd.c $(CFLAGS)
    mv ex_msgsnd.o $(OBJDIR)

$(OBJDIR)/exit_disp.o: exit_disp.c
    cc -c exit_disp.c $(CFLAGS)
    mv exit_disp.o $(OBJDIR)

$(OBJDIR)/extract.o: extract.c
    cc -c extract.c $(CFLAGS)
    mv extract.o $(OBJDIR)

$(OBJDIR)/first_proc.o: first_proc.c
    cc -c first_proc.c $(CFLAGS)
    mv first_proc.o $(OBJDIR)

$(OBJDIR)/flt_data.o: flt_data.c
    cc -c flt_data.c $(CFLAGS)
    mv flt_data.o $(OBJDIR)

$(OBJDIR)/font_num.o: font_num.c
    cc -c font_num.c $(CFLAGS)
    mv font_num.o $(OBJDIR)

$(OBJDIR)/gdr_next.o: gdr_next.c
    cc -c gdr_next.c $(CFLAGS)
    mv gdr_next.o $(OBJDIR)
```

```
$(OBJDIR)/get_disp.o: get_disp.c
    cc -c get_disp.c $(CFLAGS)
    mv get_disp.o $(OBJDIR)

$(OBJDIR)/get_fn.o: get_fn.c
    cc -c get_fn.c $(CFLAGS)
    mv get_fn.o $(OBJDIR)

$(OBJDIR)/get_plot.o: get_plot.c
    cc -c get_plot.c $(CFLAGS)
    mv get_plot.o $(OBJDIR)

$(OBJDIR)/globals.o: globals.c
    cc -c globals.c $(CFLAGS)
    mv globals.o $(OBJDIR)

$(OBJDIR)/hist_tab.o: hist_tab.c
    cc -c hist_tab.c $(CFLAGS)
    mv hist_tab.o $(OBJDIR)

$(OBJDIR)/ht_init.o: ht_init.c
    cc -c ht_init.c $(CFLAGS)
    mv ht_init.o $(OBJDIR)

$(OBJDIR)/init.o: init.c
    cc -c init.c $(CFLAGS)
    mv init.o $(OBJDIR)

$(OBJDIR)/init_disp.o: init_disp.c
    cc -c init_disp.c $(CFLAGS)
    mv init_disp.o $(OBJDIR)

$(OBJDIR)/init_fg.o: init_fg.c
    cc -c init_fg.c $(CFLAGS)
    mv init_fg.o $(OBJDIR)

$(OBJDIR)/init_label.o: init_label.c
    cc -c init_label.c $(CFLAGS)
    mv init_label.o $(OBJDIR)

$(OBJDIR)/int_ln.o: int_ln.c
    cc -c int_ln.c $(CFLAGS)
    mv int_ln.o $(OBJDIR)

$(OBJDIR)/lim_grp.o: lim_grp.c
    cc -c lim_grp.c $(CFLAGS)
    mv lim_grp.o $(OBJDIR)

$(OBJDIR)/lim_ln.o: lim_ln.c
    cc -c lim_ln.c $(CFLAGS)
    mv lim_ln.o $(OBJDIR)

$(OBJDIR)/limit_val.o: limit_val.c
    cc -c limit_val.c $(CFLAGS)
    mv limit_val.o $(OBJDIR)

$(OBJDIR)/list_files.o: list_files.c
    cc -c list_files.c $(CFLAGS)
    mv list_files.o $(OBJDIR)

$(OBJDIR)/main.o: main.c
    cc -c main.c $(CFLAGS)
    mv main.o $(OBJDIR)
```

```
$(OBJDIR)/new_disp.o: new_disp.c
    cc -c new_disp.c $(CFLAGS)
    mv new_disp.o $(OBJDIR)

$(OBJDIR)/org_file.o: org_file.c
    cc -c org_file.c $(CFLAGS)
    mv org_file.o $(OBJDIR)

$(OBJDIR)/p_atimei.o: p_atimei.c
    cc -c p_atimei.c $(CFLAGS)
    mv p_atimei.o $(OBJDIR)

$(OBJDIR)/p_dataval.o: p_dataval.c
    cc -c p_dataval.c $(CFLAGS)
    mv p_dataval.o $(OBJDIR)

$(OBJDIR)/p_itimea.o: p_itimea.c
    cc -c p_itimea.c $(CFLAGS)
    mv p_itimea.o $(OBJDIR)

$(OBJDIR)/parse_cmd.o: parse_cmd.c
    cc -c parse_cmd.c $(CFLAGS)
    mv parse_cmd.o $(OBJDIR)

$(OBJDIR)/pbi_cmd.o: pbi_cmd.c
    cc -c pbi_cmd.c $(CFLAGS)
    mv pbi_cmd.o $(OBJDIR)

$(OBJDIR)/pbi_config.o: pbi_config.c
    cc -c pbi_config.c $(CFLAGS)
    mv pbi_config.o $(OBJDIR)

$(OBJDIR)/pbi_free.o: pbi_free.c
    cc -c pbi_free.c $(CFLAGS)
    mv pbi_free.o $(OBJDIR)

$(OBJDIR)/pbi_host.o: pbi_host.c
    cc -c pbi_host.c $(CFLAGS)
    mv pbi_host.o $(OBJDIR)

$(OBJDIR)/pbi_hot.o: pbi_hot.c
    cc -c pbi_hot.c $(CFLAGS)
    mv pbi_hot.o $(OBJDIR)

$(OBJDIR)/pbi_local.o: pbi_local.c
    cc -c pbi_local.c $(CFLAGS)
    mv pbi_local.o $(OBJDIR)

$(OBJDIR)/pbi_setup.o: pbi_setup.c
    cc -c pbi_setup.c $(CFLAGS)
    mv pbi_setup.o $(OBJDIR)

$(OBJDIR)/pbi_updt.o: pbi_updt.c
    cc -c pbi_updt.c $(CFLAGS)
    mv pbi_updt.o $(OBJDIR)

$(OBJDIR)/pdt_feed.o: pdt_feed.c
    cc -c pdt_feed.c $(CFLAGS)
    mv pdt_feed.o $(OBJDIR)

$(OBJDIR)/pf_chk.o: pf_chk.c
    cc -c pf_chk.c $(CFLAGS)
    mv pf_chk.o $(OBJDIR)
```

```
$(OBJDIR)/plot_msid.o: plot_msid.c
    cc -c plot_msid.c $(CFLAGS)
    mv plot_msid.o $(OBJDIR)

$(OBJDIR)/plot_ovl.o: plot_ovl.c
    cc -c plot_ovl.c $(CFLAGS)
    mv plot_ovl.o $(OBJDIR)

$(OBJDIR)/proc_plt.o: proc_plt.c
    cc -c proc_plt.c $(CFLAGS)
    mv proc_plt.o $(OBJDIR)

$(OBJDIR)/read_disp.o: read_disp.c
    cc -c read_disp.c $(CFLAGS)
    mv read_disp.o $(OBJDIR)

$(OBJDIR)/read_fgr.o: read_fgr.c
    cc -c read_fgr.c $(CFLAGS)
    mv read_fgr.o $(OBJDIR)

$(OBJDIR)/read_files.o: read_files.c
    cc -c read_files.c $(CFLAGS)
    mv read_files.o $(OBJDIR)

$(OBJDIR)/read_ovls.o: read_ovls.c
    cc -c read_ovls.c $(CFLAGS)
    mv read_ovls.o $(OBJDIR)

$(OBJDIR)/read_pbi.o: read_pbi.c
    cc -c read_pbi.c $(CFLAGS)
    mv read_pbi.o $(OBJDIR)

$(OBJDIR)/read_pf.o: read_pf.c
    cc -c read_pf.c $(CFLAGS)
    mv read_pf.o $(OBJDIR)

$(OBJDIR)/read_plt.o: read_plt.c
    cc -c read_plt.c $(CFLAGS)
    mv read_plt.o $(OBJDIR)

$(OBJDIR)/readbg.o: readbg.c
    cc -c readbg.c $(CFLAGS)
    mv readbg.o $(OBJDIR)

$(OBJDIR)/readfg.o: readfg.c
    cc -c readfg.c $(CFLAGS)
    mv readfg.o $(OBJDIR)

$(OBJDIR)/redraw.o: redraw.c
    cc -c redraw.c $(CFLAGS)
    mv redraw.o $(OBJDIR)

$(OBJDIR)/redwbg.o: redwbg.c
    cc -c redwbg.c $(CFLAGS)
    mv redwbg.o $(OBJDIR)

$(OBJDIR)/redwfg.o: redwfg.c
    cc -c redwfg.c $(CFLAGS)
    mv redwfg.o $(OBJDIR)

$(OBJDIR)/sel_disp.o: sel_disp.c
    cc -c sel_disp.c $(CFLAGS)
    mv sel_disp.o $(OBJDIR)
```

```
$(OBJDIR)/set_cmap.o: set_cmap.c
    cc -c set_cmap.c $(CFLAGS)
    mv set_cmap.o $(OBJDIR)

$(OBJDIR)/set_gc.o: set_gc.c
    cc -c set_gc.c $(CFLAGS)
    mv set_gc.o $(OBJDIR)

$(OBJDIR)/set_label.o: set_label.c
    cc -c set_label.c $(CFLAGS)
    mv set_label.o $(OBJDIR)

$(OBJDIR)/set_timer.o: set_timer.c
    cc -c set_timer.c $(CFLAGS)
    mv set_timer.o $(OBJDIR)

$(OBJDIR)/shm_creat.o: shm_creat.c
    cc -c shm_creat.c $(CFLAGS)
    mv shm_creat.o $(OBJDIR)

$(OBJDIR)/sort_msid.o: sort_msid.c
    cc -c sort_msid.c $(CFLAGS)
    mv sort_msid.o $(OBJDIR)

$(OBJDIR)/stat_col.o: stat_col.c
    cc -c stat_col.c $(CFLAGS)
    mv stat_col.o $(OBJDIR)

$(OBJDIR)/test.o: test.c
    cc -c test.c $(CFLAGS)
    mv test.o $(OBJDIR)

$(OBJDIR)/tick_mk.o: tick_mk.c
    cc -c tick_mk.c $(CFLAGS)
    mv tick_mk.o $(OBJDIR)

$(OBJDIR)/time_val.o: time_val.c
    cc -c time_val.c $(CFLAGS)
    mv time_val.o $(OBJDIR)

$(OBJDIR)/tmr_update.o: tmr_update.c
    cc -c tmr_update.c $(CFLAGS)
    mv tmr_update.o $(OBJDIR)

$(OBJDIR)/ui_init.o: ui_init.c
    cc -c ui_init.c $(CFLAGS)
    mv ui_init.o $(OBJDIR)

$(OBJDIR)/unlatch.o: unlatch.c
    cc -c unlatch.c $(CFLAGS)
    mv unlatch.o $(OBJDIR)

$(OBJDIR)/unv_plot.o: unv_plot.c
    cc -c unv_plot.c $(CFLAGS)
    mv unv_plot.o $(OBJDIR)

$(OBJDIR)/upd_rate.o: upd_rate.c
    cc -c upd_rate.c $(CFLAGS)
    mv upd_rate.o $(OBJDIR)

$(OBJDIR)/update.o: update.c
    cc -c update.c $(CFLAGS)
    mv update.o $(OBJDIR)
```

```
$(OBJDIR)/updtbg.o: updtbg.c
    cc -c updtbg.c $(CFLAGS)
    mv updtbg.o $(OBJDIR)

$(OBJDIR)/updtfg.o: updtfg.c
    cc -c updtfg.c $(CFLAGS)
    mv updtfg.o $(OBJDIR)

$(OBJDIR)/updtht.o: updtht.c
    cc -c updtht.c $(CFLAGS)
    mv updtht.o $(OBJDIR)

$(OBJDIR)/val_dt.o: val_dt.c
    cc -c val_dt.c $(CFLAGS)
    mv val_dt.o $(OBJDIR)

$(OBJDIR)/val_fn.o: val_fn.c
    cc -c val_fn.c $(CFLAGS)
    mv val_fn.o $(OBJDIR)

$(OBJDIR)/val_msid.o: val_msid.c
    cc -c val_msid.c $(CFLAGS)
    mv val_msid.o $(OBJDIR)

$(OBJDIR)/val_ppl.o: val_ppl.c
    cc -c val_ppl.c $(CFLAGS)
    mv val_ppl.o $(OBJDIR)

$(OBJDIR)/val_src.o: val_src.c
    cc -c val_src.c $(CFLAGS)
    mv val_src.o $(OBJDIR)

$(OBJDIR)/valmsid.o: valmsid.c
    cc -c valmsid.c $(CFLAGS)
    mv valmsid.o $(OBJDIR)

$(OBJDIR)/wex.o: wex.c
    cc -c wex.c $(CFLAGS)
    mv wex.o $(OBJDIR)

$(OBJDIR)/zoom.o: zoom.c
    cc -c zoom.c $(CFLAGS)
    mv zoom.o $(OBJDIR)
```

```
################################################################################
# Makefile for the Display Manager.
################################################################################

#
# Initialize master, binary, library, and include directories.
#

MASTER  = /home/project/2984/db
BINDIR  = /WEX/Exec
INCDIR  = $(MASTER)/include
INCDIRS = -I. -I$(INCDIR)
OBJDIR  = ./sun

#
# Define the target which this file is to create.
#

TARGET      = $(BINDIR)/dm_sun

#
# Define the libraries to search.
#

LIBRARIES   = -ltui -lXm -lXt -lX11 -lm -lg

#
# Define the compiler and linker flags.
#

#CFLAGS     = -misalign -g $(INCDIRS) $(FLAGS) -DDEBUG -DFAC -DSUN -DSTUB
CFLAGS      = -misalign -g $(INCDIRS) $(FLAGS) -DFAC -DSUN -DSTUB
#LINTFLAGS  = -axz -DLINT $(INCDIRS) $(FLAGS) -DEBUG -DFAC -DSUN -DSTUB
LINTFLAGS   = -axz -DLINT $(INCDIRS) $(FLAGS) -DFAC -DSUN -DSTUB

#
# Define all objects which make up this target.
#

OBJS        =\
    $(OBJDIR)/DDpbi_updt.o\
    $(OBJDIR)/Dbdata.o\
    $(OBJDIR)/add_pt.o\
    $(OBJDIR)/cb_cmd.o\
    $(OBJDIR)/cb_exp_plot.o\
    $(OBJDIR)/cb_expose.o\
    $(OBJDIR)/cb_help.o\
    $(OBJDIR)/cb_pbi.o\
    $(OBJDIR)/cb_zoom.o\
    $(OBJDIR)/chg_gdr.o\
    $(OBJDIR)/chg_lim.o\
    $(OBJDIR)/chg_zoom.o\
    $(OBJDIR)/chk_flg.o\
    $(OBJDIR)/chk_flt.o\
    $(OBJDIR)/chk_res.o\
    $(OBJDIR)/cleanup.o\
    $(OBJDIR)/clear.o\
    $(OBJDIR)/clr_disp.o\
    $(OBJDIR)/colorpal.o\
    $(OBJDIR)/colors.o\
    $(OBJDIR)/command.o\
    $(OBJDIR)/date_chek.o\
    $(OBJDIR)/dcm_ent.o\
    $(OBJDIR)/ddd.o\
```

```
$(OBJDIR)/ddd_msid.o\
$(OBJDIR)/dec_val.o\
$(OBJDIR)/draw_axs.o\
$(OBJDIR)/draw_ovl.o\
$(OBJDIR)/draw_plt.o\
$(OBJDIR)/edit_colors.o\
$(OBJDIR)/ex_msgsnd.o\
$(OBJDIR)/exit_disp.o\
$(OBJDIR)/extract.o\
$(OBJDIR)/first_proc.o\
$(OBJDIR)/flt_data.o\
$(OBJDIR)/font_num.o\
$(OBJDIR)/gdr_next.o\
$(OBJDIR)/get_disp.o\
$(OBJDIR)/get_fn.o\
$(OBJDIR)/get_plot.o\
$(OBJDIR)/globals.o\
$(OBJDIR)/hist_tab.o\
$(OBJDIR)/ht_init.o\
$(OBJDIR)/init.o\
$(OBJDIR)/init_disp.o\
$(OBJDIR)/init_fg.o\
$(OBJDIR)/init_label.o\
$(OBJDIR)/int_ln.o\
$(OBJDIR)/lim_grp.o\
$(OBJDIR)/lim_ln.o\
$(OBJDIR)/limit_val.o\
$(OBJDIR)/list_files.o\
$(OBJDIR)/main.o\
$(OBJDIR)/new_disp.o\
$(OBJDIR)/org_file.o\
$(OBJDIR)/p_atimei.o\
$(OBJDIR)/p_dataval.o\
$(OBJDIR)/p_itimea.o\
$(OBJDIR)/parse_cmd.o\
$(OBJDIR)/pbi_cmd.o\
$(OBJDIR)/pbi_config.o\
$(OBJDIR)/pbi_free.o\
$(OBJDIR)/pbi_host.o\
$(OBJDIR)/pbi_hot.o\
$(OBJDIR)/pbi_local.o\
$(OBJDIR)/pbi_setup.o\
$(OBJDIR)/pbi_updt.o\
$(OBJDIR)/pf_chk.o\
$(OBJDIR)/plot_msid.o\
$(OBJDIR)/plot_ovl.o\
$(OBJDIR)/proc_plt.o\
$(OBJDIR)/read_disp.o\
$(OBJDIR)/read_fgr.o\
$(OBJDIR)/read_files.o\
$(OBJDIR)/read_ovls.o\
$(OBJDIR)/read_pf.o\
$(OBJDIR)/read_plt.o\
$(OBJDIR)/readbg.o\
$(OBJDIR)/readfg.o\
$(OBJDIR)/read_pbi.o\
$(OBJDIR)/redraw.o\
$(OBJDIR)/redwbg.o\
$(OBJDIR)/redwfg.o\
$(OBJDIR)/sel_disp.o\
$(OBJDIR)/set_cmap.o\
$(OBJDIR)/set_gc.o\
$(OBJDIR)/set_label.o\
$(OBJDIR)/set_timer.o\
```

```
	$(OBJDIR)/shm_creat.o\
	$(OBJDIR)/sort_msid.o\
	$(OBJDIR)/stat_col.o\
	$(OBJDIR)/tick_mk.o\
	$(OBJDIR)/time_val.o\
	$(OBJDIR)/tmr_update.o\
	$(OBJDIR)/ui_init.o\
	$(OBJDIR)/unlatch.o\
	$(OBJDIR)/unv_plot.o\
	$(OBJDIR)/upd_rate.o\
	$(OBJDIR)/update.o\
	$(OBJDIR)/updtbg.o\
	$(OBJDIR)/updtfg.o\
	$(OBJDIR)/updtht.o\
	$(OBJDIR)/val_dt.o\
	$(OBJDIR)/val_fn.o\
	$(OBJDIR)/val_msid.o\
	$(OBJDIR)/val_ppl.o\
	$(OBJDIR)/val_src.o\
	$(OBJDIR)/valmsid.o\
	$(OBJDIR)/wex.o\
	$(OBJDIR)/zoom.o

#
# Make the target.
#

all:		$(TARGET)

$(TARGET):  $(OBJS)
	$(CC) -o $(TARGET) $(OBJS) -L/home/project/2984/db/ui $(LIBRARIES) $(LDFLAGS)

lint: *.c
	$(LINT) $(LINTFLAGS) *.c $(LINTLIBS)

lintlib:
	$(LINT) -I../include -Cdm dm_lint.c

#
# Individual file dependencies.
#

$(OBJDIR)/DDpbi_updt.o: DDpbi_updt.c
	cc -c DDpbi_updt.c $(CFLAGS)
	mv DDpbi_updt.o $(OBJDIR)

$(OBJDIR)/Dbdata.o: Dbdata.c
	cc -c Dbdata.c $(CFLAGS)
	mv Dbdata.o $(OBJDIR)

$(OBJDIR)/add_pt.o: add_pt.c
	cc -c add_pt.c $(CFLAGS)
	mv add_pt.o $(OBJDIR)

$(OBJDIR)/cb_cmd.o: cb_cmd.c
	cc -c cb_cmd.c $(CFLAGS)
	mv cb_cmd.o $(OBJDIR)

$(OBJDIR)/cb_exp_plot.o: cb_exp_plot.c
	cc -c cb_exp_plot.c $(CFLAGS)
	mv cb_exp_plot.o $(OBJDIR)

$(OBJDIR)/cb_expose.o: cb_expose.c
	cc -c cb_expose.c $(CFLAGS)
```

```
	mv cb_expose.o $(OBJDIR)

$(OBJDIR)/cb_help.o: cb_help.c
	cc -c cb_help.c $(CFLAGS)
	mv cb_help.o $(OBJDIR)

$(OBJDIR)/cb_pbi.o: cb_pbi.c
	cc -c cb_pbi.c $(CFLAGS)
	mv cb_pbi.o $(OBJDIR)

$(OBJDIR)/cb_zoom.o: cb_zoom.c
	cc -c cb_zoom.c $(CFLAGS)
	mv cb_zoom.o $(OBJDIR)

$(OBJDIR)/chg_gdr.o: chg_gdr.c
	cc -c chg_gdr.c $(CFLAGS)
	mv chg_gdr.o $(OBJDIR)

$(OBJDIR)/chg_lim.o: chg_lim.c
	cc -c chg_lim.c $(CFLAGS)
	mv chg_lim.o $(OBJDIR)

$(OBJDIR)/chg_zoom.o: chg_zoom.c
	cc -c chg_zoom.c $(CFLAGS)
	mv chg_zoom.o $(OBJDIR)

$(OBJDIR)/chk_flg.o: chk_flg.c
	cc -c chk_flg.c $(CFLAGS)
	mv chk_flg.o $(OBJDIR)

$(OBJDIR)/chk_flt.o: chk_flt.c
	cc -c chk_flt.c $(CFLAGS)
	mv chk_flt.o $(OBJDIR)

$(OBJDIR)/chk_res.o: chk_res.c
	cc -c chk_res.c $(CFLAGS)
	mv chk_res.o $(OBJDIR)

$(OBJDIR)/cleanup.o: cleanup.c
	cc -c cleanup.c $(CFLAGS)
	mv cleanup.o $(OBJDIR)

$(OBJDIR)/clear.o: clear.c
	cc -c clear.c $(CFLAGS)
	mv clear.o $(OBJDIR)

$(OBJDIR)/clr_disp.o: clr_disp.c
	cc -c clr_disp.c $(CFLAGS)
	mv clr_disp.o $(OBJDIR)

$(OBJDIR)/colorpal.o: colorpal.c
	cc -c colorpal.c $(CFLAGS)
	mv colorpal.o $(OBJDIR)

$(OBJDIR)/colors.o: colors.c
	cc -c colors.c $(CFLAGS)
	mv colors.o $(OBJDIR)

$(OBJDIR)/command.o: command.c
	cc -c command.c $(CFLAGS)
	mv command.o $(OBJDIR)

$(OBJDIR)/date_chek.o: date_chek.c
	cc -c date_chek.c $(CFLAGS)
```

```
	mv date_chek.o $(OBJDIR)

$(OBJDIR)/dcm_ent.o: dcm_ent.c
	cc -c dcm_ent.c $(CFLAGS)
	mv dcm_ent.o $(OBJDIR)

$(OBJDIR)/ddd.o: ddd.c
	cc -c ddd.c $(CFLAGS)
	mv ddd.o $(OBJDIR)

$(OBJDIR)/ddd_msid.o: ddd_msid.c
	cc -c ddd_msid.c $(CFLAGS)
	mv ddd_msid.o $(OBJDIR)

$(OBJDIR)/dec_val.o: dec_val.c
	cc -c dec_val.c $(CFLAGS)
	mv dec_val.o $(OBJDIR)

$(OBJDIR)/draw_axs.o: draw_axs.c
	cc -c draw_axs.c $(CFLAGS)
	mv draw_axs.o $(OBJDIR)

$(OBJDIR)/draw_ovl.o: draw_ovl.c
	cc -c draw_ovl.c $(CFLAGS)
	mv draw_ovl.o $(OBJDIR)

$(OBJDIR)/draw_plt.o: draw_plt.c
	cc -c draw_plt.c $(CFLAGS)
	mv draw_plt.o $(OBJDIR)

$(OBJDIR)/edit_colors.o: edit_colors.c
	cc -c edit_colors.c $(CFLAGS)
	mv edit_colors.o $(OBJDIR)

$(OBJDIR)/ex_msgsnd.o: ex_msgsnd.c
	cc -c ex_msgsnd.c $(CFLAGS)
	mv ex_msgsnd.o $(OBJDIR)

$(OBJDIR)/exit_disp.o: exit_disp.c
	cc -c exit_disp.c $(CFLAGS)
	mv exit_disp.o $(OBJDIR)

$(OBJDIR)/extract.o: extract.c
	cc -c extract.c $(CFLAGS)
	mv extract.o $(OBJDIR)

$(OBJDIR)/first_proc.o: first_proc.c
	cc -c first_proc.c $(CFLAGS)
	mv first_proc.o $(OBJDIR)

$(OBJDIR)/flt_data.o: flt_data.c
	cc -c flt_data.c $(CFLAGS)
	mv flt_data.o $(OBJDIR)

$(OBJDIR)/font_num.o: font_num.c
	cc -c font_num.c $(CFLAGS)
	mv font_num.o $(OBJDIR)

$(OBJDIR)/gdr_next.o: gdr_next.c
	cc -c gdr_next.c $(CFLAGS)
	mv gdr_next.o $(OBJDIR)

$(OBJDIR)/get_disp.o: get_disp.c
	cc -c get_disp.c $(CFLAGS)
```

```
	mv get_disp.o $(OBJDIR)

$(OBJDIR)/get_fn.o: get_fn.c
	cc -c get_fn.c $(CFLAGS)
	mv get_fn.o $(OBJDIR)

$(OBJDIR)/get_plot.o: get_plot.c
	cc -c get_plot.c $(CFLAGS)
	mv get_plot.o $(OBJDIR)

$(OBJDIR)/globals.o: globals.c
	cc -c globals.c $(CFLAGS)
	mv globals.o $(OBJDIR)

$(OBJDIR)/hist_tab.o: hist_tab.c
	cc -c hist_tab.c $(CFLAGS)
	mv hist_tab.o $(OBJDIR)

$(OBJDIR)/ht_init.o: ht_init.c
	cc -c ht_init.c $(CFLAGS)
	mv ht_init.o $(OBJDIR)

$(OBJDIR)/init.o: init.c
	cc -c init.c $(CFLAGS)
	mv init.o $(OBJDIR)

$(OBJDIR)/init_disp.o: init_disp.c
	cc -c init_disp.c $(CFLAGS)
	mv init_disp.o $(OBJDIR)

$(OBJDIR)/init_fg.o: init_fg.c
	cc -c init_fg.c $(CFLAGS)
	mv init_fg.o $(OBJDIR)

$(OBJDIR)/init_label.o: init_label.c
	cc -c init_label.c $(CFLAGS)
	mv init_label.o $(OBJDIR)

$(OBJDIR)/int_ln.o: int_ln.c
	cc -c int_ln.c $(CFLAGS)
	mv int_ln.o $(OBJDIR)

$(OBJDIR)/lim_grp.o: lim_grp.c
	cc -c lim_grp.c $(CFLAGS)
	mv lim_grp.o $(OBJDIR)

$(OBJDIR)/lim_ln.o: lim_ln.c
	cc -c lim_ln.c $(CFLAGS)
	mv lim_ln.o $(OBJDIR)

$(OBJDIR)/limit_val.o: limit_val.c
	cc -c limit_val.c $(CFLAGS)
	mv limit_val.o $(OBJDIR)

$(OBJDIR)/list_files.o: list_files.c
	cc -c list_files.c $(CFLAGS)
	mv list_files.o $(OBJDIR)

$(OBJDIR)/main.o: main.c
	cc -c main.c $(CFLAGS)
	mv main.o $(OBJDIR)

$(OBJDIR)/new_disp.o: new_disp.c
	cc -c new_disp.c $(CFLAGS)
```

```
	mv new_disp.o $(OBJDIR)

$(OBJDIR)/org_file.o: org_file.c
	cc -c org_file.c $(CFLAGS)
	mv org_file.o $(OBJDIR)

$(OBJDIR)/p_atimei.o: p_atimei.c
	cc -c p_atimei.c $(CFLAGS)
	mv p_atimei.o $(OBJDIR)

$(OBJDIR)/p_dataval.o: p_dataval.c
	cc -c p_dataval.c $(CFLAGS)
	mv p_dataval.o $(OBJDIR)

$(OBJDIR)/p_itimea.o: p_itimea.c
	cc -c p_itimea.c $(CFLAGS)
	mv p_itimea.o $(OBJDIR)

$(OBJDIR)/parse_cmd.o: parse_cmd.c
	cc -c parse_cmd.c $(CFLAGS)
	mv parse_cmd.o $(OBJDIR)

$(OBJDIR)/pbi_cmd.o: pbi_cmd.c
	cc -c pbi_cmd.c $(CFLAGS)
	mv pbi_cmd.o $(OBJDIR)

$(OBJDIR)/pbi_config.o: pbi_config.c
	cc -c pbi_config.c $(CFLAGS)
	mv pbi_config.o $(OBJDIR)

$(OBJDIR)/pbi_free.o: pbi_free.c
	cc -c pbi_free.c $(CFLAGS)
	mv pbi_free.o $(OBJDIR)

$(OBJDIR)/pbi_host.o: pbi_host.c
	cc -c pbi_host.c $(CFLAGS)
	mv pbi_host.o $(OBJDIR)

$(OBJDIR)/pbi_hot.o: pbi_hot.c
	cc -c pbi_hot.c $(CFLAGS)
	mv pbi_hot.o $(OBJDIR)

$(OBJDIR)/pbi_local.o: pbi_local.c
	cc -c pbi_local.c $(CFLAGS)
	mv pbi_local.o $(OBJDIR)

$(OBJDIR)/pbi_setup.o: pbi_setup.c
	cc -c pbi_setup.c $(CFLAGS)
	mv pbi_setup.o $(OBJDIR)

$(OBJDIR)/pbi_updt.o: pbi_updt.c
	cc -c pbi_updt.c $(CFLAGS)
	mv pbi_updt.o $(OBJDIR)

$(OBJDIR)/pdt_feed.o: pdt_feed.c
	cc -c pdt_feed.c $(CFLAGS)
	mv pdt_feed.o $(OBJDIR)

$(OBJDIR)/pf_chk.o: pf_chk.c
	cc -c pf_chk.c $(CFLAGS)
	mv pf_chk.o $(OBJDIR)

$(OBJDIR)/plot_msid.o: plot_msid.c
	cc -c plot_msid.c $(CFLAGS)
```

```
	mv plot_msid.o $(OBJDIR)

$(OBJDIR)/plot_ovl.o: plot_ovl.c
	cc -c plot_ovl.c $(CFLAGS)
	mv plot_ovl.o $(OBJDIR)

$(OBJDIR)/proc_plt.o: proc_plt.c
	cc -c proc_plt.c $(CFLAGS)
	mv proc_plt.o $(OBJDIR)

$(OBJDIR)/read_disp.o: read_disp.c
	cc -c read_disp.c $(CFLAGS)
	mv read_disp.o $(OBJDIR)

$(OBJDIR)/read_fgr.o: read_fgr.c
	cc -c read_fgr.c $(CFLAGS)
	mv read_fgr.o $(OBJDIR)

$(OBJDIR)/read_files.o: read_files.c
	cc -c read_files.c $(CFLAGS)
	mv read_files.o $(OBJDIR)

$(OBJDIR)/read_ovls.o: read_ovls.c
	cc -c read_ovls.c $(CFLAGS)
	mv read_ovls.o $(OBJDIR)

$(OBJDIR)/read_pbi.o: read_pbi.c
	cc -c read_pbi.c $(CFLAGS)
	mv read_pbi.o $(OBJDIR)

$(OBJDIR)/read_pf.o: read_pf.c
	cc -c read_pf.c $(CFLAGS)
	mv read_pf.o $(OBJDIR)

$(OBJDIR)/read_plt.o: read_plt.c
	cc -c read_plt.c $(CFLAGS)
	mv read_plt.o $(OBJDIR)

$(OBJDIR)/readbg.o: readbg.c
	cc -c readbg.c $(CFLAGS)
	mv readbg.o $(OBJDIR)

$(OBJDIR)/readfg.o: readfg.c
	cc -c readfg.c $(CFLAGS)
	mv readfg.o $(OBJDIR)

$(OBJDIR)/redraw.o: redraw.c
	cc -c redraw.c $(CFLAGS)
	mv redraw.o $(OBJDIR)

$(OBJDIR)/redwbg.o: redwbg.c
	cc -c redwbg.c $(CFLAGS)
	mv redwbg.o $(OBJDIR)

$(OBJDIR)/redwfg.o: redwfg.c
	cc -c redwfg.c $(CFLAGS)
	mv redwfg.o $(OBJDIR)

$(OBJDIR)/sel_disp.o: sel_disp.c
	cc -c sel_disp.c $(CFLAGS)
	mv sel_disp.o $(OBJDIR)

$(OBJDIR)/set_cmap.o: set_cmap.c
	cc -c set_cmap.c $(CFLAGS)
```

```
	mv set_cmap.o $(OBJDIR)

$(OBJDIR)/set_gc.o: set_gc.c
	cc -c set_gc.c $(CFLAGS)
	mv set_gc.o $(OBJDIR)

$(OBJDIR)/set_label.o: set_label.c
	cc -c set_label.c $(CFLAGS)
	mv set_label.o $(OBJDIR)

$(OBJDIR)/set_timer.o: set_timer.c
	cc -c set_timer.c $(CFLAGS)
	mv set_timer.o $(OBJDIR)

$(OBJDIR)/shm_creat.o: shm_creat.c
	cc -c shm_creat.c $(CFLAGS)
	mv shm_creat.o $(OBJDIR)

$(OBJDIR)/sort_msid.o: sort_msid.c
	cc -c sort_msid.c $(CFLAGS)
	mv sort_msid.o $(OBJDIR)

$(OBJDIR)/stat_col.o: stat_col.c
	cc -c stat_col.c $(CFLAGS)
	mv stat_col.o $(OBJDIR)

$(OBJDIR)/test.o: test.c
	cc -c test.c $(CFLAGS)
	mv test.o $(OBJDIR)

$(OBJDIR)/tick_mk.o: tick_mk.c
	cc -c tick_mk.c $(CFLAGS)
	mv tick_mk.o $(OBJDIR)

$(OBJDIR)/time_val.o: time_val.c
	cc -c time_val.c $(CFLAGS)
	mv time_val.o $(OBJDIR)

$(OBJDIR)/tmr_update.o: tmr_update.c
	cc -c tmr_update.c $(CFLAGS)
	mv tmr_update.o $(OBJDIR)

$(OBJDIR)/ui_init.o: ui_init.c
	cc -c ui_init.c $(CFLAGS)
	mv ui_init.o $(OBJDIR)

$(OBJDIR)/unlatch.o: unlatch.c
	cc -c unlatch.c $(CFLAGS)
	mv unlatch.o $(OBJDIR)

$(OBJDIR)/unv_plot.o: unv_plot.c
	cc -c unv_plot.c $(CFLAGS) -DDEBUG
	mv unv_plot.o $(OBJDIR)

$(OBJDIR)/upd_rate.o: upd_rate.c
	cc -c upd_rate.c $(CFLAGS)
	mv upd_rate.o $(OBJDIR)

$(OBJDIR)/update.o: update.c
	cc -c update.c $(CFLAGS)
	mv update.o $(OBJDIR)

$(OBJDIR)/updtbg.o: updtbg.c
	cc -c updtbg.c $(CFLAGS)
```

```
	mv updtbg.o $(OBJDIR)

$(OBJDIR)/updtfg.o: updtfg.c
	cc -c updtfg.c $(CFLAGS)
	mv updtfg.o $(OBJDIR)

$(OBJDIR)/updtht.o: updtht.c
	cc -c updtht.c $(CFLAGS)
	mv updtht.o $(OBJDIR)

$(OBJDIR)/val_dt.o: val_dt.c
	cc -c val_dt.c $(CFLAGS)
	mv val_dt.o $(OBJDIR)

$(OBJDIR)/val_fn.o: val_fn.c
	cc -c val_fn.c $(CFLAGS)
	mv val_fn.o $(OBJDIR)

$(OBJDIR)/val_msid.o: val_msid.c
	cc -c val_msid.c $(CFLAGS)
	mv val_msid.o $(OBJDIR)

$(OBJDIR)/val_ppl.o: val_ppl.c
	cc -c val_ppl.c $(CFLAGS)
	mv val_ppl.o $(OBJDIR)

$(OBJDIR)/val_src.o: val_src.c
	cc -c val_src.c $(CFLAGS)
	mv val_src.o $(OBJDIR)

$(OBJDIR)/valmsid.o: valmsid.c
	cc -c valmsid.c $(CFLAGS)
	mv valmsid.o $(OBJDIR)

$(OBJDIR)/wex.o: wex.c
	cc -c wex.c $(CFLAGS)
	mv wex.o $(OBJDIR)

$(OBJDIR)/zoom.o: zoom.c
	cc -c zoom.c $(CFLAGS)
	mv zoom.o $(OBJDIR)
```

```
/*****************************************************************************
 * MODULE NAME: DDdisp.h
 *
 *  This is the main header file for the functions which are responsible for
 *  data display.
 *
 *  This file specifies integer type for most coordinates instead of float.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Tod Milam       - Ford Aerospace Corporation/Houston
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Ronnie Killough - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <X11/Xlib.h>


#define START_CIRCLE    0
#define FULL_CIRCLE     (360*64)

#define min(a,b) ((a<b) ? a : b )
#define max(a,b) ((a<b) ? b : a )

/*
 * Array of X Windows Graphics Contexts used to change colors/fonts efficiently
 */

#define MAX_GC  5

int GC_Index[129];                      /* indexes color # (0..128) to FGC */

/*
 *  This structure contains information for the Displayer task when activated.
 *  This information is used by the Displayer task within a workstation.
 */

struct data_info
{
  long buffer[2];                   /* Offset to the data buffers.           */
  long decom_buf;                   /* Offset to the decom buffer for data.  */
  short buf_ready;                  /* Flag set to either 0 or 1 depending on
                                       which data buffer has newest data    */
  short decom_in_use[MAX_DISP];     /* Flags set by the Displayer task when  */
                                    /* accessing the decom buffer            */
  short need_decom;                 /* set by the DH when an update needs to */
                                    /* be made to the decom buffer           */
  short nbr_msids[MAX_DISP];        /* total nbr MSID's per display          */
  short msid_index[MAX_DISP][MAX_MSIDS];
                                    /* indices for each possible display     */
  long spare;                       /* full word alignment                   */
};

struct shm_decom
{
  int length;                      /* length of one sample value w/o status  */
```

```
  int size;                    /* length of all status and values          */
  int offset;                  /* offset to data value parameter block     */
  short num_samps;             /* number of samples returned               */
  char  attribute;             /* parameter attribute                      */
  char  error;                 /* error indicator                          */
  short sample_size;           /* length of one sample with status word    */
  short spare;                 /* full word alignment                      */
};

/*
 *  Union used for extracting data from the decom buffers.
 */

union p_data {
   double ddata;
   float sfdata[2];
   long sldata[2];
   short ssdata[4];
   unsigned long ldata[2];
   unsigned long uldata[2];
   unsigned short usdata[4];
};

/*
 *  Structure to be used in reading the display directory and then placing.
 *  on the display.
 */

   struct file_header          /* DDF file header structure                */
     {
      short version;           /* display builder software release version */
      char disp_name[33];      /* ASCII title of the display               */
      char position[17];       /* Console position for display             */
      float x_size;            /* x-size of display                        */
      float y_size;            /* y-size of display                        */
      long s_color;            /* screen background color                  */
      long graph_num;          /* number of graphical records              */
      long char_num;           /* number of character records              */
      long subd_num;           /* number of subdrawing records             */
      char access_rs;          /* access restriction code                  */
      char spare[10];
     };

   struct rec_header           /* character record header                  */
     {
      long color;              /* color for the record item                */
      int x_position;          /* starting x coordinate                    */
      int y_position;          /* starting y coordinate                    */
      long char_len;           /* length in bytes of the item              */
      char *record_item;       /* character string to display              */
      Font  font_num;          /* font number for character fonts          */
      short   redraw_flag;     /* flag set if text needs to be redrawn     */
     };

   struct msid_ent             /* msid entry structure                     */
     {
      long MSID_Entr;          /* msid entry number                        */
      char MSID[11];           /* msid name                                */
      long Sample;             /* sample number within msid                */
      char Data_Src[4];        /* data source of msid                      */
      short Scrn_Type;         /* screen data type                         */
      long Nom_Color;          /* nominal parameter color                  */
      long Sta_Color;          /* static parameter color                   */
      long Ovr_Color;          /* override parameter color                 */
```

```
    long Dead_Color;          /* dead parameter color                          */
    float X_NDC_St;           /* status character x color                      */
    float Y_NDC_St;           /* status character y color                      */
    short Stat_Flag;          /* status character flag                         */
    long Tab_Index;           /* tabular entry index                           */
    long Txt_Index;           /* multilevel text entry index                   */
    long PBI_Indx;            /* pbi entry index                               */
    long hist_ind;            /* history tab index                             */
    long Limit_Ind;           /* limit sense index                             */
    long data_ind;            /* index into the Old_Data array for redraw      */
    short ddd0_latch;         /* YES, if msid is latched on a zero value       */
    short ddd1_latch;         /* YES, if msid is latched on a one value        */
    int Wid_Ind;              /* Index into Widget Data Buffer                 */
   };

 struct tabular_ent           /* tabular entry structure                       */
   {
    long Tab_Entry;           /* tabular entry number                          */
    long Data_Width;          /* data field width                              */
    short Dig_Right;          /* digits right of the decimal                   */
    short Just_Flag;          /*. truncate/justification flag                  */
    int   X_XC;               /* starting x coordinate                         */
    int   Y_XC;               /* starting y coordinate                         */
    Font  font_num;           /* font style                                    */
    short redraw_flag;        /* flag set if entry needs to be rerdawn         */
   };

 struct hist_tab
   {
    long  htab_entr;          /* history tab entry number                      */
    int   time_cntr;          /* sequence number of history tab entry          */
    char  llimit_flag;        /* low limit flag                                */
    char  ulimit_flag;        /* high limit flag                               */
    long  msid_index;         /* index to msid record                          */
    char  file_name[14];      /* history tab file name                         */
    char  *value;             /* current value of history tab                  */
    struct shm_decom decom_ent;/* decom entry for history tab                  */
    struct hist_tab *next_ptr;/* pointer to next history tab                   */
   };

 struct ht_files
   {
    char file_name[14];       /* history tab file name                         */
    FILE *file_ptr;           /* file open pointer                             */
    int num_entries;          /* number of msid entries in file                */
    struct hist_tab *ht_rec_ptr;/* pointer to history tab structure            */
    struct ht_files *next_ptr;/* pointer to next file entry                    */
   };

 struct ddd_ent
   {
     short  ddd_entr;         /* ddd entry number                              */
     long   zero_val_cor;     /* color indicating 0 state                      */
     long   one_val_cor;      /* color indicating 1 state                      */
     short  zero_locked;      /* 0 state locked                                */
     short  one_locked;       /* 1 state locked                                */
     int    ddd_msids;        /* nbr ddd appended to icon                      */
     short  *ddd_app_ptr;     /* ptr to msid appended to ddd                   */
   };

  struct ddd
  {
    double msid_app_ind;      /* x coordinate of object                        */
  };
```

```
struct label_ent
  {
   int   lbl_entr;          /* label entry number                          */
   long  lbl_color;         /* color of text for label                     */
   short strt_x_pos;        /* label descriptor x position                 */
   short strt_y_pos;        /* label descriptor y position                 */
   int   num_labels;        /* nbr of lines used for label                 */
   int   label_len;         /* label length                                */
   char *label;             /* lbl ptr to value/lbl structure              */
   Font  font_num;          /* font number for labels                      */
  };

struct scale_ent
  {
    short  scale_entr;      /* scale entry number                          */
    char   axis_type;       /* axis type (cartesion, etc..)                */
    char   scale_type;      /* scale type (time / number)                  */
    double low_scale;       /* low scale value                             */
    double high_scale;      /* high scale value                            */
    float  low_scale_x;     /* scale starting x position                   */
    float  low_scale_y;     /* scale starting y position                   */
    float  high_scale_x;    /* scale ending x position                     */
    float  high_scale_y;    /* scale ending y position                     */
    float  scale_x_diff;    /* scale coordinate difference x position      */
    float  scale_y_diff;    /* scale coordinate difference y position      */
    float  msid_scale_range;/* scale msid range                            */
  };

struct limit_ent            /* limit sensing structure                     */
  {
   long Limt_Entr;          /* limit entry number                          */
   short Limt_Flag;         /* limit sense flag                            */
   double Low_Limit;        /* operational low limit value                 */
   double Hi_Limit;         /* operational high limit value                */
   long Lo_Color;           /* low limit color                             */
   long Hi_Color;           /* high limit color                            */
   short Crit_Flag;         /* critical limit check flag                   */
   double Crit_Low;         /* critical low value                          */
   double Crit_Hi;          /* critical high value                         */
   long Cr_Lcolor;          /* critical low limit color                    */
   long Cr_Hcolor;          /* critical high limit color                   */
  };

struct mtext_ent            /* multilevel text entry structure             */
  {
   long Mult_Entr;          /* multilevel text entry number                */
   short Num_Values;        /* number of value/text pairs                  */
   char Def_Text[7];        /* default text value                          */
   struct val_txt *text_ptr;/* text ptr to value/text structure            */
  };

struct val_txt              /* value/text structure for multilevel text    */
  {
   long Value;              /* incoming value                              */
   char Text[7];            /* text string to display                      */
  };

struct fg_file_header /* DDF foreground file header structure    */
  {
   short Version;           /* display builder version                     */
   char Disp_Name[33];      /* ASCII title of the display                  */
   char Position[17];       /* console position for display                */
   char spare3[2];
```

```
        float X_Size;              /* x-size of display                    */
        float Y_Size;              /* y-size of display                    */
        long S_Color;              /* screen color                         */
        long Tab_Num;              /* number of tabular entries            */
        long Entry_Num;            /* number of msid entries               */
        long PBI_Num;              /* number of pbi entries                */
        long Icon_Num;             /* number of icon entries               */
        long Tmplt_Num;            /* number of template entries           */
        long Htab_Num;             /* number of history tab entries        */
        long Mltxt_Num;            /* number of multilevel entries         */
        long Limit_Num;            /* number of limit entries              */
        char Access_Rs[2];         /* access restriction code              */
        char Spare[20];
       };

/*
 *  Structure containing the information representing the different types of
 *  graphical records used by the Displayer Task background
 */

struct graph_record                /* graphical record type structure      */
{
  short   graph_typ;               /* type of graphical record             */
  char   *graph_ptr;               /* graphical record type ptr            */
  short   redraw_flag;             /* flag set if graphic needs to be redrawn  */
};

struct line_record                 /* graphical line record structure      */
{
  int    graph_col;                /* object color                         */
  short  line_type;                /* line type of object                  */
  float  line_wdth;                /* line width of object                 */
  int  point1_x;                   /* x coordinate of first point          */
  int  point1_y;                   /* y coordinate of first point          */
  int  point2_x;                   /* x coordinate of second point         */
  int  point2_y;                   /* y coordinate of second point         */
};

struct rectangle_record            /* graphical rectangle record structure */
{
  int    graph_col;                /* object color                         */
  short  line_type;                /* line type of object                  */
  float  line_wdth;                /* line width of object                 */
  short  pat_type;                 /* pattern type                         */
  float  pat_sizex;                /* pattern x size                       */
  float  pat_sizey;                /* pattern y size                       */
  int  ul_x;                       /* lower left x coordinate              */
  int  height;                     /* height of rectangle in pixels        */
  int  width;                      /* width of rectangle in pixels         */
  int  ul_y;                       /* lower upper rigth y coordinate       */
};

struct polygon_record              /* graphical polygon record structure   */
{
  int    graph_col;                /* object color                         */
  short  line_type;                /* line type of object                  */
  float  line_wdth;                /* line width of object                 */
  short  pat_type;                 /* pattern type                         */
  float  pat_sizex;                /* pattern x size                       */
  float  pat_sizey;                /* pattern y size                       */
  int    nmbr_pts;                 /* number of points                     */
  struct graph_pts *poly_pts_ptr;
                                   /* ptr to x and y coord. pts            */
};
```

```
struct curve_record             /* graphical polygon record structure        */
{
  int    graph_col;             /* object color                              */
  short  line_type;             /* line type of object                       */
  float  line_wdth;             /* line width of object                      */
  int    nmbr_pts;              /* number of points                          */
  struct graph_pts *curve_pts_ptr;
                                /* ptr to x and y coord. pts                 */
};

struct graph_pts                /* graphical points struct for poly and cur. */
{
  int point_x;                  /* x coordinate of object                    */
  int point_y;                  /* y coordinate of object                    */
};

struct circle_record            /* circle record structure                   */
{
  int    graph_col;             /* object color                              */
  short  line_type;             /* line type of object                       */
  float  line_wdth;             /* line width of object                      */
  short  pat_type;              /* pattern type                              */
  float  pat_sizex;             /* pattern x size                            */
  float  pat_sizey;             /* pattern y size                            */
  int  bb_x;                    /* upper left x coordinate of bounding box   */
  int  bb_y;                    /* upper left y coordinate of bounding box   */
  float  radius;                /* circle radius                             */
};

struct arc_record               /* arc record structure                      */
{
  int    graph_col;             /* object color                              */
  short  line_type;             /* line type of object                       */
  float  line_wdth;             /* line width of object                      */
  short  pat_type;              /* pattern type                              */
  float  pat_sizex;             /* pattern x size                            */
  float  pat_sizey;             /* pattern y size                            */
  int  bb_x;                    /* upper left x coordinate of bounding box   */
  int  bb_y;                    /* upper left y coordinate of bounding box   */
  int  maj_axis;                /* semi-major axis                           */
  int  min_axis;                /* semi-minor axis                           */
  int angle1;                   /* angle 1 in 64th degrees                   */
  int angle2;                   /* angle 2 in 64th degrees                   */
};

struct ellipse_record           /* ellipse record structure                  */
{
  int    graph_col;             /* object color                              */
  short  line_type;             /* line type of object                       */
  float  line_wdth;             /* line width of object                      */
  short  pat_type;              /* pattern type                              */
  float  pat_sizex;             /* pattern x size                            */
  float  pat_sizey;             /* pattern y size                            */
  int  bb_x;                    /* x coordinate of center                    */
  int  bb_y;                    /* y coordinate of center                    */
  int  maj_axis;                /* semi-major axis of ellipse                */
  int  min_axis;                /* semi-minor axis of ellipse                */
};

struct ell_arc_record           /* elliptical arc record structure           */
{
  int    graph_col;             /* object color                              */
  short  line_type;             /* line type of object                       */
```

```
   float  line_wdth;            /* line width of object                    */
   short  pat_type;             /* pattern type                            */
   float  pat_sizex;            /* pattern x size                          */
   float  pat_sizey;            /* pattern y size                          */
   int  center_x;               /* x coordinate of center                  */
   int  center_y;               /* y coordinate of center                  */
   float  maj_axis;             /* semi-major axis of ellipse              */
   float  min_axis;             /* semi-minor axis of ellipse              */
   float  angle1;               /* angle 1 in radians                      */
   float  angle2;               /* angle 2 in radians                      */
};

struct vtext_record             /* vector text record header               */
{
  int    graph_col;             /* object color                            */
  Font   font_num;              /* X font number                           */
  int    font_style;            /* font style for the item                 */
  float  vert_size;             /* vertical font size                      */
  float  char_width;            /* character font width                    */
  float  char_spac;             /* spacing between characters              */
  int    char_angl;             /* angle to write characters               */
  int    x_position;            /* starting x coordinate                   */
  int    y_position;            /* starting y coordinate                   */
  long   char_len;              /* length in bytes of the item             */
  char   *record_item;          /* character string to display             */
};

struct subd_records             /* subdrawing records structure            */
{
  long   subd_num;              /* subdrawing entry number                 */
  int  ul_x;                    /* lower left x coordinate                 */
  int  lr_y;                    /* lower left y coordinate                 */
  int  lr_x;                    /* upper right x coordinate                */
  int  ul_y;                    /* upper right y coordinate                */
  char   subd_fname[50];        /* subdrawing file name                    */
};

struct hdr_info                 /* background header file information      */
{
  long   graph_num;             /* number of graphical records             */
  long   char_num;              /* number of character records             */
  long   subd_num;              /* number of subdrawing records            */
  long   s_color;               /* color of background screen              */
};

struct bg_recs
{
  short graph_num;              /* number of graphical records             */
  struct graph_record *graph_rec;
                                /* ptr to the graphical records            */
  short char_num;               /* number of character records             */
  struct rec_header *record;    /* ptr to the character records            */
  short subd_num;               /* number of subdrawing files              */
  int total_nbr_records;        /* total nbr of records for that background */
  long s_color;                 /* background screen color                 */
  struct subd_records *subd_rec;
                                /* ptr to subdrawing information           */
  struct bg_recs *next_rec;     /* ptr to the next bg record information   */
};

struct label_indices
{
   short index;
};
```

```
struct pbi_ent
{
   short    entry_num;
   int      grph_indx;
   long     grph_color;
   int      ddd_indx;
   int      num_labels;
   struct   label_indices *label_ptr;
};
```

```
/*****************************************************************************
 * MODULE NAME: DDfg_graph.h
 *
 *  This is the header file which defines the foreground graphic types used by
 *  the data display functions.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Tod Milam       - Ford Aerospace Corporation/Houston
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Ronnie Killough - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


/*
 *  Structure containing the information representing the different types of
 *  foreground graphical records used by the Displayer Task.
 */

struct fg_line_rec              /* graphical line record structure          */
{
  short  line_type;             /* line type of object                      */
  float  line_wdth;             /* line width of object                     */
  float  point1_x;              /* x world coordinate of first point        */
  float  point1_y;              /* y world coordinate of first point        */
  float  point2_x;              /* x world coordinate of second point       */
  float  point2_y;              /* y world coordinate of second point       */
  int    msid1_x;               /* msid x coordinate for 1st point          */
  int    msid1_y;               /* msid y coordinate for 1st point          */
  int    msid2_x;               /* msid x coordinate for 2nd point          */
  int    msid2_y;               /* msid y coordinate for 2nd point          */
  int    scale_ind1;            /* index for appending scales               */
  int    scale_ind2;            /* index for appending scales               */
  int    scale_ind3;            /* index for appending scales               */
  int    scale_ind4;            /* index for appending scales               */
  int    ddd_ind;               /* index for appending DDDs                 */
  int    scale_ind;             /* scale nbr thats not used                 */
  int    pbi_ind;               /* index for pbi indicator                  */
  int    label_num;             /* number of label entries                  */
  struct label_index  *line_lbl_ptr;
                                /* pointer to label entry index             */
  int    rev_video;             /* flag for label reverse video             */
  int    rot_ind;               /* index to rotate msid                     */
  int    vis_ind;               /* index to visible msid                    */
  long   cur_color;             /* save the most recent color value         */
  XPoint points[2];             /* save the most recent values for redraw   */
};

struct fg_rectangle_rec         /* graphical rectangle record structure     */
{
  short  line_type;             /* line type of object                      */
  float  line_wdth;             /* line width of object                     */
  short  pat_type;              /* pattern type                             */
  float  pat_sizex;             /* pattern x size                           */
  float  pat_sizey;             /* pattern y size                           */
  float  ul_x;                  /* upper left x world coordinate            */
  float  lr_y;                  /* lower right y world coordinat            */
```

```
  float  lr_x;                  /* lower right x world coordinat          */
  float  ul_y;                  /* upper left y world coordinate          */
  int    msid_ul_x;             /* msid x coordinate for upper left       */
  int    msid_lr_y;             /* msid y coordinate for lower right      */
  int    msid_lr_x;             /* msid x coordinate for lower right      */
  int    msid_ul_y;             /* msid y coordinate for upper left       */
  int    scale_ind1;            /* index for appending scales             */
  int    scale_ind2;            /* index for appending scales             */
  int    scale_ind3;            /* index for appending scales             */
  int    scale_ind4;            /* index for appending scales             */
  int    ddd_ind;               /* index for appending DDDs               */
  int    scale_ind;             /* index for appending scale              */
  int    pbi_ind;               /* index for pbi indicator                */
  int    label_num;             /* number of label entries                */
  struct label_index  *rect_lbl_ptr;
                                /* pointer to label entry index           */
  int    rev_video;             /* flag for label reverse video           */
  int    vis_ind;               /* index to visible msid                  */
  long   cur_color;             /* save the most recent color value       */
  XPoint rect;                  /* current pixel coordinates for redraw   */
  short  width, height;         /* current width/height of rectangle      */
};

struct fg_polygon_rec           /* graphical polygon record structure     */
{
  short  line_type;             /* line type of object                    */
  float  line_wdth;             /* line width of object                   */
  short  pat_type;              /* pattern type                           */
  float  pat_sizex;             /* pattern x size                         */
  float  pat_sizey;             /* pattern y size                         */
  long   fnmbr_pts;             /* number of fixed points                 */
  long   mnmbr_pts;             /* number of points                       */
  struct fg_graph_pts *poly_pts_ptr;
                                /* ptr to x and y coord. pts              */
  struct msid_index   *msid_ind_ptr;
                                /* ptr to msid indexes                    */
  struct scale_index  *poly_scale_ptr;
                                /* ptr to index for appending scales      */
  int    ddd_ind;               /* index for appending DDDs               */
  int    scale_ind;             /* index for appending scales             */
  int    pbi_ind;               /* index for pbi indicator                */
  int    label_num;             /* number of label entries                */
  struct label_index  *poly_lbl_ptr;
                                /* pointer to label entry index           */
  int    rev_video;             /* flag for label reverse video           */
  int    rot_ind;               /* index to rotate msid                   */
  int    vis_ind;               /* index to visible msid                  */
  long   cur_color;             /* save the most recent color value       */
  XPoint points[100];           /* save the values for redraw             */
};

struct fg_curve_rec             /* graphical polygon record structure     */
{
  short  line_type;             /* line type of object                    */
  float  line_wdth;             /* line width of object                   */
  short  pat_type;              /* pattern type                           */
  long   fnmbr_pts;             /* number of fixed points                 */
  long   mnmbr_pts;             /* number of msid points                  */
  struct fg_graph_pts *cur_pts_ptr; /* ptr to x and y coord. pts          */
  struct msid_index   *msid_ind_ptr; /* ptr to msid index                 */
  struct scale_index  *cur_scale_ptr; /* ptr to index for appending scales */
  int    ddd_ind;               /* index for appending DDDs               */
  int    scale_ind;             /* index for appending scales             */
  int    pbi_ind;               /* index for pbi indicator                */
```

```
    int     label_num;              /* number of label entries                  */
    struct label_index   *cur_lbl_ptr;   /* pointer to label entry index        */
    int     rev_video;              /* flag for label reverse video             */
    int     vis_ind;                /* index to visible msid                    */
    long    cur_color;              /* save the most recent color value         */
    XPoint points[100];             /* save the values for redraw               */
};

struct fg_circle_rec                /* circle record structure                  */
{
    short  line_type;               /* line type of object                      */
    float  line_wdth;               /* line width of object                     */
    short  pat_type;                /* pattern type                             */
    float  pat_sizex;               /* pattern x size                           */
    float  pat_sizey;               /* pattern y size                           */
    float  center_x;                /* x coordinate of center                   */
    float  center_y;                /* y coordinate of center                   */
    float  radius;                  /* circle radius                            */
    int    msid_cen_x;              /* msid x coordinate for center point       */
    int    msid_cen_y;              /* msid y coordinate for center point       */
    int    msid_radius;             /* msid index for radius                    */
    int    scale_ind1;
    int    scale_ind2;              /* index for appending scale                */
    int    scale_ind3;              /* index for appending scale                */
    int    ddd_ind;                 /* index for appending DDDs                 */
    int    scale_ind;               /* index to scale record                    */
    int    pbi_ind;                 /* index for pbi indicator                  */
    int    label_num;               /* number of label entries                  */
    struct label_index  *cir_lbl_ptr;    /* pointer to label entry index       */
    int    rev_video;               /* flag for label reverse video             */
    int    vis_ind;                 /* index to visible msid                    */
    long   cur_color;               /* save the most recent color value         */
    float  cur_rad;                 /* save for redraw                          */
    XPoint bb;                      /* save for redraw                          */
};

struct fg_arc_rec                   /* arc record structure                     */
{
    short  line_type;               /* line type of object                      */
    float  line_wdth;               /* line width of object                     */
    short  pat_type;                /* pattern type                             */
    float  pat_sizex;               /* pattern x size                           */
    float  pat_sizey;               /* pattern y size                           */
    float  center_x;                /* x coordinate of center                   */
    float  center_y;                /* y coordinate of center                   */
    double  angle1;                 /* angle 1 in radians                       */
    double  angle2;                 /* angle 2 in radians                       */
    float  maj_axis;                /* major axis length                        */
    float  min_axis;                /* minor axis length                        */
    int    msid_cen_x;              /* msid x coordinate for center point       */
    int    msid_cen_y;              /* msid y coordinate for center point       */
    int    msid_ang1;               /* msid for angle 1                         */
    int    msid_ang2;               /* msid for angle 2                         */
    int    msid_maj;                /* msid x for major axis                    */
    int    msid_min;                /* msid y for minor axis                    */
    int    scale_ind1;              /* index for appending scale                */
    int    scale_ind2;              /* index for appending scale                */
    int    scale_ind3;              /* index for appending scale                */
    int    scale_ind4;              /* index for appending scale                */
    int    scale_ind5;              /* index for appending scale                */
    int    scale_ind6;              /* index for appending scale                */
    int    ddd_ind;                 /* index to DDD record                      */
    int    scale_ind;               /* index to scale record                    */
    int    pbi_ind;                 /* index for pbi indicator                  */
```

```
  int     label_num;              /* number of label entries                    */
  struct label_index  *arc_lbl_ptr;          /* pointer to label entry index*/
  int     rev_video;              /* flag for label reverse video               */
  int     rot_ind;                /* index to rotate msid                       */
  int     vis_ind;                /* index to visible msid                      */
  long    cur_color;              /* save the most recent color value           */
  short   smajor;                 /* save for redraw                            */
  short   sminor;                 /* save for redraw                            */
  XPoint  center;                 /* save for redraw                            */
  float   cur_angl;               /* save for redraw                            */
  float   cur_ang2;               /* save for redraw                            */
};

struct fg_ellipse_rec             /* ellipse record structure                   */
{
  short   line_type;              /* line type of object                        */
  float   line_wdth;              /* line width of object                       */
  short   pat_type;               /* pattern type                               */
  float   pat_sizex;              /* pattern x size                             */
  float   pat_sizey;              /* pattern y size                             */
  float   center_x;               /* x coordinate of center                     */
  float   center_y;               /* y coordinate of center                     */
  float   maj_axis;               /* semi-major axis of ellipse                 */
  float   min_axis;               /* semi-minor axis of ellipse                 */
  int     msid_cen_x;            /* msid x coordinate for center point         */
  int     msid_cen_y;            /* msid y coordinate for center point         */
  int     msid_len;              /* msid coordinate for length                 */
  int     msid_hgh;              /* msid coordinate for heigth                 */
  int     scale_ind1;            /* index for appending scale                  */
  int     scale_ind2;            /* index for appending scale                  */
  int     scale_ind3;            /* index for appending scale                  */
  int     scale_ind4;            /* index for appending scale                  */
  int     ddd_ind;               /* index to DDD record                        */
  int     scale_ind;             /* index to scale record                      */
  int     pbi_ind;                /* index for pbi indicator                   */
  int     label_num;             /* number of label entries                    */
  struct label_index   *ell_lbl_ptr;  /* pointer to label entry index          */
  int     rev_video;              /* flag for label reverse video               */
  int     vis_ind;               /* state of visibility                        */
  long    cur_color;              /* save the most recent color value          */
  float   smajor;                 /* save for redraw                            */
  float   sminor;                 /* save for redraw                            */
  XPoint  center;                 /* save for redraw                            */
};


struct fg_piechrt_rec
{
  long    def_col;                /* default color for pie volume               */
  short   pat_type;               /* pattern type                               */
  float   pat_sizex;              /* pattern x size                             */
  float   pat_sizey;              /* pattern y size                             */
  float   center_x;               /* x coordinate of center                     */
  float   center_y;               /* y coordinate of center                     */
  float   radius;                 /* radius of pie chart                        */
  short   sum_flag;               /* pie sum of volume flag                     */
  double  sum_pie;                /* pie sum of volume                          */
  int     num_msid;               /* number of msid appended                    */
  struct pie_msid_index  *pie_msid_ptr; /* pointer to msid for pie chart        */
  float   smajor;                 /* smajor for redraw of pie chart             */
  float   sminor;                 /* sminor for redraw of pie chart             */
  float   angle1;                 /* beginning angle of closing slice           */
  float   angle2;                 /* ending angle of closing slice              */
};
```

```
struct fg_clkmtr_rec
{
  short  line_type;              /* line type of object                    */
  float  line_wdth;              /* line width of object                   */
  long   clkmtr_col;             /* clock/meter color for dial             */
  short  pat_type;               /* pattern type                           */
  float  pat_sizex;              /* pattern x size                         */
  float  pat_sizey;              /* pattern y size                         */
  float  center_x;               /* x coordinate of center                 */
  float  center_y;               /* y coordinate of center                 */
  float  radius;                 /* radius of clock/meter chart            */
  double angle_1;                /* starting angle of the clock/meter      */
  double angle_2;                /* ending angle of the clock/meter        */
  double angle_diff;
  int    num_msid;               /* number of msid                         */
  short  hand_type[10];          /* hand type                              */
  struct cm_msid_index   *clk_msid_ptr; /* pointer to msid appended to clock*/
  struct scale_index  *clk_scale_ptr; /* ptr to index for appending scales  */
  int    label_num;              /* number of label entries                */
  struct label_index  *clk_lbl_ptr;   /* pointer to label entry index       */
  int    rev_video;              /* flag for label reverse video           */
  short  init_draw;              /* initial pass of clock/meter            */
};

struct fg_bar_rec
{
  short  line_type;              /* line type of object                    */
  float  line_wdth;              /* line width of object                   */
  char   direction;              /* direction of movement                  */
  short  pat_type;               /* pattern type                           */
  float  pat_sizex;              /* pattern x size                         */
  float  pat_sizey;              /* pattern y size                         */
  float  ul_x;                   /* upper left x coordinate                */
  float  lr_y;                   /* lower right y coordinate               */
  float  lr_x;                   /* lower right x coordinate               */
  float  ul_y;                   /* upper left y coordinate                */
  int    num_msid;               /* number of msid appended                */
  int    msid_indx;              /* index for msid to drive bar chart      */
  int    scale_indx;             /* ndex for appending scale               */
  int    label_num;              /* number of label entries                */
  struct label_index *bar_lbl_ptr;   /* pointer to label entry index       */
  int    rev_video;              /* flag for label reverse video           */
  long   cur_color;              /* save the most recent color value       */
};


struct fgr_record
{
  short graph_typ;
  int   graph_ent;
  char *graph_ptr;
  short redraw_flag;               /* flag set if graphic needs to be redrawn*/
};


struct fg_recs
{
  long   graph_num;                /* number of graphical records           */
  struct fgr_record *graph_rec;    /* ptr to graphical records              */
};


struct fg_graph_pts
```

```
{
  float  point_x;                 /* x coordinate of object          */
  float  point_y;                 /* y coordinate of object          */
};

struct label_index
{
  short  label_ind[10];           /* label entry index               */
};

struct scale_index
{
  long scale_ind_num;             /* index for appending scales      */
};

struct msid_index
{
  long msid_ind;                  /* index for appending msids       */
};

struct cm_msid_index
{
  long msid_ind;                  /* index for appending msids       */
  long cur_color;
  XPoint end_pt;
};

struct pie_msid_index
{
  long msid_ind;                  /* index for appending msids       */
  long cur_color;
  float angle1;
  float angle2;
};
```

```
/*****************************************************************************
 * MODULE NAME: DDplot.h
 *
 *  This include file defines structures and constants needed for plot display.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Tod Milam       - Ford Aerospace Corporation/Houston
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Ronnie Killough - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


/*
 * These structures contain information for plot processing.
 */

struct plot_ptrs
{
  struct plot_tmplt *plot_pos;          /* ptr to the plot position         */
  double seconds_elapsed;               /* nbr secs since plot was started  */
  struct plot_hdr *header;              /* ptr to plot header               */
  struct axis_info *axis;               /* ptr to axis information          */
  struct msid_info *msids;              /* ptr to msid information          */
  struct lim_lines *nline;              /* ptr to nominal line information  */
  struct lim_lines *lline;              /* ptr to limit line information    */
  struct shm_decom *plt_decom;          /* ptr to plot decom buffer         */
  char *plot_data;                      /* ptr to plot data buffer          */
  char plot_name[DNAME_LEN];            /* plot file name                   */
  char plot_data_file[DNAME_LEN];       /* plot file name for data file     */
  char user_disp_name[DNAME_LEN];       /* plot file name with user-display */
  char plot_ovr[DNAME_LEN];             /* plot overlay file name           */
  Widget scrl_win;                      /* widget id for scrolling window   */
  Widget draw_win;                      /* widget id for drawing area       */
  short act_flg;                        /* 1 if plot is active              */
  short prev_act_flg;                   /* 1 if plot is or has been active  */
  short ovr_flg;                        /* 1 if plot has been overlayed     */
  int plot_fp;                          /* file pointer to plot data file   */
  int buf_size;                         /* size of data buffer              */
  short ovl_color_flg;                  /* overlay color change flag        */
  short redraw_flag;                    /* flag set if needs to be redrawn  */
};

/*
 *  Structure containing the information representing the plot definition
 *  files.
 */

struct plot_tmplt
{
    long    tmplt_entr;
    short   tmplt_type;
    short   bb_xul;
    short   bb_yul;
    short   bb_width;
    short   bb_height;
    short   drw_width;
```

```
    short   drw_height;
    short   offset_x;          /* current pixel offset for plot points */
    short   offset_y;          /*  (used to keep zoom focus).          */
    double  factor_x;          /* current world coordinate             */
    double  factor_y;          /*  transformation factors.             */
    double  org_factor_x;      /* original world coordinate            */
    double  org_factor_y;      /*  transformation factors.             */
    char    tmplt_nam[9];
};

struct plot_hdr
{
  short version;
  char  plot_titl[33];
  char  position[17];
  int   xaxes_num;
  int   yaxes_num;
  int   msid_num;
  int   actual_msids;
  int   nline_num;
  int   lline_num;
  short  upd_rate;
  short access_rs;
};

struct axis_info
{
  char   axis_xory;            /* specifies if an x axis or y axis          */
  int    axis_num;             /* numeric identifier...unique by x/y only   */
  short  axis_type;            /* cartesian, logarithmic, or polar          */
  char   scal_type;            /* N = number, T = Time                      */
  int    end_code;             /* what to do when time plot reaches axis    */
  short  axis_pos;             /* x or y position of the y or x axis        */
  short  pixel_axis_pos;       /* permanent pixel axis position             */
  short  cur_axis_pos;         /* current pixel axis pos (save for redraw)  */
  short  axis_col;             /* color of the axis                         */
  char   low_scale[15];        /* ascii version of original low scale       */
  char   high_scal[15];        /* ascii version of original low scale       */
  double low_value;            /* current low scale in float                */
  double high_value;           /* current high scale in float               */
  double org_low_val;          /* original low scale in float               */
  double org_high_val;         /* original high scale in float              */
  char   auto_flag;            /* auto scaling on enabled? YES/NO           */
  short  grad_vals;            /* # of graduations from low to high scale   */
  char   vis_flag;             /* is this axis visible?                     */
  char   grid_flag;            /* are grid lines parallel to this axis?     */
  short  grid_gran;            /* granularity of the grid lines             */
  short  grid_type;            /* line type of grid (solid, dashed, dot-dash*/
  short  grd_color;            /* color of the grid lines                   */
  short  maj_ticks;            /* # of major tick marks (incl. ends)        */
  short  min_ticks;            /* # of minor tick marks (excl. maj tick mks)*/
  short  end_of_plot;          /* end-of-plot flag for time plots           */
  short  auto_scale;           /* 1-update low scale, 2-update high scale   */
  double new_scale;            /* new scale value for rescale               */
  short axis_active;           /* YES to plot on axis                       */
  double scale_ratio;          /* ratio of plot length / scale units        */
  float logval;                /* log value for axis                        */
};

struct msid_info
{
  short  msid_indx;
  char   msid_name[11];
  short  sample;
```

```
    char   data_src[4];
    char   xory_axis;
    int    axis_num;
    char   plot_msid[11];      /* name of the msid pair */
    char   plot_type;
    int    line_type;
    float  line_width;
    char   plot_char[2];
    Font   plot_font;
    short  icon_indx;
    char   plot_conn;
    short  plot_color;
    int    stat_flag;
    int    miss_flag;
    short  stat_color;
    short  miss_color;
    short  ovl_color;
    short  limt_color;
    short  crit_color;
    int    oper_type;
    float  oper_width;
    int    crit_type;
    float  crit_width;
    struct msid_info *pair_ptr;
    short  first_pt;                  /* YES, if first point plotted     */
    short  prev_pt_x;                 /* last x coordinate in pixels     */
    short  prev_pt_y;                 /* last y coordinate in pixels     */
};

struct lim_lines
{
    char    line_type;
    short  line_color;
    int    xaxis_num;
    int    yaxis_num;
    char   line_def;
    short  point_num;
    short  polyn_num;
    struct plot_pts *plot_pts_ptr;
    char   coeff[6][15];
};

struct plot_pts                /* plot points struct for poly              */
{
    char  point_x[15];         /* x coordinate of object                   */
    char  point_y[15];         /* y coordinate of object                   */
};
```

```
/*****************************************************************************
 * MODULE NAME: constants.h
 *
 *  This file defines constants and structures needed by the Display Manager.
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


/*
 * Courier fonts
 */

#define R_08     "-adobe-courier-medium-r-normal--8-80-75-75-m-50-iso8859-1"
#define B_08     "-adobe-courier-bold-r-normal--8-80-75-75-m-50-iso8859-1"
#define I_08     "-adobe-courier-medium-o-normal--8-80-75-75-m-50-iso8859-1"
#define R_10     "-adobe-courier-medium-r-normal--10-100-75-75-m-60-iso8859-1"
#define B_10     "-adobe-courier-bold-r-normal--10-100-75-75-m-60-iso8859-1"
#define I_10     "-adobe-courier-medium-o-normal--10-100-75-75-m-60-iso8859-1"
#define R_12     "-adobe-courier-medium-r-normal--12-120-75-75-m-70-iso8859-1"
#define B_12     "-adobe-courier-bold-r-normal--12-120-75-75-m-70-iso8859-1"
#define I_12     "-adobe-courier-medium-o-normal--12-120-75-75-m-70-iso8859-1"
#define R_14     "-adobe-courier-medium-r-normal--14-140-75-75-m-90-iso8859-1"
#define B_14     "-adobe-courier-bold-r-normal--14-140-75-75-m-90-iso8859-1"
#define I_14     "-adobe-courier-medium-o-normal--14-140-75-75-m-90-iso8859-1"
#define R_18     "-adobe-courier-medium-r-normal--18-180-75-75-m-110-iso8859-1"
#define B_18     "-adobe-courier-bold-r-normal--18-180-75-75-m-110-iso8859-1"
#define I_18     "-adobe-courier-medium-o-normal--18-180-75-75-m-110-iso8859-1"
#define R_24     "-adobe-courier-medium-r-normal--24-240-75-75-m-150-iso8859-1"
#define B_24     "-adobe-courier-bold-r-normal--24-240-75-75-m-150-iso8859-1"
#define I_24     "-adobe-courier-medium-o-normal--24-240-75-75-m-150-iso8859-1"

/*
 * Masscomp fixed fonts
 */

#define R_3X5               "3x5"
#define B_3X5               "3x5"
#define I_3X5               "3x5"
#define R_5X7               "5x7"
#define B_5X7               "5x7_bold"
#define I_5X7               "5x7_italic"
#define R_7X9               "7x9"
#define B_7X9               "7x9_bold"
#define I_7X9               "7x9_italic"
#define R_9X11              "9x11"
#define B_9X11              "9x11_bold"
#define I_9X11              "9x11_italic"

#define MAX_COLORS          256
#define NUM_MOTIF_COLORS    16
#define CLR_SIZE            15
#define CLR_NUM             16
#define CLR_SPACE           3
#define CLR_TOTAL           ( ( CLR_SIZE + CLR_SPACE ) * CLR_NUM + CLR_SPACE )
#define CLR_WIDTH_RGB       308
#define CLR_LEFT_RGB        1
```

```
#define HELP_MAX            26
#define HELP_BUF_SIZE       5000
#define HELP_DIR            "/home/project/2984/db/dm/help/"

#define W_RED               1           /* WEX advisory color                    */
#define W_YELLOW            2           /* WEX advisory color                    */
#define W_GREEN             3           /* WEX advisory color                    */
#define W_BLUE              4           /* WEX advisory color                    */
#define W_WHITE             5           /* WEX advisory color                    */

#define NO_CHANGE           -1

#define BAD                 -1
#define GOOD                1
#define QT2                 1
#define YES                 1
#define NO                  0
#define START               1
#define STOP                0
#define ON                  1
#define OFF                 0
#define NONE                -1
#define INVALID             -1
#define ERROR               -1
#define SETUP_ERROR         1
#define VALID               0
#define CREAT_SHM           1
#define NO_CREAT            2
#define READY               1
#define NOT_READY           0
#define SCREEN_XSIZE        1152.0      /* Masscomp/Sun Screen X pixels          *
#define SCREEN_YSIZE        910.0       /* Masscomp/Sun Screen Y pixels          *
#define MAJOR_TICK_LEN      2.0
#define MINOR_TICK_LEN      1.0
#define SRC_NUM             1.0

#define MAX_DISP            4           /* max number of Displayer tasks per DM  */
#define MAX_DISPS           4           /* max number of Displayer tasks per DM  */
#define MAX_WINDOWS         14          /* windows per terminal                  */
#define DM_SHM_KEY          414         /* Display Manager shared memory key     */

#define DISP_NAME_LEN       10          /* max length of a display name          */
#define NO_PATH_DISP        8           /* max length of a display name with no path */
#define FLT_ID              4           /* length of the flight id               */

#define DISP_LOOP           30          /* loop cntr for Display Init check      */
#define DDH_LOOP            30          /* loop cntr for Data Handler Init check */
#define HALT_TIME           5           /* loop cntr for task halts              */

#define MAX_FLTS            2           /* max. number of flights                */
#define GDR_SOURCE          "GR"        /* pseudo source for GDR data sources    */

#define NBR_DATA_SRC        ( NUM_PPM + NUM_GDR + NUM_USR + 5 )
                            /* PPM, GDR, USR, PTM, NDM, MTM, 2 ENV src */
#define PRIV_DECOM          ( ( 15 * MAX_FLTS ) + 7 + MAX_DISP )
                            /* private decom - 13 data source statuses */
                            /* 2 entries for flight and data type      */
                            /* total 15 entries for each flight        */
                            /* 7 for traj count and 6 msgs.,           */
                            /* MAX_DISP disp rates                     *
#define TOTAL_DECOM         ( PRIV_DECOM + MAX_MSIDS )
                            /* total decom entries                     */

#define PR_MSID_OFF 3000            /* start of private msids in decom table */
```

```
#define ACT_NDM PR_MSID_OFF             /* active NDM data source flag                */
#define ACT_PTM ( ACT_NDM + MAX_FLTS )  /* active PTM data source flag                */
#define ACT_MTM ( ACT_PTM + MAX_FLTS )  /* active MTM data source flag                */
#define ACT_MOC ( ACT_MTM + MAX_FLTS )  /* active MOC data source flag                */
#define ACT_GDR ( ACT_MOC + MAX_FLTS )  /* active GDR data source flag                */
#define ACT_PPM ( ACT_GDR + MAX_FLTS )  /* active PPM data source flag                */
#define ACT_USR ( ACT_PPM + MAX_FLTS )  /* active USR data source flag                */

#define TRJ_MSG_LEN         20

#define HM_NDM      ( ACT_USR + MAX_FLTS )  /* homog. NDM data source flag            */
#define HM_PTM      ( HM_NDM + MAX_FLTS )   /* homog. PTM data source flag            */
#define HM_MTM      ( HM_PTM + MAX_FLTS )   /* homog. MTM data source flag            */
#define HM_MOC      ( HM_MTM + MAX_FLTS )   /* homog. MOC data source flag            */
#define HM_PPM      ( HM_MOC + MAX_FLTS )   /* homog. PPM data source flag            */
#define HM_USR      ( HM_PPM + MAX_FLTS )   /* homog. USR data source flag            */
#define TRAJ_CNT    ( HM_USR + MAX_FLTS )   /* trajectory message count               */
#define TRAJ_MSG1   ( TRAJ_CNT + 1 )        /* trajectory message 1                   */
#define TRAJ_MSG2   ( TRAJ_MSG1 + 1 )       /* trajectory message 2                   */
#define TRAJ_MSG3   ( TRAJ_MSG2 + 1 )       /* trajectory message 3                   */
#define TRAJ_MSG4   ( TRAJ_MSG3 + 1 )       /* trajectory message 4                   */
#define TRAJ_MSG5   ( TRAJ_MSG4 + 1 )       /* trajectory message 5                   */
#define TRAJ_MSG6   ( TRAJ_MSG5 + 1 )       /* trajectory message 6                   */
#define UPD_MSID    ( TRAJ_MSG6 + 1 )       /* start of update rates per display      */
#define FLT_MSID    ( UPD_MSID + MAX_DISP ) /* start of flight Id's                   */
#define STRM_MSID   ( FLT_MSID + MAX_FLTS ) /* start of stream type                   */
#define GDR_MSG1    0x80                    /* mask for traj. message 1 status        */
#define GDR_MSG2    0x10                    /* mask for traj. message 2 status        */
#define GDR_MSG3    0x08                    /* mask for traj. message 3 status        */
#define GDR_MSG4    0x04                    /* mask for traj. message 4 status        */
#define GDR_MSG5    0x02                    /* mask for traj. message 5 status        */
#define GDR_MSG6    0x01                    /* mask for traj. message 6 status        */

#define NEW_GDR             "255"       /* retrieve data on update only               */
#define ADD                 1           /* add a new msid                             */

#define START_DISPLAY       1           /* function code to start a display           */
#define START_PDISPLAY      2           /* function code to start a particular display */
#define CLEAR_DISPLAY       3           /* function number to clear a display         */
#define SCRN_DUMP           4           /* function number for a screen dump          */
#define MAIN_HELP           5           /* function number to main menu help          */
#define HALT_DISPLAY        6           /* function number to halt the Display Manager */
#define LIM_MENU            7           /* function number to bring up limit change mnu */
#define CHG_LIM             8           /* function number to change limits           */
#define DRAW_PF             9           /* function number to draw PF definitions     */
#define DRAW_MAIN           10          /* function number to draw main menu          */
#define UPD_RATE            11          /* function number to change display update   */
#define LIM_GRP             12          /* function number to change limit group      */
#define POS_ALARM           13          /* function number to turn/off w/s alarm      */
#define PLOT                14          /* function number to start/stop a plot       */
#define PLOT_OVRLAY         15          /* function number to overlay a plot          */
#define HIST_TAB            16          /* function number to process history tabs    */
#define ZOOM_DIS            17          /* function number for zooming the display    */
#define ZOOM_RES            18          /* function number for reseting the display   */
#define ZOOM_FAC            19          /* function number for changing the zoom factor */
#define LOGENABLE_DISPLAY   20          /* function number for display log enable     */
#define LOGDISABLE_DISPLAY  21          /* function number for display log disable    */
#define LOGENABLE_ALL       22          /* function number for all DM log enable      */
#define LOGDISABLE_ALL      23          /* function number for all DM log disable     */
#define GDR_GETNEXT         24          /* function number for GDR get next option    */
#define PBI_ENABLE          25          /* function number to enable PBIs             */
#define PBI_DISABLE         26          /* function number to disable PBIs            */
#define EXMSG_SEND          27          /* function number to send an EXmsg           */
#define DDD_UNLATCH         28          /* function number to unlatch DDDs            */
```

```
#define DDD_UNL_ALL          29          /* function number to unlatch DDDs                    */
#define SET_FLIGHT           30          /* function number used to save flight/datatype *
#define GDR_CHG              31          /* function number used to save change GDR
#define LIM_GRP_OFF          32          /* function number used to turn off limit group */
#define LIM_LIST             33          /* function number used to list limits                */
#define PLOT_OFF             34          /* function number used turn a plot off               */
#define PLOT_LIST            35          /* function number used list plots                    */
#define SAVE_OVRLAY          36          /* function number used save an overlay               */
#define PLOT_UNV             37          /* function number used to define a UNV plot          */
#define POS_ALARM_OFF        38          /* function number used to turn a POS alarm off */
#define DISPLAY              40          /* used by Data Handler for disp initialization */
#define MSG_ON               41          /* function number for turning on messages            */
#define MSG_OFF              42          /* function number for turning off messages           */
#define EDIT_COLORS          43          /* function number for editing the colors             */
#define FREEZE_DISPLAY       44          /* function number for freezing a display             */
#define RESTART_DISPLAY      45          /* function number for restarting a display           */
#define UNIX_COMMAND         90          /* function number to send a unix command             */

#define NEW_SCALE            1           /* update universal scale values only                 */
#define READ_PLOT            2           /* read the universal plot file in again              */

#define STRT_PLOT            1           /* action flag to start a plot                        */
#define STOP_PLOT            2           /* action flag to stop a plot                         */
#define SAVE_OVERLAY         3           /* action flag to save a plot overlay                 */
#define OVERLAY              4           /* action flag to overlay a plot                      */

#define MAX_PLOTS            20          /* w/s max. of active plots                           */
#define MAX_POS_ID           4           /* max. position Id's for a * workstation             */
#define PLOT_MSIDS           30          /* max msids to plot                                  */

#define PLOT_DECOM_SIZE      24          /* size of a plot decom entry                       ,

#define MEDICAL_USR          1
#define PAYLOAD_USR          2

#define OPS                  1           /* WEX Operational mode                               */
#define CERT                 2           /* WEX Certification mode                             */
#define DEV                  3           /* WEX Development mode                               */

#define TRIG_LIM_FILE        0x0001      /* mask to trigger a limit group change               */
#define TRIG_CRT_HIGH        0x0002      /* mask to trigger a limit group change               */
#define TRIG_CRT_LOW         0x0004      /* mask to trigger a limit group change               */
#define TRIG_OPS_HIGH        0x0008      /* mask to trigger a limit group change               */
#define TRIG_OPS_LOW         0x0010      /* mask to trigger a limit group change               */

#define DISCRETE             0x00000001  /* discrete data mask                                 */
#define NO_MSID              1           /* in limit change - no match found                   */
#define DISP_INFO            48          /* Display information size                           */
#define FILE_DESC            33          /* File desc size                                     */

#define MAX_AXES             3           /* maximum nbr of axes pairs                          */
#define TOTAL_AXES           6           /* maximum nbr total axes                             */
#define AXES_OFFSET ( MAX_AXES - 1 )     /* offset to axis information                         */

#define RR                   0x5252      /* ASCII                                              */
#define R1                   0x5231      /* ASCII                                              */
#define R2                   0x5232      /* ASCII                                              */
#define SR                   0x5352      /* ASCII                                              */
#define S1                   0x5331      /* ASCII                                              */
#define S2                   0x5332      /* ASCII                                              */

#define TWO_BLANKS           0x2020      /* ASCII                                              */

#define DNAME_LEN            41          /* max. name length for a display                     */
```

```
#define PPL_NAME_LEN        14          /* max. name length for a ppl file              */

#define UPD_ADJ             150         /* plot update rate adj. in ms.                 */

#define BYTE_MASK           0x0e700001  /* save bits 4, 5, 6, 9, 10, 11 of * status word*/
#define BYTE_CLR            0xf997fffd  /* clear bits 12 and 30 of the status * word    */
#define DATA_DIFF           0x10        /* set bit 11 of the status word - * data update*/
#define NO_DIFF             0xEF        /* clr bit 11 of the status word - * data update*/
#define LOW_LIM             0x02        /* set bit low bit of the status - * limit chk  */
#define HIGH_LIM            0x04        /* set bit high bit of the status - * limit chk */
#define CRIT_LOW            0x20        /* set bit low bit of the status - * limit chk  */
#define CRIT_HIGH           0x40        /* set bit high bit of the status - * limit chk */


#define MAX_MSIDS           3000        /* maximum nbr of msids to process in * a w/s   */
#define MAX_SAMPLES         3000        /* maximum nbr of samples to process * in a w/s */
#define STATUS_SIZE         4           /* size of the status word in the data          */
#define INVALID_ENTRY       0x00000020  /* invalid bit set in status field              */
#define NUM_USR             24          /* max. nbr of User Comp data sources * per w/s */
#define NUM_PPM             4           /* max. nbr of PPM data sources per * stream    */
#define NUM_GDR             6           /* max. nbr of GDR data sources per * w/s       */

#define TAB_ENT_SIZE        41          /* size of a tabular entry                      */
#define VAL_TXT_SIZE        12          /* size of a value text pair                    */
#define MUL_TXT_SKIP        14          /* amount to skip before the * value/text pair  */

#define ALL                 -1          /* number of MSIDs to retrieve                  */
#define LAST                0           /* retrieve the last sample                     */
#define UNLATCH_MSID        1           /* unlatch one DDD msid                         */

#define MSID_LENGTH         10          /* length of an MSID for string * compare       */
#define RETRY_COUNT         60          /* counter for a connect or getkeys * retry     */

#define EXTRA_BYTES         100         /* extra bytes added to size of data * buffer   */

#define UPDATE_RATE         850         /* time in ms. to wait to call * getparms       */

#define TRIG_ADJ            ( 1000.0/UPDATE_RATE )

#define UPDATE_WAIT         200         /* time in ms. to wait after a call to * chk_upd*/
#define DISP_WSID           48          /* Displayer workstation id                     */
#define MENU_WS_BASE        1           /* MENU workstation base Id                     */
#define STRM_LEN            2           /* length of the stream type                    */
#define DD_SHM_KEY          415         /* Displayer key number                         */
#define STOP_TIME           10          /* Displayer halt time for loop                 */
#define WAIT_CNT            10          /* number of times to loop waiting for flags to */

#define UNAVAIL_DATA        0x00000040  /* data is unavailable                          */
#define DEAD_DATA           0x40000000  /* Dead data mask for status bit 1              */
#define MISSING_DATA        0x80000000  /* Missing data mask for status bit 0           */
#define STATIC_DATA         0x08000000  /* Static data mask for status bit 4            */
#define HOMOG_DATA          0x01000000  /* homogeneous data mask for status * bit 7     */
#define LIMIT_HIGH          0x04000000  /* Limit high data mask for status bit * 5      */
#define LIMIT_LOW           0x02000000  /* Limit low data mask for status bit * 6       */
#define CRITICAL_HIGH       0x00400000  /* Critical high data mask for status * bit 9   */
#define CRITICAL_LOW        0x00200000  /* Critical low data mask for status * bit 10   */

#define DOUBLE_UP_ARROW     144         /* status character double up arrow             */
#define DOUBLE_DOWN_ARROW   146         /* status character double down arrow           */
#define UP_ARROW            218         /* status character up arrow                    */
#define DOWN_ARROW          250         /* status character down arrow                  */
#define MAX_FONTS           10          /* status character down arrow                  */
#define COLOR_OFFSET        110         /* color offset for color palette               */
```

```
#define SEC_YR_CONV          ( 1/ ( 60*60*24*365.25 ) )   /* seconds to year conversion  */
#define BASE_YEAR            70           /* system clock begins in Jan 1, 1970
#define BASE_YEAR2           0            /* system clock diff. for the next decade
#define COMP_BASE_YEAR       1970         /* system clock begins in Jan 1, 1970          */
#define YEAR_DIFF            30           /* system clock diff. for the next  decade     */

#define NEW_LOW_SCALE        1            /* calculate a new low scale                   */
#define NEW_HIGH_SCALE       2            /* calculate a new low scale                   */

#define CARTESIAN            0            /* CARTESIAN axis type                         */
#define POLAR                1            /* POLAR axis type                             */
#define LOGARITHMIC          2            /* LOGARITHMIC axis type                       */

/*
 *  Constant to check if bit is set for binary display.
 */

#define BIT_IS_SET ( bit, pointer ) \
     ( pointer[ ( bit/8 ) ] & ( unsigned char ) ( 1 << ( 7 - ( bit % 8 ) ) ) )

/*
 * Constants for graphical records
 */

#define LINE                 1            /* graphical line records                      */
#define RECTANGLE            2            /* graphical rectangle records                 */
#define POLYGON              3            /* graphical polygon records                   */
#define CIRCLE               4            /* graphical circle records                    */
#define ARC                  5            /* graphical arc records                       */
#define ELLIPSE              6            /* graphical ellipse records                   */
#define VECT_TXT             7            /* graphical vector text records
#define CURVE                8            /* graphical curve records                     */
#define ELLIPTICAL_ARC       9            /* graphical ellipse records                   */
#define PIE_CHART            11           /* pie chart records                           */
#define CLOCK_METER          12           /* clock meter records                         */
#define BAR_CHART            13           /* bar chart records                           */

#define OFF_SCALE_HIGH       0x20000000   /* Off scale high data mask for status * bit 2 */
#define OFF_SCALE_LOW        0x10000000   /* Off scale low data mask for status * bit 3  */

#define VERSION              3            /* Software release version                    */

#define HOLLOW               0            /* type of fill pattern                        */
#define SOLID                1            /* type of fill pattern                        */
#define HATCH                2            /* type of fill pattern                        */

#define DBUFFSIZE            14           /* size of the data buffer in bytes            */

#define LOCAL_TIME           "LOCAL_TIME"/* for the local time msid                      */
#define PLOT_ADVISE          0x00000001   /* WEX advisory bit 0 plot end code            */
#define PLOT_BELL            0x00000002   /* alarm bit 1 for plot end code               */

#define END_CODE_MASK        0x0000000C   /* mask off bits 0 and 1                       */

#define PI                   3.1415926536            /* mathematical constant           */
#define TWO_PI               ( 2 * 3.1415926536 )    /* mathematical constant mult. by 2 */

#define PLOT_STOP            0            /* stop the plot bits 2 and 3 off              */
#define PLOT_WRAP            4            /* wrap bit 2 and 3 for plot end code          */
#define PLOT_RESCALE         8            /* rescale bit 2 and 3 for plot end code       */
#define PLOT_BELL_STOP       2            /* stop the plot bits 2 and 3 off, bit  1 on   */
#define PLOT_BELL_WRAP       6            /* wrap bit 1,2 and 3 for plot end code        */
#define PLOT_BELL_RESCALE    10           /* rescale bit 1,2 and 3 for plot end code     */
#define PLOT_ADV_STOP        1            /* stop the plot bits 2 and 3 off              */
```

```
#define PLOT_ADV_WRAP        5          /* wrap bit 2 and 3 for plot end code          */
#define PLOT_ADV_RESCALE     9          /* rescale bit 2 and 3 for plot end code       */
#define PLOT_BELL_ADV_STOP   3          /* stop the plot bits 2 and 3 off              */
#define PLOT_BELL_ADV_WRAP   7          /* wrap bit 2 and 3 for plot end code          */
#define PLOT_BELL_ADV_RESCALE   11      /* rescale bit 2 and 3 for plot end code       */

#define MIN_FONT_SIZE        0.5        /* minimum character height for plot labels    */
#define MAX_FONT_SIZE        1.0        /* maximum character height for plot labels    */
#define LABEL_STYLE          "ital"     /* font style of plot axis labels              */

#define OFFSET_INTO_COLORMAP 110        /* offset for colors in the colormap           */

#define PBI_PRESET_NONE      0          /* PBI has no backlighting selected            */
#define PBI_PRESET_OFF       1          /* PBI has preset backlight selection off      */
#define PBI_PRESET_ON        2          /* PBI has preset backlight selection on       */
#define PBI_BKLGHT_OFF       0          /* PBI backlight condition is off              */
#define PBI_BKLGHT_ON        1          /* PBI backlight condition is on               */
#define PBI_HOST_DEST        "MOC"      /* PBI Host Destination code                   */
#define PBI_LOCAL_DEST       "DMR"      /* PBI Local Destination code                  */
#define PBI_HST_RSP_OVRD     'N'        /* Host Response Override flag for Pbi's       */
#define PBI_ACTIVE           1          /* Pbi is active with "1" -- ie on display     */
#define MULTIDEF             100        /* Multi-definition offset                     */
#define LOCAL_PBI            20         /* Offset for a local Pbi                      */
#define SP                   1          /* Give names for Pbi types: SP - Standard     */
#define FS                   2          /* Field select pbi type                       */
#define DP                   3          /* Dependent Pbi type                          */
#define DE                   4          /* Dependent Execute type                      */
#define DC                   5          /* Dependent Clear type                        */
#define DG                   6          /* Dependent Group type                        */
#define FP                   7          /* Forward for Multi def PBI type              */
#define RP                   8          /* Reverse for Multi def PBI type              */
#define ME                   9          /* MED PBI type                                */
#define DT                   10         /* DTE PBI type                                */
#define CD                   11         /* Command PBI type                            */
#define EN                   12         /* Dependent Enable pbi code                   */
#define DS                   13         /* Dependent Disable pbi code                  */

#define TOGGLE_FORWARD       1          /* Multidef pbi processing direction           */
#define TOGGLE_REVERSE       -1         /* Multidef pbi processing direction           */

#define NO_BCKLGHT           0          /* Backlighting disabled code                  */
#define INIT_BCKLGHT         2          /* Backlighting enabled and initially on       */

#define MFS                  102        /* multidef Field Selection Option             */
#define MFP                  107        /* multidef Forward Button                     */
#define MRP                  108        /* multidef Reverse Button                     */

#define DISABLED             1          /* flag for disabling PBIs                     */
#define ENABLED              0          /* flag for enabling PBIs                      */
#define PBI                  0          /* signifies PBIs being parsed                 */
#define PFKEY                1          /* signifies PF keys being parsed              */
#define COMMAND_LINE         100        /* lenghth of a command line                   */
#define MAX_SCREEN_LEN       100        /* maximum screen length in world coord.       */
#define MIN_SCREEN_LEN 0                /* minimum screen length in world coord.       */

#define BORDER               20         /* border for help menus with elevator         */
#define SLIDER_BAR           16         /* slider bar interactive input type           */
#define TOGGLE_SWITCH        17         /* toggle switch interactive input             */

#define HAND_ANG             ( PI/48 )  /* angle for poly. hands                       */
#define LG_LINE_HAND         0          /* large clock/meter line hand type            */
#define MED_LINE_HAND        1          /* medium clock/meter line hand type           */
#define SM_LINE_HAND         2          /* small clock/meter line hand type            */
#define LG_POLY_HAND         3          /* large clock/meter poly hand type            */
```

```
#define MED_POLY_HAND       4           /* medium clock/meter poly hand type      */
#define SM_POLY_HAND        5           /* small clock/meter poly hand type
```

```
/*******************************************************************************
 * MODULE NAME: disp.h
 *
 *  This function defines structures required for a display.
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Ronnie Killough - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *******************************************************************************/


#include <X11/Xlib.h>
#include <X11/Intrinsic.h>

/*
 *  This structure contains information for each displayer task that is acti-
 *  vated.  This information is used by all Display Manager tasks within a
 *  workstation.
 */

struct ppl_record {
    char            ppl_filename[51];       /* PPL Filename                      */
    char            host_name[4];           /* host name ( "MOC" or "DSC" )      */
    char            retrieval_qualifier[11];/* PPL Retrieval Qualifier           */
    char            rate_units[2];          /* unit of time ( 'D', 'H', 'M', 'S' */
                                            /* for days, hours, minutes, secs )  */
    char            duration_units[2];      /* unit of time ( 'D', 'H', 'M', 'S' */
                                            /*  for days, hours, minutes, secs ) */
    char            ppl_rate[5];            /* PPL rate ( 0 - 255 )              */
    char            duration[5];            /* PPL duration ( 0 - 255 )          */
    long            total_bytes;            /* total bytes in data value buffer  */
    short           get_next;               /* YES, if get command for GDR       */
    short           update_retr;            /* YES, if retrieval qual. updated   */
    short           src_indx;               /* source index into source info     */
    short           strm_id;                /* stream Id of the source           */
};

struct limit_chg {
    char            msid_name[11];          /* MSID name                         */
    short           limt_flag;              /* limit sense flag                  */
    double          low_limit;              /* ops low limit                     */
    short           ol_alrm;                /* ops low alarm flag                */
    short           ol_adv;                 /* ops low advisory flag             */
    double          hi_limit;               /* ops high limit                    */
    short           oh_alrm;                /* ops high alarm flag               */
    short           oh_adv;                 /* ops high advisory flag            */
    short           crit_flag;              /* crit limit sense flag             */
    double          crit_low;               /* crit low value                    */
    short           cl_alrm;                /* critical low alarm flag           */
    short           cl_adv;                 /* critical low advisory flag        */
    double          crit_high;              /* crit high value                   */
    short           ch_alrm;                /* critical high alarm flag          */
    short           ch_adv;                 /* critical high advisory flag       */
    char            src[4];                 /* MSID source                       */
    short           updated;                /* YES when limits retrieved or      */
                                            /* updated                           */
    char            option[4];              /* option of EVN or PPM data source  */
    short           indx;                   /* index of limit entry              */
    short           alarm;                  /* set to yes if alarm is enabled    */
};
```

```
struct disp_info {
    short           disp_active;    /* flag set by display manager when a display   */
                                    /* manager is started                           */
    long            update_rate;    /* update rate for the display in milliseconds  */
    short           disp_init;      /* flag set by a displayer when initialized     */
    short           active_display; /* flag set by Display Mgr when display is up   */
    short           clear;          /* flag set by display manager to clear a disp  */
    short           dh_clear;       /* flag set by display manager to clear a disp  */
                                    /* used by DDH                                  */
    short           halt;           /* flag set by display manager to halt a disp   */
    short           disp_pause;     /* flag set by display manager to "pause" a disp*/
    short           new_display;    /* flag set by display manager for a change in  */
                                    /* displays                                     */
    short           disp_halt_ack;  /* flag set by displayer when ready to exit     */
    short           get_lim;        /* flag set by display manager to get limit     */
                                    /* values                                       */
    short           upd_lim;        /* flag set by display manager to notify the DDH*/
                                    /* to update the limits in displays and its list*/
    short           grp_lim;        /* flag set by display manager to notify the DDH*/
                                    /* to turn on or off limit groups               */
    short           dh_grp_limit;   /* flag set by Data Handler to acknowledge if   */
                                    /* a good limit group cmd executed              */
    short           dh_plot;        /* command to start or stop a plot for the DH   */
    short           dh_plot_ack;    /* flag set by Data Handler to acknowledge if   */
                                    /* a good plot command executed.                */
    short           read_plot;      /* command to read a plot def. file for the DD  */
    short           dd_strt;
    short           dd_stop;
    short           dd_ovrl;        /* command to overlay a plot for the Displayer  */
    short           action;         /* type of action to take, i.e. start, stop...  */
    struct limit_chg limits;        /* limits to change                             */
    short           dh_new_disp;    /* flag set by display manager for a change in  */
                                    /* displays - used by DDH                       */
    short           dh_disp_init;   /* flag set by Data Handler to acknowledge if   */
                                    /* good display initialization                  */
    int             disp_pid;       /* process Id of the Displayer task             */
    char            display_name[DNAME_LEN];    /* name of display                  */
    char            plot_name[DNAME_LEN];       /* name of plot                     */
    char            plot_overlay[DNAME_LEN];    /* name of plot overlay             */
    char            plot_path[DNAME_LEN];       /* default plot path                */
    char            disp_path[DNAME_LEN];       /* default display path             */
    char            flight_id[5];               /* flight associated with display   */
    char            strm_type[3];   /* stream   type associated with this display   */
    short           strm_no;        /* stream number associated with this display   */
    char            pos_id[14];     /* position id of the user                      */
    short           log_enable;     /* used by the Displayer to log display data set*/
                                    /* by the Display Manager                       */
    short           dd_zoom;        /* zoom flag for displayer                      */
    short           pos_id_indx;    /* index to the matching pos. Id for w/s alarm  */
    short           dh_htab;        /* history tab command for handler              */
    short           dd_htab;        /* set by DH for displayer to update hist tabs  */
    char            htab_file[DNAME_LEN];       /* name of the history tab file to  */
                                    /* update                                       */
    short           unlatch;        /* ddd unlatch flg  1- all; 2 - particular msid */
    char            msid_name[MSID_LENGTH + 1]; /* MSID to unlatch                  */
    char            src[4];         /* source of msid to unlatch                    */
    float           dd_zfact;       /* zoom factor for displayer                    */
    float           dd_xpt;         /* zoom in to x coordinate                      */
    float           dd_ypt;         /* zoom in to y coordinate                      */
    float           low_x;          /* world coord. for displayer zoom              */
    float           low_y;          /* world coord. for displayer zoom              */
    float           high_x;         /* world coord. for displayer zoom              */
    float           high_y;         /* world coord. for displayer zoom              */
```

```
    int             size_x;         /* Size in pixels of the display                */
    int             size_y;         /* Size in pixels of the display                */
    float           factor_x;       /* Multiply value for WC -> pixels conversion   */
    float           factor_y;       /* Multiply value for WC -> pixels conversion   */
};

struct stream_info {
    char            flt_id[5];      /* flight Id for this stream                    */
    char            strm_type[3];   /* stream type for this stream                  */
    short               nbr_conn;       /* number of displays using this flight         */
};

struct flags {
    short           dh_not_halted;  /* Data Handler halt flag - set to FALSE when   */
                                    /* task is to be halted                         */
    short           disp_halt_nbr;  /* display nbr which is being halted when the   */
                                    /* Data Handler is being halted                 */
    short           dh_initialized; /* set by Data Handler when the task is         */
                                    /* successfully initialized.                    */
    short           dh_num;         /* Data Handler display number for event file   */
    short           dh_evnt;        /* read the event trigger files                 */
    short           dh_ack_evnt;    /* processed event and default limit files      */
    short           disp_init;      /* set by the Data Handler after new display is */
                                    /* is initialized                               */
    short           dh_halt_ack;    /* Data Handler halt acknowledge flag           */
    short           disp_nbr;       /* number of active displays in a workstation   */
    int             dh_pid;         /* process id of Data Handler                   */
    short           gdr_get_next;   /* YES, if GDR get next command                 */
    short           upd_retrieval;  /* YES, for retrieval qualifiers update         */
    short           nbr_streams;    /* number of active streams                     */
    int             data_shm_id;    /* shared memory Id for the Data segment        */
    short           wex_mode;       /* set to OPS if in operational mode            */
    short           alarm[MAX_POS_ID];  /* set to ON if audible alarm is enabled for*/
                                        /* w/s within a position Id                 */
    char            pos_id[MAX_POS_ID][14]; /* pos. id for each w/s alarm flag      */
    short           nbr_pos[MAX_POS_ID];    /* number display managers to pos. Id   */
    short           log_enable;     /* Enable/Disable all DM logging flag           */
};

struct plot_info {
    char            act_plots[MAX_PLOTS][DNAME_LEN]; /* list of all the active plots*/
    char            stop_plot[MAX_PLOTS][DNAME_LEN]; /* list of all plots to stop   */
    char            unv_plot[DNAME_LEN];    /* universal plot file to update        */
    long            plot_cycles[MAX_PLOTS]; /* number of plot data cycles in plot   */
};

/*
 *  PBI Display Definition Structures for Shared Memory
 */

struct pbi_changes {
    short           pbi_chg_ndx;        /* index of the pbi to be changed           */
    short           pbi_active_flag;    /* new value of the active flag for the pbi */
    short           pbi_feedback_flag;  /* new value of the fdbk indicator for pbi  */
};

struct pbi_shm {
    short       disp_num;
    short           number_of_changes;
    struct pbi_changes pbi_change[128];
};

struct pbi_redraw_rect {
    double          ulx;                /* upper left x of the rect to be drawn     */
```

```
    double          uly;                /* upper left y of the rect to be drawn     */
    double          lrx;                /* lower right x of the rect to be drawn    */
    double          lry;                /* lower right y of the rect to be drawn    */
};

struct dm_shmemory {
    struct flags        process;            /* structure containing halt/count flags*/
    struct disp_info    display[MAX_DISP];  /* information used in display proc     */
    short               dm_pid[MAX_DISP];   /* pid's for all the active manager's   */
    struct stream_info  strm[MAX_FLTS];     /* stream information                   */
    struct plot_info    plots;              /* plot information                     */
    Window              window[MAX_DISP];   /* window information                   */
    Display             *xdisplay[MAX_DISP];/* X display information                */
    Widget              shell[MAX_DISP];    /* X display widget */
    char                display_name[MAX_DISP][10]; /* X display name               */
    GC                  gc[MAX_DISP];
    XGCValues           gc_val[MAX_DISP];
    struct ppl_record   ppl_recs[NUM_GDR];  /* ppl record information               */
    struct pbi_shm      pbi_shmemory;       /* changes to PBIs                      */
    struct pbi_redraw_rect pbi_redraw;      /* area to be redrawn for pbi updates   */
};

struct file_info {
    char            name[15];               /* display file name                    */
    char            desc[33];               /* display description                  */
    short           inverse_flag;           /* display select inverse video flag    */
    char            act_flag[9];            /* plot active/inactive flag            */
};

/*
 *  Structure for universal plot msid definitions.
 */

struct msid_record {
    char            msid_name[MSID_LENGTH+1];   /* actual msid name                 */
    char            source[4];                  /* msid data source                 */
    char            sample[4];                  /* sample nbr for this msid         */
    char            axis[2];                    /* x or y axis                      */
    short           axis_nbr;                   /* axis nbr to plot this msid on    */
    char            plot_msid[MSID_LENGTH + 1]; /* msid to plot against             */
    short           plot_axis;                  /* axis nbr of plot msid            */
    char            plot_type[2];               /* axis type of plot msid           */
    char            plot_source[4];             /* source of the plot msid          */
    char            plot_sample[4];             /* plot msid sample nbr             */
    short           plot_indx;                  /* plot msid index                  */
};

/*
 *  PBI Display Definition Structure for Display Manager PBI Internal Processes
 */

struct pbi_def {
    short           pbi_disable;        /* Pbi disabled if on/enabled if off w/ global  */
    double          pbi_ul_x;           /* Upper Left x coordinate for PBI hot box      */
    double          pbi_ul_y;           /* Upper Left y coordinate for PBI hot box      */
    double          pbi_lr_x;           /* Lower Left x coordinate for PBI hot box      */
    double          pbi_lr_y;           /* Lower Left y coordinate for PBI hot box      */
    int             pbi_ul_x_p;         /* Upper Left x coordinate for PBI hot box      */
    int             pbi_ul_y_p;         /* Upper Left y coordinate for PBI hot box      *
    int             pbi_lr_x_p;         /* Lower Left x coordinate for PBI hot box      *
    int             pbi_lr_y_p;         /* Lower Left y coordinate for PBI hot box      */
    short           pbi_bklght;         /* Pbi backlight feedback preset opt: 0, 1, 3   */
    short           pbi_mesg_len;       /* Message length for local command PBIs 0=none */
    char            *pbi_message;       /* Message for local command PBIs               */
```

```
    short           pbi_dest_len;       /* Destination length for PBI record          */
    char            *pbi_dest;          /* Destination of the PBI record entry        */
    short           pbi_dep_msid_cnt;   /* Number of Dependent group MSID's           */
    char            *pbi_dep_msids;     /* Pointer to arrays of Dependent Group MSID's */
    short           pbi_cmd_cnt;        /* Number of command structures               */
    struct pfkey_defs *pbi_cmd_ptr;     /* Pointer to array of command structures     */
};

struct pbi_msid_rec {
    char            pbi_msid[MSID_LENGTH];  /* Entry for PBI MSIDs for Dependent Groups */
};

struct limit_file {
    struct limit_file *next_ptr;
    struct limit_file *prev_ptr;
    char            file_name[DNAME_LEN];
};
```

```
/*****************************************************************************
 * MODULE NAME: ds_stub.h
 *
 *  This is the header file for the stubbed versions of ds_connect,
 *  ds_getkeys, ds_getparms.
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Ronnie Killough - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/

/*
 * Constants
 */

#define FAKE_KEY        7

#define EVN             1
#define MTM             2
#define GDR             3
#define NDM             4
#define PTM             5
#define PPM             6
#define USR             7

#define INVALID         -1
/* #define md_fn            "mdef" */
#define md_fn           "new_mdef"
#define md_fn_new       "new_mdef"
#define NUM_SRCS        7
#define MAX_DEF         1050

#define MAX_FLOAT       40000.0
#define HALF_MAX_FLOAT  20000.0
#define MAX_LONG        64000
#define HALF_MAX_LONG   32767
#define MAX_DBL         40000.0
#define HALF_MAX_DBL    20000.0

#define DATA_DIR        "/WEX/Datafiles/display/SWRITEST"


/*
 *  Structures.
 */

struct mdef_node {
    char msid[20];
    long length;
    int attribute;
    int low_scale;
    int high_scale;
    struct mdef_node *next;
};
```

```
/*****************************************************************************
 * MODULE NAME: pf_key.h
 *
 *  This header defines the structure used to keep track of a command (and
 *  function keys).
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#define PFKEY_COUNT 28

/*
 *  This structure contains information used to cause limit changes.
 */

struct new_limits {
    double      ops_ll;             /* new operational lower limit          */
    int         ol_alrm;            /* ops low alarm flag                   */
    int         ol_adv;             /* ops low advisory flag                */
    double      ops_ul;             /* new operational upper limit          */
    int         oh_alrm;            /* ops high alarm flag                  */
    int         oh_adv;             /* ops high advisory flag               */
    double      crit_ll;            /* new critical lower limit             */
    int         cl_alrm;            /* crit low alarm flag                  */
    int         cl_adv;             /* crit low advisory flag               */
    double      crit_ul;            /* new critical upper limit             */
    int         ch_alrm;            /* crit high alarm flag                 */
    int         ch_adv;             /* crit high advisory flag              */
    char        msid[11];           /* MSID on which to change limits       */
    char        src[4];             /* MSID source where limits will change */
    char        option[4];          /* option for the EVN and PPM data sources */
    int         flag;               /* 0 = change operational limits        */
                                    /* 1 = change critical limits           */
                                    /* 2 = change both                      */
};

/*
 *  This structure contains information for each PF key binding.
 */

struct pfkey_defs {
    int         key_no;             /* PF key number                        */
    int         prompt_flag;        /* prompt flag for pf keys              */
    int         valid_flag;         /* Set to 0 for valid, -1 for invalid   */
    int         defined;            /* set to 0 if not defined              */
    int         func_no;            /* number pertaining to the function    */
    char        disp_name[DNAME_LEN]; /* display, or limit group, or plot file */
    char        ovr_name[DNAME_LEN]; /* plot overlay file name              */
    struct new_limits limit_change; /* pointer to list of new limit values  */
    long        upd_rate;           /* display update rate in milliseconds  */
    long        rate;               /* display update rate in seconds       */
    int         action;             /* set to one if start limit group or plot */
    float       factor;             /* zoom factor                          */
    char        *mesg_ptr;          /* message for pbi external commands    */
};
```

```
/*****************************************************************************
 * MODULE NAME: user_inter.h
 *
 *  Define the types of the user interface library functions used to create
 *  the various * types of widgets.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


extern Widget   tui_create_app_shell     ( ),
                tui_create_pushbutton    ( ),
                tui_create_cascade       ( ),
                tui_create_fileselector  ( ),
                tui_create_form          ( ),
                tui_create_label         ( ),
                tui_create_rb            ( ),
                tui_create_sel           ( ),
                tui_create_separator     ( ),
                tui_create_scale         ( ),
                tui_create_text          ( ),
                tui_create_toggle        ( ),
                tui_create_trans_shell   ( ),
                tui_display_message      ( );

char            *tui_radio_get_value     ( );
```

```
/*****************************************************************************
 * MODULE NAME: DDpbi_updt
 *
 *  This function process requests to reconfigure or change backlighting
 *  of a pbi button.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Scott Zrubek    - Ford Aerospace Corporation/Houston
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Ronnie Killough - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <wex/EXmsg.h>
#include <constants.h>
#include <disp.h>
#include <DDdisp.h>


extern struct dm_shmemory   *Dm_Address;     /* Ptr to DM shared memory.                */


int DDpbi_updt(disp_num)

    short   disp_num;                        /* Effective display number.               */
{
    struct pbi_changes      *pbi_chg_ptr;    /* Pointer to change flags.                */
    struct pbi_redraw_rect  redraw_rect;     /* Rectangle for redraw to redraw.         */

    int                     i,               /* Loop counter.                           */
                            num_of_changes;  /* Number of pbis changed.                 */

    char                    *calloc();       /* Get malloc as a pointer.                */

    D(printf("START DDpbi_updt\n"));
/*
 *  Copy all of the necessary pbi shared memory information
 */

    num_of_changes = Dm_Address->pbi_shmemory.number_of_changes;

    pbi_chg_ptr = (struct pbi_changes *)
                        calloc((unsigned)num_of_changes, sizeof(struct pbi_changes));

    for (i = 0; i < num_of_changes; i++) {

        pbi_chg_ptr[i].pbi_chg_ndx =
                    Dm_Address->pbi_shmemory.pbi_change[i].pbi_chg_ndx;
        pbi_chg_ptr[i].pbi_active_flag =
                    Dm_Address->pbi_shmemory.pbi_change[i].pbi_active_flag;
        pbi_chg_ptr[i].pbi_feedback_flag =
                    Dm_Address->pbi_shmemory.pbi_change[i].pbi_feedback_flag;
    }

/*
```

```
     *  Reset shared memory flags
     */

        redraw_rect = Dm_Address->pbi_redraw;

        Dm_Address->pbi_shmemory.number_of_changes = 0;
        Dm_Address->pbi_shmemory.disp_num = -1;

        pbi_config(disp_num, redraw_rect, pbi_chg_ptr, num_of_changes);

    /*
     *  Normal return.
     */

        D(printf("END DDpbi_updt\n"));
        return ( 0 );
    }
```

```
/*****************************************************************************
 * MODULE NAME: add_pt.c
 *
 *  This function returns the value of the function defined for
 *  the limit or nominal line for the input value.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Tod Milam       - Ford Aerospace Corporation/Houston
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Ronnie Killough - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <math.h>   /* to be able to use pow */


float add_pt(xnum, coeffs, poly_num)

    float           xnum;
    double          coeffs[6];
    int             poly_num;
{
    float   ynum = 0;               /* the summations variable              */

    int     count;                  /* a general purpose loop variable      */

/*
 *  Loop to sum all of the coefficient values
 */

    for (count = 0; count < poly_num; count++) {

        if (xnum == 0.0 && (poly_num - count - 1.0) == 0)
            ynum += coeffs[count];

        else if (xnum != 0.0)
            ynum += coeffs[count]
                        * pow((float) xnum, (float) (poly_num - count - 1.0));
    }

/*
 *  Return the summed value.
 */

    return (ynum);
}
```

```
/*****************************************************************************
 * MODULE NAME: cb_cmd.c
 *
 *  This callback function handles menu and function key generated commands. It
 *  calls the (command) function to actually handle the command.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <memory.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Shell.h>
#include <constants.h>
#include <disp.h>
#include <pf_key.h>
#include <wex/EXmsg.h>


extern struct pfkey_defs    Act_Pfkeys[],   /* Contains the set of act. function keys.  */
                            Current_Com;    /* Contains the current command.            */


/* ARGSUSED */
XtCallbackProc cb_cmd ( w, closure, calldata )

    Widget  w;                  /* Set to widget which in which callback originated.    */

    caddr_t closure,            /* Callback specific data. Set to desired command.      */
            *calldata;          /* Widget-specific information.                         */
{
    D(printf("START cb_cmd\n"));
/*
 *  If the closure value is greater than zero, then the command was generated
 *  via a menu. In this case, save the command, set the prompt flag to NO and
 *  call (command) to actually process the command.
 */

    if ( (int)closure > 0 ) {
        Current_Com.func_no     = (int)closure;
        Current_Com.prompt_flag = NO;
        command ( YES );

/*
 *  Otherwise the command was generated from a function key. If the command
 *  does not need to be verified or if it does and the user verified it, copy
 *  the command information into the current command structure and then call
 *  (command) to actually execute the command.
 */

    } else {
        if ( Act_Pfkeys[-(int)closure].prompt_flag == 0 ||
             pf_chk ( &Act_Pfkeys[-(int)closure] ) ) {
            memcpy ( (char *)&Current_Com, (char *)&Act_Pfkeys[-(int)closure],
                     sizeof(struct pfkey_defs) );
            command ( NO );
```

```
            }
        }

    /*
     *  Normal return.
     */

        D(printf("END cb_cmd\n"));
        return;
    }
```

```
/*****************************************************************************
 * MODULE NAME: cb_expose_plot
 *
 *  This callback function is responsible for keeping track of all
 *  expose events generated for plots.  When the expose event count
 *  reaches zero, all plot axes, grid lines, labels, and tick marks,
 *  and plot data lines are redrawn.
 *
 *  Expose events occur upon window creation, when an obscuring window is
 *  removed or moved to the back of the window stack, and when the user
 *  uses the scrolled window to pan through the image.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Ronnie Killough - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <fcntl.h>
#include <unistd.h>
#include <Xm/Xm.h>
#include <Xm/DrawingA.h>
#include <constants.h>
#include <disp.h>
#include <DDplot.h>
#include <wex/wex.h>


extern struct dm_shmemory   *Dm_Address;     /* ptr to DM shared memory       */
extern struct plot_ptrs *Plot_info_ptr;      /* ptr to list of plots          */
extern short    Nbr_of_plots;                /* number of plots in plot list  */
extern short    End_of_file;                 /* plot data file EOF flag      */


/* ARGSUSED */
XtCallbackProc cb_expose_plot( widget, closure, calldata)

    Widget      widget;         /* widget in which callback originated       */

    caddr_t     closure;        /* callback specific data (display number    */

    XmDrawingAreaCallbackStruct
                *calldata;      /* widget-specific data                      */

{
    XExposeEvent    *expose;                /* expose event structure        */

    struct plot_ptrs *plot_ptr;             /* ptr thru plot list            */

    int i;                                  /* loop counter                  */

    short   disp_num;                       /* effective display number      */


    D(printf("START cb_expose_plot\n"));

/*
 *  If another expose event is pending, exit from function.
```

```
 */

    expose = &calldata->event->xexpose;

    if ( expose->count != 0 )
        return;

/*
 *  Extract display number from the arg list.
 */

    disp_num = (short)closure;

/*
 *  Locate the plot associated with this expose event
 */

    i = 0;
    plot_ptr = Plot_info_ptr;
    while (widget != plot_ptr->draw_win) {
        plot_ptr = Plot_info_ptr + i;
        i++;
    }

/*
 *  If an associated plot is found, redraw the plot axes,
 *  grid lines, labels, and tick marks.  If the plot has
 *  been active, rewind the plot data file and redraw the
 *  plot data lines.
 */

    if (widget == plot_ptr->draw_win) {

        draw_plt(disp_num, plot_ptr, expose->x, expose->y,
                                          expose->width, expose->height);

        if (plot_ptr->prev_act_flg == YES) {

/*
 *          Reset all first point flags.
 */

            for (i=0; i<plot_ptr->header->actual_msids; i++)
                (plot_ptr->msids + i)->first_pt = YES;

/*
 *          Rewind plot data file to
 *          beginning of plot data.
 */

            lseek(plot_ptr->plot_fp, 0, SEEK_SET);
            lseek(plot_ptr->plot_fp,
                            80 + (plot_ptr->header->msid_num * 24), SEEK_SET);

/*
 *          Reset seconds elapsed to 0 (only meaningful
 *          for time plots).
 */

            plot_ptr->seconds_elapsed = 0;

/*
 *          Initialize end_of_file flag and call proc_plt() to plot
 *          data points.  Continue calls to proc_plt() until EOF, signalling
```

```
 *          all data points plotted.  Check for EOF necessary since
 *          proc_plt() and subordinate plot_msid() may exit prematurely due
 *          to an out-of-scale data point.
 */

            End_of_file = NO;

            while (End_of_file == NO)
                proc_plt(disp_num, plot_ptr);
        }

    }

    D(printf("END cb_expose_plot\n"));
    return;
}
```

```
/*****************************************************************************
 * MODULE NAME: cb_expose_display
 *
 *  This callback function is responsible for keeping track of all current
 *  expose events and the exposed rectangle coordinates for the base display
 *  window (expose events on plots are handled by cb_expose_plot).  When the
 *  expose event count reaches zero, this function calls redraw for each
 *  exposed rectangle.
 *
 *  Expose events occur upon window creation, when an obscuring window is
 *  removed or moved to the back of the window stack, and when the user
 *  uses the scrolled window to pan through the image. This function
 *  collects all exposed rectangles and only refreshes the required
 *  portions of the image.
 *
 * DEVELOPMENT NOTES:
 *
 *  o   This routine keeps track of all exposed rectangle coordinates.
 *      However, in its current implementation, it will keep track of expose
 *      area coordinates for a single display only.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Ronnie Killough - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include <Xm/DrawingA.h>
#include <constants.h>
#include <disp.h>
#include <wex/wex.h>


extern struct dm_shmemory   *Dm_Address;    /* ptr to DM shared memory       */


/* ARGSUSED */
XtCallbackProc cb_expose_display( widget, closure, calldata)

    Widget      widget;         /* widget in which callback originated       */

    caddr_t     closure;        /* callback specific data (display number)   */

    XmDrawingAreaCallbackStruct
                *calldata;      /* widget-specific data                      */

{
    static int  index = 0;          /* number of outstanding expose areas    */

    static XRectangle   rects[50];  /* coord. of outstanding expose areas    */

    XExposeEvent    *expose;        /* ptr to current expose event structure */

    short               disp_num;   /* disp # extracted from cb arg list     */

    D(printf("START cb_expose_display\n"));
```

```
    /*
     *  Extract expose event structure address from the callback data
     *  and save the expose area coordinates in the save array.
     */

        expose = &calldata->event->xexpose;

        if (expose->width == 0 || expose->height == 0)
            return;

        rects[index].x      = expose->x;
        rects[index].y      = expose->y;
        rects[index].width  = expose->width;
        rects[index].height = expose->height;
        index++;

    /*
     *  If another expose event is pending, exit from function
     *  to prevent extraneous screen refreshes.
     */

        if ( expose->count != 0 )
            return;

    /*
     *  Extract display number from the arg list.
     */

        disp_num = (short)closure;

    /*
     *  Clear each exposed rectangle and redraw its contents
     */

        for (; index; index--) {

            XClearArea(Dm_Address->xdisplay[disp_num], XtWindow(widget),
                       rects[index-1].x, rects[index-1].y,
                       rects[index-1].width, rects[index-1].height, False);

            redraw(disp_num, rects[index-1].x, rects[index-1].y,
                       rects[index-1].x + rects[index-1].width - 1,
                       rects[index-1].y + rects[index-1].height - 1);
        }

        D(printf("END cb_expose_display\n"));
        return;
    }
```

```
/*****************************************************************************
 * MODULE NAME: cb_help.c
 *
 *  This function displays a help popup. The popup is non-transient and may
 *  remain displayed as long as needed by the user.
 *
 *
 * INTERNAL FUNCTIONS:
 *
 *  o   cb_remove   -   Callback used to delete the help popup.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Shell.h>
#include <Xm/Xm.h>
#include <Xm/Form.h>
#include <user_inter.h>
#include <constants.h>
#include <wex/EXmsg.h>


extern Widget   Top;                    /* The top level widget used as the help parent */


static char *help_files[HELP_MAX] = {
    "enable_msg",
    "flight_data",
    "screen_dump",
    "edit_colors",
    "exit",
    "select_disp",
    "remove_disp",
    "freeze_disp",
    "update_rate",
    "unlatch_ddd",
    "unlatch_all",
    "change_gdr",
    "enable_alarms",
    "enable_pbis",
    "enable_log",
    "enable_all_log",
    "list_limits",
    "change_limits",
    "list_plots",
    "disp_ovl",
    "save_ovl",
    "def_univ",
    "zoom",
    "reset_zoom",
    "set_zoom",
    "keys" };
```

```
/* ARGSUSED */
XtCallbackProc cb_help ( w, closure, calldata )

    Widget          w;          /* Set to widget which in which callback originated.    */

    caddr_t         closure,    /* Callback specific data. Set to desired command.      */
                    *calldata;  /* Widget-specific information.                         */
{
    register int    i,
                    size = 0;

    Widget          shell,
                    form;

    Arg             args[10];

    XtCallbackProc  cb_remove();

    FILE            *fp;

    char            *buffer,
                    string[50],
                    *malloc();

    D(printf("START cb_help\n"));
/*
 *  Output an error if an invalid help index was passed.
 */

    if ( (int)closure < 0 || (int)closure > HELP_MAX-1 ) {
        tui_msg ( M_YELLOW, "Help called with invalid help text index" );
        return;
    }

/*
 *  Allocate buffer. Output an error if this fails.
 */

    if ( ( buffer = malloc ( HELP_BUF_SIZE ) ) == NULL ) {
        tui_msg ( M_YELLOW, "Could not allocate buffer for file" );
        return;
    }

/*
 *  Attempt to open the help file. Output an error if this fails.
 */

    strcpy ( string, HELP_DIR );
    strcat ( string, help_files[(int)closure] );

    if ( ( fp = fopen ( string, "r" ) ) == NULL ) {
        tui_msg ( M_YELLOW, "Could not open requested help file" );
        return;
    }

/*
 *  Read help file into buffer. Terminate the buffer with a NULL.
 */

    while ( size < HELP_BUF_SIZE && ( *(buffer+size) = fgetc ( fp ) ) != -1 )
        size++;

    *(buffer+size) = NULL;
```

```
    /*
     *  Close the file.
     */

        fclose ( fp );

    /*
     *  Create the shell widget, the main form, the text area, a separator, and
     *  a push button widgets to allow the popup to be removed. Note that the
     *  push button causes a callback to the (cb_remove) function which in turn
     *  removes the popup.
     */

        i = 0;
        shell = tui_create_trans_shell ( "Display Manager Help", args, i );

        i = 0;
        form = tui_create_form ( shell, "form", TRUE, args, i );

        i = 0;
        tui_create_text ( form, "text", buffer, 0, XmMULTI_LINE_EDIT, FALSE, args, i );

        i = 0;
        XtManageChild ( XmCreateSeparator ( form, "sep", args, i ) );

        i = 0;
        tui_create_pushbutton ( form, "Done", cb_remove, (caddr_t)shell, args, i );

    /*
     *  Realize all widgets, popup the shell, and set the color map.
     */

        XtRealizeWidget ( shell );
        XtPopup ( shell, None );
        set_cmap ( shell );

    /*
     *  Free the text buffer.
     */

        free ( buffer );

    /*
     *  Normal return.
     */

        D(printf("END cb_help\n"));
        return;
    }
```

```
/*****************************************************************************
 * MODULE NAME: cb_remove
 *
 *  This function removes the help popup. It is passed the shell widget which
 *  is the parent of the widget hierarchy to remove.
 *****************************************************************************/

/* ARGSUSED */
static XtCallbackProc cb_remove ( w, closure, calldata )

    Widget  w;                  /* Set to the widget which initiated this
                                 * callback function.
                                 */

    caddr_t closure,            /* Callback specific data.  This parameter
                                 * is set to the shell of the help popup.
                                 */
            *calldata;          /* Specifies any callback-specific data the
                                 * widget needs to pass to the client.
                                 */
{
    D(printf("START cb_remove\n"));
/*
 *  Destroy the shell which is the parent of the help popup.
 */

    XtDestroyWidget ( (Widget)closure );

/*
 *  Normal return.
 */

    D(printf("END cb_remove\n"));
    return;
}
```

```
/*****************************************************************************
 * MODULE NAME: cb_pbi.c
 *
 *  This callback function is called when the user selects the left mouse but-
 *  ton in the display window. This event corresponds to selection of a PBI.
 *  If a PBI is defined at the coordinates of the cursor where the button was
 *  selected, the PBI will be executed (if enabled).
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Shell.h>
#include <Xm/Xm.h>
#include <Xm/DrawingA.h>
#include <constants.h>
#include <disp.h>
#include <pf_key.h>
#include <wex/EXmsg.h>


extern struct pfkey_defs    Act_Pfkeys[];   /* Array of active function keys.          */

extern short                Pbi_Hot_Ndx,    /* Pointer to the current PBI.             */
                            Pbi_Num,        /* Number of PBI's.                        */
                            Pbi_Disable;    /* Flag indicating if PBI's are active.    */


/* ARGSUSED */
XtCallbackProc cb_pbi ( w, closure, calldata )

    Widget          w;          /* Set to widget which in which callback originated.   */

    caddr_t         closure;    /* Callback specific data.                             */

    XmDrawingAreaCallbackStruct
                    *calldata;  /* Widget-specific information.                        */
{
    XButtonEvent    *button;

    D(printf("START cb_pbi\n"));
/*
 *  If the event was a button press, return. The only events processed are
 *  button releases.
 */

    button = &calldata->event->xbutton;
    if ( button->state == 0 )
        return;

/*
 *  If PBI's are disabled or if there are no PBI's active in the display,
 *  output the appropriate warning.
 */
```

```
    if ( Pbi_Disable ) {
        tui_msg ( M_YELLOW, "Input to PBI's Disabled" );
        return;
    } else if ( Pbi_Num == 0 ) {
        tui_msg ( M_YELLOW, "No PBI's are active" );
        return;
    }

/*
 *  PBI's enabled and present, so if the coordinates of the events are with
 *  in the bounding box of any PBI, call (pbi_cmd) to execute the command.
 */

    if ( Pbi_Hot_Ndx = pbi_hot ( button->x, button->y ) )
        pbi_cmd ( );

/*
 *  Normal return.
 */

    D(printf("END cb_pbi\n"));
    return;
}
```

```
/*****************************************************************************
 * MODULE NAME: cb_zoom.c
 *
 *  This function is added as a callback routine to all active plots by zoom()
 *  on the effective display whenever the Zoom Display or Zoom Reset command is
 *  called.  This function removes the callbacks added by zoom(), restores the
 *  cursor to its normal mode, does all the calculations necessary to
 *  effect the zoom or zoom reset and calls the function to redraw the plot.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  K. Noonan       - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Ronnie Killough - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <X11/Intrinsic.h>
#include <X11/cursorfont.h>
#include <Xm/Xm.h>
#include <constants.h>
#include <disp.h>
#include <DDplot.h>
#include <wex/EXmsg.h>


extern Widget   Top;                          /* top-level widget             */

extern struct dm_shmemory   *Dm_Address;      /* ptr to DM shared memory      */
extern struct plot_ptrs     *Plot_info_ptr;   /* ptr thru plot records        */

extern float    Zoom_factor;                  /* current zoom factor          */

extern short    Nbr_of_plots;                 /* # of plots in display        */
extern short    End_of_file;                  /* EOF flag for plot data file  */


XtCallbackProc  cb_zoom (widget, closure, calldata)

    Widget      widget;             /* widget in which callback originated  */

    caddr_t     closure;            /* callback-specific data (disp_num)    */

    XmDrawingAreaCallbackStruct
                *calldata;          /* widget-specific data                 */

{
    XButtonEvent    *button;                  /* container for button event   */
    XPoint          focus;                    /* zoom focus point             */

    struct plot_ptrs    *plot_ptr;            /* ptr thru plot records        */
    struct plot_tmplt   *tmplt_ptr;           /* ptr to plot positional info  */
    struct axis_info    *axis_ptr;            /* ptr thru x/y axes records    */
```

```
    double          wc_base;                /* zoomed wrld ccord base size  */

    int             i;                      /* loop control                 */

    short           disp_num,               /* effective display number     */
                    command;                /* zoom (17) or zoom reset (18) */


    D(printf("START cb_zoom\n"));

/*
 *  If the event was a button press, return. The only events processed are
 *  button releases.
 */

    button = &calldata->event->xbutton;
    if ( button->state == 0 )
        return;

/*
 *  Extract display number from arg list
 *  and determine which zoom command to process.
 */

    disp_num = (short)closure;
    command = Dm_Address->display[disp_num].dd_zoom;

/*
 *  Remove the plot callbacks.
 */

    for (i=0; i<Nbr_of_plots; i++) {
        plot_ptr = Plot_info_ptr + i;
        XtRemoveCallback(plot_ptr->draw_win,
                                        XmNinputCallback, cb_zoom, disp_num);
    }

/*
 *  Restore the cursor to normal
 */

    XDefineCursor (XtDisplay(Top), XtWindow(Top), None);

    if (Dm_Address->shell[disp_num])
        XDefineCursor(XtDisplay(Top),
                                XtWindow(Dm_Address->shell[disp_num]), None);

/*
 *  Determine which plot was selected by comparing
 *  the callback widget ID to the plot widget IDs.
 */

    i = 0;
    plot_ptr = Plot_info_ptr;
    while (widget != plot_ptr->draw_win && i < Nbr_of_plots) {
        plot_ptr = Plot_info_ptr + i;
        i++;
    }

    if (i > Nbr_of_plots) {
        tui_msg(M_YELLOW, "Error in processing zoom");
        return;
    }
```

```
    tmplt_ptr = plot_ptr->plot_pos;

/*
 *  If the command is Zoom Display, restore the cursor,
 *  obtain the focus point from the event structure,
 *  and compute the new world coordinate transformation
 *  factors and the zoom focus offset values.
 */

    if (command == ZOOM_DIS) {

/*
 *      If ESCAPE was selected, return.  Else, obtain the coordinates
 *      of the mouse cursor as the zoom focus point.

        if (calldata->event.xkey.keycode == ESCAPE_KEYCODE)
            return(0);
 */

        focus.x = button->x;
        focus.y = button->y;

/*
 *      Calculate the zoom coordinate offset values
 *      and the zoom world coordinate transformation factors.
 */

        Dm_Address->display[disp_num].dd_zfact = Zoom_factor;

        wc_base = 100.0 / Zoom_factor;

        tmplt_ptr->factor_x = (double) ((tmplt_ptr->drw_width - 1) / wc_base);
        tmplt_ptr->factor_y = (double)((tmplt_ptr->drw_height - 1) / wc_base);

        tmplt_ptr->offset_x = (short) ((50.0 * tmplt_ptr->org_factor_x)
                - ((focus.x / tmplt_ptr->org_factor_x) * tmplt_ptr->factor_x));

        tmplt_ptr->offset_y = (short) ((50.0 * tmplt_ptr->org_factor_y)
            - ((focus.y / tmplt_ptr->org_factor_y) * tmplt_ptr->factor_y));


/* RLK 11/19/90 This is partial code to insure an offset too small or large
                isn't calculated.

        if ((xaxis_ptr->org_low_val * tmplt_ptr->factor_x
                + tmplt_ptr->offset_x) > 0)
            tmplt_ptr->offset_x = 0;
        else if ((xaxis_ptr->org_high_val * tmplt_ptr->factor_x
                + tmplt_ptr->offset_x) < tmplt_ptr->drw_width)

        if ((yaxis_ptr->org_low_val * tmplt_ptr->factor_y
                + tmplt_ptr->offset_y) > 0)
            tmplt_ptr->offset_x = 0;
*/

/*
 *      Adjust current axis positions to insure
 *      axis remains visible.
 */

        axis_ptr = plot_ptr->axis;

        for (i=0; i<plot_ptr->header->xaxes_num; i++) {
            axis_ptr->cur_axis_pos = ((100.0 - axis_ptr->axis_pos)
```

```
                                * tmplt_ptr->factor_y) + tmplt_ptr->offset_y;

            if (axis_ptr->cur_axis_pos < 0)
                axis_ptr->cur_axis_pos = 0;
            else if (axis_ptr->cur_axis_pos > tmplt_ptr->drw_height - 1)
                axis_ptr->cur_axis_pos = tmplt_ptr->drw_height - 1;

        }

        axis_ptr = plot_ptr->axis + plot_ptr->header->xaxes_num;

        for (i=0; i<plot_ptr->header->yaxes_num; i++) {
            axis_ptr->cur_axis_pos = (short) (axis_ptr->axis_pos
                                * tmplt_ptr->factor_x) + tmplt_ptr->offset_x;

            if (axis_ptr->cur_axis_pos < 0)
                axis_ptr->cur_axis_pos = 0;
            else if (axis_ptr->cur_axis_pos > tmplt_ptr->drw_width - 1)
                axis_ptr->cur_axis_pos = tmplt_ptr->drw_width - 1;

        }

/*
 *  If the command is Zoom Reset, restore the original
 *  world coordinate transformation factors and zero
 *  the offset values.  Restore axis original positions.
 */

    } else if (command == ZOOM_RES) {

        tmplt_ptr->factor_x = tmplt_ptr->org_factor_x;
        tmplt_ptr->factor_y = tmplt_ptr->org_factor_y;
        tmplt_ptr->offset_x = 0;
        tmplt_ptr->offset_y = 0;

        axis_ptr = plot_ptr->axis;

        for (i=0; i<plot_ptr->header->xaxes_num; i++)
            axis_ptr->cur_axis_pos = axis_ptr->pixel_axis_pos;

        axis_ptr = plot_ptr->axis + plot_ptr->header->xaxes_num;

        for (i=0; i<plot_ptr->header->yaxes_num; i++)
            axis_ptr->cur_axis_pos = axis_ptr->pixel_axis_pos;
    }

/*
 *  Clear the plot command in shared memory.
 */

    Dm_Address->display[disp_num].dd_zoom = 0;

/*
 *  Clear the plot and issue a redraw on the plot.
 */

    XClearArea(Dm_Address->xdisplay[disp_num], XtWindow(plot_ptr->draw_win),
                    0, 0, tmplt_ptr->drw_width, tmplt_ptr->drw_height, False);

    draw_plt(disp_num, plot_ptr, 0, 0,
                                tmplt_ptr->drw_width, tmplt_ptr->drw_height);

    if (plot_ptr->prev_act_flg == YES) {
```

```
/*
 *      Reset all first point flags.
 */

        for (i=0; i<plot_ptr->header->actual_msids; i++)
            (plot_ptr->msids + i)->first_pt = YES;

/*
 *      Rewind plot data file to
 *      beginning of plot data.
 */

        lseek(plot_ptr->plot_fp, 0, SEEK_SET);
        lseek(plot_ptr->plot_fp,
                        80 + (plot_ptr->header->msid_num * 24), SEEK_SET);

/*
 *      Reset seconds elapsed to 0 (only meaningful for time plots).
 */

        plot_ptr->seconds_elapsed = 0;

/*
 *      Initialize end_of_file flag and call proc_plt() to plot
 *      data points.  Continue calls to proc_plt() until EOF, signalling
 *      all data points plotted.  Check for EOF necessary since
 *      proc_plt() and subordinate plot_msid() may exit prematurely due
 *      to an out-of-scale data point.
 */

        End_of_file = NO;

        while (End_of_file == NO)
            proc_plt(disp_num, plot_ptr);
    }

    D(printf("END cb_zoom\n"));
}
```

```
/*****************************************************************************
 * MODULE NAME: chg_gdr.c
 *
 *  This function allows the user to change GDR retrieval information. After
 *  all data is entered, the Data Handler is notified of the retrieval infor-
 *  mation change via a flag in shared memory. This function currently does
 *  not have any effect because GDR's are not handled by the stubbed data
 *  handler.
 *
 *
 * INTERNAL FUNCTIONS:
 *
 *      o   gdr_menu     -   This function presents the menu which allows the
 *                           GDR source to be changed.
 *
 *      o   cb_data      -   This function handles all callbacks generated by
 *                           the menu.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  K. Noonan        - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <ctype.h>
#include <X11/Intrinsic.h>
#include <X11/Shell.h>
#include <Xm/Text.h>
#include <constants.h>
#include <disp.h>
#include <user_inter.h>
#include <wex/EXmsg.h>


extern Widget               Top;                    /* Top level widget.              */
extern struct pfkey_defs    Current_Com;            /* current commands definition    */
extern struct dm_shmemory   *Dm_Address;            /* Display Manager shared memory  */

extern short                Disp_Num;               /* Display Manager number         */

extern char                 Disp_Path[DNAME_LEN];   /* Default path for ppl file      */


#define NUM_DURS            4
static char                 *durations[NUM_DURS] = { "S", "M", "H", "D" };

static struct ppl_record    new_ppl;

static Widget               t_ppl,
                            t_host,
                            t_rq,
                            t_duration,
                            t_ppl_rate,
                            r_dur_units,
                            r_ppl_units;
```

```
static short                flag,
                            ppl_nbr;


int chg_gdr ( )
{
    D(printf("START chg_gdr\n"));
/*
 *  Display a popup waiting for the user to enter data. Note that the (chg_gdr) function
 *  is not called from a function key or PBI.
 */

    gdr_menu ( );

/*
 *  If OK was selected, then store the information in shared memory and
 *  notify the Data Handler of the retrieval information update.
 */

    if ( flag ) {
        strcpy ( Dm_Address->ppl_recs[ppl_nbr].host_name,     new_ppl.host_name           )
;
        strcpy ( Dm_Address->ppl_recs[ppl_nbr].rate_units,    new_ppl.rate_units          )
;
        strcpy ( Dm_Address->ppl_recs[ppl_nbr].ppl_rate,      new_ppl.ppl_rate            )
;
        strcpy ( Dm_Address->ppl_recs[ppl_nbr].duration,      new_ppl.duration            )
;
        strcpy ( Dm_Address->ppl_recs[ppl_nbr].duration_units,
                                                              new_ppl.duration_units      )
;
        strcpy ( Dm_Address->ppl_recs[ppl_nbr].retrieval_qualifier,
                                                              new_ppl.retrieval_qualifier )
;
        Dm_Address->ppl_recs[ppl_nbr].update_retr = YES;
    }

/*
 *  Return the status of the popup.
 */

    D(printf("END chg_gdr\n"));
    return ( flag );
}
```

```
/*****************************************************************************
 * MODULE NAME: gdr_menu
 *
 *  This function actually presents the popup which allows the user to change
 *  GDR information.
 *****************************************************************************/

static int gdr_menu ( )
{
    register int    i;

    Arg             args[10];

    Widget          shell,
                    form,
                    f_ppl,
                    f_data,
                    f_cmd;

    XtCallbackProc  cb_data();

    XEvent          event;

    D(printf("START gdr_menu\n"));
/*
 *  Create the shell widget.
 */

    i = 0;
    shell = tui_create_trans_shell ( "Change GDR", args, i );

/*
 *  Create the main form and all sub-forms.
 */

    i = 0;
    form   = tui_create_form ( shell, "form",   TRUE,  args, i );
    f_ppl  = tui_create_form ( form,  "f_ppl",  FALSE, args, i );
    f_data = tui_create_form ( form,  "f_data", FALSE, args, i );
    f_cmd  = tui_create_form ( form,  "f_cmd",  FALSE, args, i );

/*
 *  Create a label and text widget to allow the user to specify the PPL
 *  filename.
 */

    i = 0;
    tui_create_label ( f_ppl, "l_ppl", "PPL File Name", args, i );
    t_ppl = tui_create_text ( f_ppl, "t_ppl", new_ppl.ppl_filename, DNAME_LEN-1,
                              XmSINGLE_LINE_EDIT, TRUE, args, i );

/*
 *  Create labels for the host name, retrieval qualifier, duration, duration
 *  units, PPL rate, and PPL rate units.
 */

    i = 0;
    tui_create_label ( f_data, "l_host",      "Host Name",           args, i );
    tui_create_label ( f_data, "l_rq",        "Retrieval Qualifier", args, i );
    tui_create_label ( f_data, "l_duration",  "Duration",            args, i );
    tui_create_label ( f_data, "l_dur_units", "Duration Units",      args, i );
    tui_create_label ( f_data, "l_ppl_rate",  "PPL Rate",            args, i );
    tui_create_label ( f_data, "l_ppl_units", "PPL Units",           args, i );
```

```
/*
 *  Create text widgets and radio boxes for the actual input fields for the
 *  host name, retrieval qualifier, duration, duration units, PPL rate, and
 *  PPL rate units.
 */

    i = 0;
    t_host      = tui_create_text ( f_data, "t_host",      new_ppl.host_name,
                                    3,  XmSINGLE_LINE_EDIT, TRUE, args, i );
    t_rq        = tui_create_text ( f_data, "t_rq",        new_ppl.retrieval_qualifier,
                                    10, XmSINGLE_LINE_EDIT, TRUE, args, i );
    t_duration  = tui_create_text ( f_data, "t_duration",  new_ppl.duration,
                                    3,  XmSINGLE_LINE_EDIT, TRUE, args, i );
    t_ppl_rate  = tui_create_text ( f_data, "t_ppl_rate",  new_ppl.ppl_rate,
                                    3,  XmSINGLE_LINE_EDIT, TRUE, args, i );

    r_dur_units = tui_create_rb ( f_data, "r_dur_units", durations, NUM_DURS,
                                  durations[0], args, i );
    r_ppl_units = tui_create_rb ( f_data, "r_ppl_units", durations, NUM_DURS,
                                  durations[0], args, i );

/*
 *  Create separator widgets.
 */

    i = 0;
    tui_create_separator ( form, "sep0", args, i );
    tui_create_separator ( form, "sep1", args, i );

/*
 *  Create pushbuttons with the appropriate callbacks.
 */

    i = 0;
    tui_create_pushbutton ( f_cmd, "OK",     cb_data, (caddr_t)1, args, i );
    tui_create_pushbutton ( f_cmd, "PPL",    cb_data, (caddr_t)2, args, i );
    tui_create_pushbutton ( f_cmd, "Cancel", cb_data, (caddr_t)0, args, i );
    tui_create_pushbutton ( f_cmd, "Help",   cb_data, (caddr_t)3, args, i );

/*
 *  Put all input widgets in a tab group. Note that at this time, radio boxes
 *  do not work, but may in the future.
 */

    XmAddTabGroup ( t_ppl       );
    XmAddTabGroup ( t_host      );
    XmAddTabGroup ( t_rq        );
    XmAddTabGroup ( t_duration  );
    XmAddTabGroup ( r_dur_units );
    XmAddTabGroup ( t_ppl_rate  );
    XmAddTabGroup ( r_ppl_units );

/*
 *  Realize, popup, and set the colormap of the shell.
 */

    XtRealizeWidget ( shell );
    XtPopup ( shell, None );
    set_cmap ( shell );

/*
 *  Wait in a loop until the user is finished with the popup.
 */
```

```
    flag = -1;
    while ( flag == -1 ) {
        XtNextEvent     ( &event );
        XtDispatchEvent ( &event );
    }

    XtDestroyWidget ( shell );

/*
 *  Normal return.
 */

    D(printf("END gdr_menu\n"));
    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: cb_data
 *
 *  This function processes commands from the popup. This includes OK, PPL,
 *  and CANCEL.
 *****************************************************************************/

/* ARGSUSED */
static XtCallbackProc cb_data ( w, closure, calldata )

    Widget          w;          /* Set to widget which in which callback originated.    */

    caddr_t         closure,    /* Indicates the selected button.                       */
                    *calldata;  /* Widget-specific information.                         */
{
    register int    i,
                    new_num;

    XtCallbackProc  cb_help();

    char            *fn,
                    *s,
                    string[DNAME_LEN + 5];

    short           match = NO;

    D(printf("START cb_data\n"));
/*
 *  Process OK button. If no valid PPL filename has been entered yet, generate a warning
 *  and return.
 */


    if ( (int)closure == 1 ) {
        if ( *new_ppl.ppl_filename == NULL ) {
            tui_msg ( M_YELLOW, "A valid PPL filename must be entered" );
            return;
        }

/*
 *      Retrieve and check the host name. Shift the hostname to uppercase and verify
 *      that it is valid. If not generate a warning and return. If valid, save in the
 *      (new_ppl) structure.
 */

        strcpy ( string, s = XmTextGetString ( t_host ) );
        free ( s );

        s = string;
        while ( *s ) {
            *s = toupper ( *s );
            s++;
        }
        if ( strcmp ( string, "MOC" ) && strcmp ( string, "DSC" ) ) {
            tui_msg ( M_YELLOW, "Invalid host name - input either 'MOC' or 'DSC'" );
            return;
        }
        strcpy ( new_ppl.host_name, string );

/*
 *      Retrieve the retrieval qualifier. No checking is required for this value.
 */
```

```
        strcpy ( new_ppl.retrieval_qualifier, s = XmTextGetString ( t_rq ) );
        free ( s );

    /*
     *      Retrieve and check the duration. Verify that all characters are digits
     *      and then verify that the value is in the range of 0 to 255. If not,
     *      generate a warning. If valid, save in the (new_ppl) structure.
     */

        strcpy ( string, s = XmTextGetString ( t_duration ) );
        free ( s );

        s = string;
        while ( *s )
            if ( isdigit ( *s++ ) == NO ) {
                tui_msg ( M_YELLOW, "Invalid duration - input integer between 0 and 255" )
    ;
                return;
            }

        new_num = atoi ( string );
        if ( new_num < 0 || new_num > 255 ) {
            tui_msg ( M_YELLOW, "Invalid duration - input integer between 0 and 255" );
            return;
        }
        strcpy ( new_ppl.duration, string );

    /*
     *      Retrieve and check the duration units. No verification is necessary because
     *      the user cannot select an invalid value.
     */

        strcpy ( new_ppl.duration_units, tui_radio_get_value ( r_dur_units ) );

    /*
     *      Retrieve and check the ppl rate. Verify that all characters are digits and then
     *      verify that the value is in the range of 0 to 255. If not, generate a warning.
     *      If valid, save in the (new_ppl) structure.
     */

        strcpy ( string, s = XmTextGetString ( t_ppl_rate ) );
        free ( s );

        s = string;
        while ( *s )
            if ( isdigit ( *s++ ) == NO ) {
                tui_msg ( M_YELLOW, "Invalid ppl rate - input integer between 0 and 255" )
    ;
                return;
            }

        new_num = atoi ( string );
        if ( new_num < 0 || new_num > 255 ) {
            tui_msg ( M_YELLOW, "Invalid ppl rate - input integer between 0 and 255" );
            return;
        }
        strcpy ( new_ppl.ppl_rate, string );

    /*
     *      Retrieve and check the ppl rate units. No verification is necessary because
     *      the user cannot select an invalid value.
     */

        strcpy ( new_ppl.rate_units, tui_radio_get_value ( r_ppl_units ) );
```

```
/*
 *  Everything is valid, so set (flag) to cause the popup to be removed.
 */

        flag = (int)closure;

/*
 *  The user selected the PPL button, so process entry of a PPL filename. First
 *  retrieve the filename and return if invalid.
 */

    } else if ( (int)closure == 2 ) {
        strcpy ( fn, s = XmTextGetString ( t_ppl ) );
        free ( s );
        if ( val_ppl ( fn ) == 0 )
            return;

/*
 *      If the filename is not path qualified, then add the default path to the
 *      filename.  If the filename is path qualified, then strip off the path name.
 *      The simple name of the file is used for the change retrieval information
 *      menu title.
 */

        if ( *fn != '/' ) {
            strcpy ( new_ppl.ppl_filename, Disp_Path );
            strcat ( new_ppl.ppl_filename, fn        );
        } else {
            strcpy ( new_ppl.ppl_filename, fn );
            get_fn ( new_ppl.ppl_filename, fn );
        }

/*
 *      Search for a match on the PPL filename that the retrieval information is
 *      to be changed. If match is found, copy the information in shared memory
 *      into local memory and call a routine to size the change retrieval
 *      information menu. If no match is found, then advise and remain in a loop
 *      to receive another ppl filename.
 */

        i     = 0;
        match = NO;
        while ( i < NUM_GDR && match == NO ) {
            if ( strcmp ( new_ppl.ppl_filename,
                          Dm_Address->ppl_recs[i].ppl_filename ) == 0 ) {
                match   = YES;
                ppl_nbr = i;
                strcpy ( new_ppl.host_name,      Dm_Address->ppl_recs[i].host_name      );
                strcpy ( new_ppl.rate_units,     Dm_Address->ppl_recs[i].rate_units     );
                strcpy ( new_ppl.duration_units, Dm_Address->ppl_recs[i].duration_units );
                strcpy ( new_ppl.ppl_rate,       Dm_Address->ppl_recs[i].ppl_rate       );
                strcpy ( new_ppl.duration,       Dm_Address->ppl_recs[i].duration       );
                strcpy ( new_ppl.retrieval_qualifier,
                                          Dm_Address->ppl_recs[i].retrieval_qualifier );
            } else
                i++;
        }

        if ( match == NO )
            tui_msg ( M_YELLOW, "PPL file %s is not active", new_ppl.ppl_filename );

/*
 *  Process CANCEL button.
```

```
 */

    } else if ( (int)closure == 0 ) {
        flag = (int)closure;

/*
 *  If help button was selected, display appropriate help text.
 */

    } else if ( (int)closure == 3 )
        cb_help ( (Widget)0, (caddr_t)11, (caddr_t)0 );

    D(printf("END cb_data\n"));
    return;
}
```

```
/*****************************************************************************
 * MODULE NAME: chg_lim.c
 *
 *  This function is invoked when the user selects the change limits option
 *  from the utilities menu or a PF key is entered to change the limits. This
 *  function allows the user to retrieve limits for a given MSID and then
 *  change the limits themselves and enable/disable alarms and advisories.
 *
 * INTERNAL FUNCTIONS:
 *
 *      o   chgl_menu    -   This function presents the form which allows
 *                           the limits to be reviewed and changed.
 *
 *      o   cb_limit     -   This callback function processes each of the
 *                           commands from the popup form.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  K. Noonan        - Ford Aerospace Corporation
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <X11/Shell.h>
#include <Xm/Xm.h>
#include <Xm/MessageB.h>
#include <Xm/Text.h>
#include <constants.h>
#include <disp.h>
#include <DDdisp.h>
#include <pf_key.h>
#include <user_inter.h>
#include <wex/EXmsg.h>


extern Widget               Top;            /* The top level widget.              */
extern struct msid_ent      *Msid;          /* List of MSID entries.              */
extern struct pfkey_defs    Current_Com;    /* Current commands definition        */
extern struct dm_shmemory   *Dm_Address;    /* Display manager shared memory.     */

extern short                Disp_Num;       /* Display manager number             */

extern char                 **Msid_list_lim;/* List of MSID strings for limits.   */

extern int                  Msid_num_lim;   /* Number of MSID's available for limits.  */


static Widget               shell,
                            t_msid,
                            tl [4],
                            tg [4],
                            tga[4];

static int                  flag;
```

```
static struct disp_info     *display;


int chg_lim ( )
{
    short           error;                  /* Return value from chk_flg.          */

    D(printf("START chg_lim\n"));
/*
 *  Save pointer to the display structure and initialize the display information
 *  limit change index to INVALID.
 */

    display = &Dm_Address->display[Disp_Num];
    display->limits.indx = INVALID;

/*
 *  If called from menu, clear command values and display the popup.
 */

    if ( Current_Com.func_no == LIM_MENU ) {
        Current_Com.limit_change.msid  [0] = '\0';
        Current_Com.limit_change.src   [0] = '\0';
        Current_Com.limit_change.option[0] = '\0';
        chgl_menu ( );

/*
 *  Otherwise, new limits were already given through a PF key selection.  Store the
 *  limits and alarm commands from the current command structure into the display
 *  information table for the Data Handler. Copy the source and MSID into the
 *  display information table.  Set the finished flag to YES so the limit change
 *  menu will not be redrawn and set the advise flag to YES to advise of
 *  successful limit changes
 */

    } else {
#if FAC == NO
        i = strlen ( Current_Com.limit_change.msid );
        for ( j = i; j < MSID_LENGTH; j++ )
            Current_Com.limit_change.msid[j] = ' ';
        Current_Com.limit_change.msid[MSID_LENGTH] = 0;
#endif
        strcpy ( display->limits.msid_name, Current_Com.limit_change.msid );
        strcpy ( display->limits.src, Current_Com.limit_change.src );
        strcpy ( display->limits.option, Current_Com.limit_change.option );
        if ( ( Current_Com.limit_change.flag == 0 ) ||
             ( Current_Com.limit_change.flag == 2 ) ) {
            display->limits.low_limit = Current_Com.limit_change.ops_ll;
            display->limits.ol_alrm   = Current_Com.limit_change.ol_alrm;
            display->limits.ol_adv    = Current_Com.limit_change.ol_adv;
            display->limits.hi_limit  = Current_Com.limit_change.ops_ul;
            display->limits.oh_alrm   = Current_Com.limit_change.oh_alrm;
            display->limits.oh_adv    = Current_Com.limit_change.oh_adv;
            display->limits.limt_flag = YES;
        } else {
            display->limits.limt_flag = NO;
        }

        if ( ( Current_Com.limit_change.flag == 1 ) ||
             ( Current_Com.limit_change.flag == 2 ) ) {
            display->limits.crit_low  = Current_Com.limit_change.crit_ll;
            display->limits.cl_alrm   = Current_Com.limit_change.cl_alrm;
            display->limits.cl_adv    = Current_Com.limit_change.cl_adv;
```

```
            display->limits.crit_high = Current_Com.limit_change.crit_ul;
            display->limits.ch_alrm   = Current_Com.limit_change.ch_alrm;
            display->limits.ch_adv    = Current_Com.limit_change.ch_adv;
            display->limits.crit_flag = YES;
        } else {
            display->limits.crit_flag = NO;
        }

/*
 *      New limits were given through a PF key selection then set the command in the
 *      display information table for the Data Handler.  Wait for a response and advise
 *      if no response is given.  An advisory is given for updated limits if the command
 *      came from a PF key with the limits specified.
 */

        display->limits.updated = NO;
        display->upd_lim        = YES;
        error = chk_flg ( &display->limits.updated, 5, 1 );

        if ( error )
            tui_msg ( M_YELLOW, "Error in msid %s limit update",display->limits.msid_name
);
        else
            tui_msg ( M_WHITE, "MSID %s limits updated", display->limits.msid_name );
    }

    D(printf("END chg_lim\n"));
    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: chgl_menu
 *
 *  This function presents the actual popup form which allows limits to be
 *  changed.
 *****************************************************************************/

static int chgl_menu ( )
{
    register int    i;

    Arg             args[10];

    Widget          form,
                    f_msid,
                    f_data,
                    f_cmd;

    XtCallbackProc  cb_limit();

    XEvent          event;

    D(printf("START chgl_menu\n"));
/*
 *  Create the shell widget.
 */

    i = 0;
    shell = tui_create_trans_shell ( "Change Limits", args, i );

/*
 *  Create the main and all sub-forms.
 */

    i = 0;
    form   = tui_create_form ( shell, "form",   TRUE,  args, i );
    f_msid = tui_create_form ( form,  "f_msid", FALSE, args, i );
    f_data = tui_create_form ( form,  "f_data", FALSE, args, i );
    f_cmd  = tui_create_form ( form,  "f_cmd",  FALSE, args, i );

/*
 *  Create all widgets. First create a selection list which allows the user to select
 *  the desired MSID to change limits for.
 */

    i = 0;
    t_msid = tui_create_sel ( f_msid, "t_msid", Msid_list_lim, Msid_num_lim, "MSID's",
                              args, i );

/*
 *  Create labels for the high and low values for operational and critical limits.
 */

    i = 0;
    tui_create_label ( f_data, "l_opslow",     "Ops Low",  args, i );
    tui_create_label ( f_data, "l_opshigh",    "Ops HI",   args, i );
    tui_create_label ( f_data, "l_critlow",    "Crit Low", args, i );
    tui_create_label ( f_data, "l_crithigh",   "Crit HI",  args, i );

/*
 *  Create text widget for the high and low values for operational and critical limits.
 */
```

```
    i = 0;
    tl[0] = tui_create_text ( f_data, "t_opslow",    "", 14, XmSINGLE_LINE_EDIT, TRUE,
                              args, i );
    tl[1] = tui_create_text ( f_data, "t_opshigh",   "", 14, XmSINGLE_LINE_EDIT, TRUE,
                              args, i );
    tl[2] = tui_create_text ( f_data, "t_critlow",   "", 14, XmSINGLE_LINE_EDIT, TRUE,
                              args, i );
    tl[3] = tui_create_text ( f_data, "t_crithigh",  "", 14, XmSINGLE_LINE_EDIT, TRUE,
                              args, i );

/*
 *  Create toggles for the alarm and advisory flags for each limit value.
 */

    i = 0;
    tg [0] = tui_create_toggle ( f_data, "tg_opslow",    "Alarm",    FALSE, NULL, 0, args,i
);
    tg [1] = tui_create_toggle ( f_data, "tg_opshigh",   "Alarm",    FALSE, NULL, 0, args,i
);
    tg [2] = tui_create_toggle ( f_data, "tg_critlow",   "Alarm",    FALSE, NULL, 0, args,i
);
    tg [3] = tui_create_toggle ( f_data, "tg_crithigh",  "Alarm",    FALSE, NULL, 0, args,i
);
    tga[0] = tui_create_toggle ( f_data, "tg_opslow_a",  "Advisory",FALSE, NULL, 0, args,i
);
    tga[1] = tui_create_toggle ( f_data, "tg_opshigh_a", "Advisory",FALSE, NULL, 0, args,i
);
    tga[2] = tui_create_toggle ( f_data, "tg_critlow_a", "Advisory",FALSE, NULL, 0, args,i
);
    tga[3] = tui_create_toggle ( f_data, "tg_crithigh_a","Advisory",FALSE, NULL, 0, args,i
);

/*
 *  Create separator widgets.
 */

    i = 0;
    XtManageChild ( XmCreateSeparator ( form, "sep0", args, i ) );
    XtManageChild ( XmCreateSeparator ( form, "sep1", args, i ) );

/*
 *  Create command widgets with the appropriate callbacks.
 */

    i = 0;
    tui_create_pushbutton ( f_cmd, "Done", cb_limit, (caddr_t)0, args, i );
    tui_create_pushbutton ( f_cmd, "Save", cb_limit, (caddr_t)1, args, i );
    tui_create_pushbutton ( f_cmd, "MSID", cb_limit, (caddr_t)2, args, i );
    tui_create_pushbutton ( f_cmd, "Help", cb_limit, (caddr_t)3, args, i );

/*
 *  Put all input widgets in a tab group.
 */

    XmAddTabGroup ( t_msid );

    for ( i = 0; i < 4; i++ ) {
        XmAddTabGroup ( tl [i] );
        XmAddTabGroup ( tg [i] );
        XmAddTabGroup ( tga[i] );
    }

/*
 *  Realize and popup the shell.
```

```
 */

    XtRealizeWidget ( shell );
    XtPopup ( shell, None );
    set_cmap ( shell );

/*
 *  Wait until the user finishes with the popup.
 */

    flag = -1;
    while ( flag == -1 ) {
        XtNextEvent     ( &event );
        XtDispatchEvent ( &event );
    }

    XtDestroyWidget ( shell );

/*
 *  Return the value selected by the user (0 is for not verified, 1 is for
 *  verified.
 */

    D(printf("END chgl_menu\n"));
    return ( flag );
}
```

```
/*****************************************************************************
 * MODULE NAME: cb_limit
 *
 *  This function is called when the user selects any of the commands from the
 *  popup form.
 *****************************************************************************/

/* ARGSUSED */
static XtCallbackProc cb_limit ( w, closure, calldata )

    Widget          w;          /* Set to widget which in which callback originated.    */

    caddr_t         closure,    /* Indicates the selected button.                       */
                    *calldata;  /* Widget-specific information.                         */
{
    register int    i;

    XtCallbackProc  cb_help();

    char            *ptr,
                    new_src [4],
                    real_src[4],
                    limit   [4][17];

    short           alarm_flg,
                    error,
                    advise,
                    pos_indx;

    D(printf("START cb_limit\n"));
/*
 *  Process SAVE button. If the user did not specify an MSID yet, generate a
 *  warning and return.
 */

    if ( (int)closure == 1 ) {
        if ( *Current_Com.limit_change.msid == '\0' ) {
            tui_msg ( M_YELLOW, "Use MSID to specify the MSID to update" );
            return;
        }

/*
 *      Retrieve and verify the new limits. First retrieve the operational low limit.
 */

        display->limits.limt_flag = NO;
        ptr = XmTextGetString ( tl[0] );
        if ( limit_val ( ptr ) ) {
            sscanf ( ptr, "%lf", &display->limits.low_limit );
            display->limits.limt_flag = YES;
            free ( ptr );
        } else {
            free ( ptr );
            tui_msg ( M_YELLOW, "Invalid value for low limit" );
            return;
        }

/*
 *      Retrieve and verify the operational high limit.
 */

        ptr = XmTextGetString ( tl[1] );
        if ( *ptr && limit_val ( ptr ) ) {
```

```
            sscanf ( ptr, "%lf", &display->limits.hi_limit );
            display->limits.limt_flag = YES;
            free ( ptr );
        } else {
            free ( ptr );
            tui_msg ( M_YELLOW, "Invalid value for high limit" );
            return;
        }

/*
 *      Retrieve and verify the critical low limit.
 */

        display->limits.crit_flag = NO;
        ptr = XmTextGetString ( tl[2] );
        if ( limit_val ( ptr ) ) {
            sscanf ( ptr, "%lf", &display->limits.crit_low );
            display->limits.crit_flag = YES;
            free ( ptr );
        } else {
            free ( ptr );
            tui_msg ( M_YELLOW, "Invalid value for critical low limit" );
            return;
        }

/*
 *      Retrieve and verify the critical high limit.
 */

        ptr = XmTextGetString ( tl[3] );
        if ( limit_val ( ptr ) ) {
            sscanf ( ptr, "%lf", &display->limits.crit_high );
            display->limits.crit_flag = YES;
            free ( ptr );
        } else {
            free ( ptr );
            tui_msg ( M_YELLOW, "Invalid value for critical high limit" );
            return;
        }

/*
 *      Retrieve the values for the alarm and advisory flags.
 */

        display->limits.ol_alrm = XmToggleButtonGadgetGetState ( tg [0] );
        display->limits.oh_alrm = XmToggleButtonGadgetGetState ( tg [1] );
        display->limits.cl_alrm = XmToggleButtonGadgetGetState ( tg [2] );
        display->limits.ch_alrm = XmToggleButtonGadgetGetState ( tg [3] );

        display->limits.ol_adv  = XmToggleButtonGadgetGetState ( tga[0] );
        display->limits.oh_adv  = XmToggleButtonGadgetGetState ( tga[1] );
        display->limits.cl_adv  = XmToggleButtonGadgetGetState ( tga[2] );
        display->limits.ch_adv  = XmToggleButtonGadgetGetState ( tga[3] );

/*
 *      If the position Id alarm flag has been disabled and one of the limits alarms
 *      has been enabled, then advise that the alarm will not be enabled.
 */

        pos_indx  = display->pos_id_indx;
        alarm_flg = Dm_Address->process.alarm[pos_indx];
        if ( alarm_flg == NO ) {
            i = 0;
            advise = NO;
```

```
            while ( advise == NO && i < 4 ) {
                if ( XmToggleButtonGadgetGetState ( tga[i] ) ) {
                    tui_msg ( M_WHITE, "Limit alarm not enabled - pos. Id alarm disabled"
);
                    advise = YES;
                }
                i++;
            }
        }

/*
 *      Set the command in the display information table for the Data Handler.  Wait
 *      for a response and advise if no response is given.  An advisory is given for
 *      updated limits if the command came from a PF key with the limits specified.
 */

        display->limits.updated = NO;
        display->upd_lim        = YES;
        error = chk_flg ( &display->limits.updated, 5, 1 );

        if ( error )
            tui_msg ( M_YELLOW, "Error in msid %s limit update",display->limits.msid_name
);
        else
            tui_msg ( M_WHITE, "MSID %s limits updated", display->limits.msid_name );

/*
 *  Process the CANCEL button. Simply set the global flag to 0.
 */

    } else if ( (int)closure == 0 ) {
        flag = (int)closure;

/*
 *  Process the MSID button. First retrieve the new MSID and if necessary, pad with
 *  blanks.
 */

    } else if ( (int)closure == 2 ) {
        ptr = XmTextGetString ( t_msid );

#if FAC == NO
        for ( i = strlen ( ptr ); i < MSID_LENGTH; i++ )
            *(ptr+i) = ' ';
        *(ptr+i) = '\0';
#endif

/*
 *      Verify that the specified MSID is really one in the list. Upon return from the
 *      (val_msid) function, (i) will be set to the index of the MSID in the (Msid)
 *      list.
 */

        if ( ( i = val_msid ( Msid_list_lim, Msid_num_lim, ptr ) ) == -1 ) {
            free ( ptr );
            return;
        }

/*
 *      MSID is valid, so first clear out all limit flags and values.
 */

        *display->limits.msid_name =
        *display->limits.src       =
```

```
        *display->limits.option     = '\0';
        display->limits.low_limit   =
        display->limits.hi_limit    =
        display->limits.crit_low    =
        display->limits.crit_high   = 0.0;
        display->limits.ol_alrm     =
        display->limits.ol_adv      =
        display->limits.oh_alrm     =
        display->limits.oh_adv      =
        display->limits.limt_flag   =
        display->limits.cl_alrm     =
        display->limits.cl_adv      =
        display->limits.ch_alrm     =
        display->limits.ch_adv      =
        display->limits.crit_flag   = FALSE;

/*
 *      Copy the specified MSID into the current command structure and free temporary
 *      storage.
 */

        strcpy ( Current_Com.limit_change.msid, ptr );
        free ( ptr );

/*
 *      Save the corresponding SOURCE for the specified MSID. Verify that the source is
 *      valid.
 */

        strcpy ( new_src, (Msid+i)->Data_Src );
        if ( val_src ( new_src, real_src ) == 0 ) {
            tui_msg ( M_YELLOW, "Invalid data source" );
            return;
        }

        strcpy ( Current_Com.limit_change.src, new_src );
        if ( ( strcmp ( real_src, "PPM" ) == 0 ) || ( strcmp ( real_src, "EVN" ) == 0 ) )
{
            strcpy ( Current_Com.limit_change.src,    real_src );
            strcpy ( Current_Com.limit_change.option, new_src );
        } else {
            strcpy ( Current_Com.limit_change.src, new_src );
            Current_Com.limit_change.option[0] = '\0';
        }

/*
 *      Copy the source and MSID into the display information table and set the command
 *      flag for the Data Handler.
 */

        strcpy ( display->limits.msid_name, Current_Com.limit_change.msid   );
        strcpy ( display->limits.src,       Current_Com.limit_change.src    );
        strcpy ( display->limits.option,    Current_Com.limit_change.option );
        display->get_lim        = YES;
        display->limits.updated = NO;

/*
 *      Wait for the limits to be returned from the Data Handler. If an error occurs,
 *      generate a warning and return.
 */

        if ( chk_flg ( &display->limits.updated, 5, 1 ) != 0 ) {
            tui_msg ( M_YELLOW, "Unable to get MSID %s limit information",
                    display->limits.msid_name );
```

```
            return;
        }

    /*
     *      At this point, updated limit values have been returned. Initialized the text
     *      widgets with the appropriately formated limits.
     */

        sprintf ( limit[0], "%e", display->limits.low_limit );
        sprintf ( limit[1], "%e", display->limits.hi_limit  );
        sprintf ( limit[2], "%e", display->limits.crit_low  );
        sprintf ( limit[3], "%e", display->limits.crit_high );

        XmTextSetString ( tl[0], limit[0] );
        XmTextSetString ( tl[1], limit[1] );
        XmTextSetString ( tl[2], limit[2] );
        XmTextSetString ( tl[3], limit[3] );

    /*
     *      Set toggles based on the state of the alarm flags.
     */

        XmToggleButtonGadgetSetState ( tg [0], display->limits.ol_alrm, FALSE );
        XmToggleButtonGadgetSetState ( tg [1], display->limits.oh_alrm, FALSE );
        XmToggleButtonGadgetSetState ( tg [2], display->limits.cl_alrm, FALSE );
        XmToggleButtonGadgetSetState ( tg [3], display->limits.ch_alrm, FALSE );

    /*
     *      Set toggles based on the state of the advisory flags.
     */

        XmToggleButtonGadgetSetState ( tga[0], display->limits.ol_adv, FALSE );
        XmToggleButtonGadgetSetState ( tga[1], display->limits.oh_adv, FALSE );
        XmToggleButtonGadgetSetState ( tga[2], display->limits.cl_adv, FALSE );
        XmToggleButtonGadgetSetState ( tga[3], display->limits.ch_adv, FALSE );

    /*
     *  If help button was selected, display appropriate help text.
     */

    } else if ( (int)closure == 3 )
        cb_help ( (Widget)0, (caddr_t)17, (caddr_t)0 );

    D(printf("END cb_limit\n"));
    return;
}
```

```
/*****************************************************************************
 * MODULE NAME: chg_zoom.c
 *
 *  This function allows the user to change the zoom factor. At this time,
 *  this value is not used.
 *
 *
 * INTERNAL FUNCTIONS:
 *
 *  o   cb_chg_zoom      -   Callback function which processes all callbacks
 *                           from the form.
 *
 *  o   chg_zoom_menu    -   This function displays the popup and waits for the
 *                           user to enter the new zoom factor.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  K. Noonan        - Ford Aerospace Corporation
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <X11/Intrinsic.h>
#include <Xm/Scale.h>
#include <constants.h>
#include <disp.h>
#include <user_inter.h>
#include <wex/EXmsg.h>


extern Widget               Top;            /* Top level widget for attaching popup.    */
extern struct dm_shmemory   *Dm_Address;    /* Shared memory area                       */

extern short                Disp_Num;       /* Display Manager number                   */

extern float                Zoom_factor;    /* current zoom factor                      */


static Widget               scale;

static int                  flag;

#define NUM_LABELS          2
static char                 *labels[NUM_LABELS] = { "0.01", "9.90" };


int chg_zoom ( )
{
    D(printf("START chg_zoom\n"));
/*
 *  Call the menu function to allow the user to update the zoom factor.
 */

    chg_zoom_menu ( );

/*
```

```
     *  Return the status of the popup.
     */

        D(printf("END chg_zoom\n"));
        return ( flag );
    }
```

```
/*****************************************************************************
 * MODULE NAME: chg_zoom_menu
 *
 *  This function displays the form and waits for the user to selecte the new
 *  zoom factor.
 *****************************************************************************/

static int chg_zoom_menu ( )
{
    register int    i;

    Arg             args[10];

    Widget          shell, form, f_data, f_cmd;

    XtCallbackProc  cb_chg_zoom();

    XEvent          event;

    D(printf("START chg_zoom_menu\n"));
/*
 *  Create the shell widget.
 */

    i = 0;
    shell = tui_create_trans_shell ( "Change Zoom Factor", args, i );

/*
 *  Create the main and all sub-forms.
 */

    i = 0;
    form   = tui_create_form ( shell, "form",   TRUE,  args, i );
    f_data = tui_create_form ( form,  "f_data", FALSE, args, i );
    f_cmd  = tui_create_form ( form,  "f_cmd",  FALSE, args, i );

/*
 *  Create all widgets. Create a label widget which identifies the scale. Create a
 *  scale widget which allows the new value to be selected. Note that the limits of
 *  the scale are 1 to 990. This is because the scale only handles integer values.
 *  The 1 and 990 correspond to 0.01 and 9.90. A resource is used to force the
 *  current value to be displayed as the proper decimal value.
 */

    i = 0;
    tui_create_label ( f_data, "label", "Zoom factor", args, i );

    i = 0;
    scale = tui_create_scale ( f_data, "scale", 1, 990, (int)(Zoom_factor*100.0),
                               labels, 2, args, i );

/*
 *  Create separator widget.
 */

    i = 0;
    XtManageChild ( XmCreateSeparator ( form, "sep0", args, i ) );

/*
 *  Create all command widgets with the appropriate callbacks.
 */

    i = 0;
```

```
    tui_create_pushbutton ( f_cmd, "Cancel", cb_chg_zoom, (caddr_t)0, args, i );
    tui_create_pushbutton ( f_cmd, "OK",     cb_chg_zoom, (caddr_t)1, args, i );
    tui_create_pushbutton ( f_cmd, "Help",   cb_chg_zoom, (caddr_t)2, args, i );

/*
 *  Realize and popup the shell.
 */

    XtRealizeWidget ( shell );
    XtPopup ( shell, None );
    set_cmap ( shell );

/*
 *  Wait until the user finishes with the popup.
 */

    flag = -1;
    while ( flag == -1 ) {
        XtNextEvent     ( &event );
        XtDispatchEvent ( &event );
    }

    XtDestroyWidget ( shell );

/*
 *  Return the value selected by the user (0 is for not verified, 1 is for
 *  verified.
 */

    D(printf("END chg_zoom_menu\n"));
    return ( flag );
}
```

```
/*****************************************************************************
 * MODULE NAME: cb_chg_zoom
 *
 *  This callback function is called when the user selects one of the buttons
 *  on the form.
 *
 *****************************************************************************/

/* ARGSUSED */
static XtCallbackProc cb_chg_zoom ( w, closure, calldata )

    Widget          w;          /* Set to widget which in which callback originated.    */

    caddr_t         closure,    /* Indicates the selected button.                       */
                    *calldata;  /* Widget-specific information.                         */
{
    XtCallbackProc  cb_help();

    int             zoom_factor;

    D(printf("START cb_chg_zoom\n"));
/*
 *  Process OK button. Retrieve the value of the scale. Divide the value by 100 and
 *  save in the extern variable (Zoom_factor) and in the display structure. Note
 *  that the scale returns values in the range of 1 to 990.
 */

    if ( (int)closure == 1 ) {
        XmScaleGetValue ( scale, &zoom_factor );
        Zoom_factor = zoom_factor / 100.0;
        Dm_Address->display[Disp_Num].dd_zfact = Zoom_factor;
        flag = (int)closure;

/*
 *  Process CANCEL button. Simply set (flag) to the value of (closure).
 */

    } else if ( (int)closure == 0 ) {
        flag = (int)closure;

/*
 *  If help button was selected, display the appropriate help text.
 */

    } else if ( (int)closure == 2 )
        cb_help ( (Widget)0, (caddr_t)24, (caddr_t)0 );

    D(printf("END cb_chg_zoom\n"));
    return;
}
```

```
/*****************************************************************************
 * MODULE NAME: chk_flg.c
 *
 *  This function sets up a counter using a timing mechanism to monitor the
 *  setting of a flag. The flag is usually a flag used for communication be-
 *  tween the Display Manager and the Data Handler.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  K. Noonan       - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <constants.h>
#include <wex/EXmsg.h>


int chk_flg ( flag, wait_count, value )

    short           *flag,          /* flag to monitor                      */
                     wait_count,    /* max. number of pauses to do          */
                     value;         /* end value of the passed flag         */
{
    short            pause_cnt = 0; /* counts number of astpauses           */

    D(printf("START chk_flg\n"));
/*
 *  Display the wait cursor to inform the user that a time-consuming operation
 *  is about to take place.
 */

    tui_start_wait ( );

/*
 *  Monitor the passed flag to change to the value requested.  If no change then
 *  pause one second and check the flag again.
 */

    while ( ( *flag != value ) && ( pause_cnt < wait_count ) ) {
#ifdef SUN
        usleep ( 1000000 );
#else
        astpause ( 0, 1000 );
#endif
        pause_cnt++;
    }

/*
 *  Restore the default cursor.
 */

    tui_stop_wait ( );

/*
 *  If the flag was not set within the specified time period, return an error.
```

```
 */

    if ( *flag != value )
        return ( -1 );

/*
 *  Normal return.
 */

    D(printf("END chk_flg\n"));
    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: chk_flt.c
 *
 *  This function validates the flight ID for a given display by checking the
 *  existing flgiht ID's and/or if a new flight ID can be added to the work-
 *  station.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  K. Noonan       - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <constants.h>
#include <disp.h>
#include <wex/EXmsg.h>


extern struct dm_shmemory   *Dm_Address;    /* address of shared memory                 */
extern short                Disp_Num,       /* slot nbr where the display info is       */
                            Flt_Selected;   /* Yes, if flight and data type have been   */


int chk_flt ( add )

    short               add;                /* set to yes if a flt is to be added       */
{
    struct disp_info    *display;           /* ptr to current display information       */

    int                 error = NO;         /* return value for valid flt info          */

    short               match = NO,         /* set to yes if match is found             */
                        i,                  /* counter                                  */
                        strm_nbr;           /* stream number of the flight              */

    D(printf("START chk_flt\n"));
/*
 *  Check the flight Id and stream type that are in the Display Manager
 *  shared memory for each stream with the flight Id and stream type for the
 *  new display.    If there is no match then, if the number of streams is less
 *  than the number of flights allowed, then store the flight Id and stream
 *  type of the display in the Display Manager shared memory.   If the number
 *  of stream is the maximum already, advise and return an error.   If a match
 *  is found, then store the stream Id in the display information table.    Set
 *  the flight connection flag to YES.  This is used when a user wants to
 *  add another flight and stream type. If other displays are using the both
 *  the flight and stream type, then the user will not be allowed to add
 *  another flight and stream type.
 */

    display = &Dm_Address->display[Disp_Num];
    for ( i = 0; i < MAX_FLTS; i++ ) {
#if FAC == YES
        if ( ( ( strncmp ( Dm_Address->strm[i].flt_id, display->flight_id, 4 ) ) == 0 )
            && ( strncmp ( Dm_Address->strm[i].strm_type, display->strm_type, 2 ) ) == 0 )
```

```
#else
        if ( ( ( strncmp ( Dm_Address->strm[i].flt_id, display->flight_id, 3 ) ) == 0 )
            && ( strncmp ( Dm_Address->strm[i].strm_type, display->strm_type, 2 ) ) == 0
#endif
        {
            if ( ( display->flight_id[0] != 0 ) && ( display->strm_type[0] != 0 ) ) {
                match = YES;
                strm_nbr = i;
                if ( add == YES ) {
                    display->strm_no = i;
                    Flt_Selected = YES;
                    Dm_Address->strm[strm_nbr].nbr_conn++;
                }
                break;
            }
        }
    }

    if ( ( match == NO ) && ( add == YES ) ) {
        if ( Dm_Address->process.nbr_streams < MAX_FLTS ) {
            for ( i = 0; i < MAX_FLTS; i++ ) {
                if ( Dm_Address->strm[i].nbr_conn <= 0 ) {
                    strm_nbr = i;
                    break;
                }
            }
#if FAC == YES
            strncpy ( Dm_Address->strm[strm_nbr].flt_id, display->flight_id, 4 );
#else
            strncpy ( Dm_Address->strm[strm_nbr].flt_id, display->flight_id, 3 );
#endif
            strncpy ( Dm_Address->strm[strm_nbr].strm_type, display->strm_type, 2 );
            display->strm_no = strm_nbr;
            Dm_Address->process.nbr_streams++;
            Dm_Address->strm[strm_nbr].nbr_conn++;
            Flt_Selected = YES;
        } else {
            tui_msg ( M_YELLOW, "Maximum number of flights exceeded on this workstation" )
;
            Flt_Selected = NO;
            error = YES;
        }
    }

/*
 *  If a match was found and the flight is not to be added, but taken away,
 *  then decrement the number of connects to the stream.    If this was the only
 *  display using this stream, then decrement the number of streams active.
 */

    if ( ( match == YES ) && ( add == NO ) ) {
        Dm_Address->strm[strm_nbr].nbr_conn--;
        if ( Dm_Address->strm[strm_nbr].nbr_conn <= 0 ) {
            Dm_Address->strm[strm_nbr].flt_id[0] = 0;
            Dm_Address->strm[strm_nbr].strm_type[0] = 0;
            Dm_Address->process.nbr_streams--;
        }
    }

    D(printf("END chk_flt\n"));
    return ( error );
}
```

```
/*****************************************************************************
 * MODULE NAME: chk_res.c
 *
 *  This function checks the access restriction code for files that have been
 *  restricted by medical or payload users.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  K. Noonan       - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <constants.h>
#include <disp.h>
#include <wex/EXmsg.h>


int chk_res ( access, pos_id )

    short           access;     /* Access restriction code.                  */

    char            *pos_id;    /* Position Id to validate against.          */
{
    short           restricted; /* Return flag.                              */

    D(printf("START chk_res\n"));
/*
 *  Check to see if MEDICAL or PAYLOAD access restricted. If so, then advise.
 */

    restricted = NO;
    switch ( access ) {
    case MEDICAL_USR:
        if ( ( strcmp ( pos_id, "MED\0" ) != 0 ) ) {
            tui_msg ( M_YELLOW, "Medical file - access restricted" );
            restricted = YES;
        }
        break;
    case PAYLOAD_USR:
        if ( ( strcmp ( pos_id, "PAY\0" ) != 0 ) ) {
            tui_msg ( M_YELLOW, "Payload file - access restricted" );
            restricted = YES;
        }
        break;
    default:
        break;
    }

/*
 *  Return the (restricted) flag.
 */

    D(printf("END chk_res\n"));
    return ( restricted );
}
```

```
/*****************************************************************************
 * MODULE NAME: cleanup.c
 *
 *  This function does all processing necessary to cause the Display Manager
 *  to exit.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  K. Noonan       - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <X11/Xlib.h>
#include <X11/Intrinsic.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <signal.h>
#include <constants.h>
#include <disp.h>
#include <wex/EXmsg.h>


extern Widget               Top;            /* The top level widget.          */
extern struct dm_shmemory   *Dm_Address;    /* address of Display Manager shm */

extern short                Disp_Num,       /* slot nbr where the display info is  */
                            Msg_Popup_Flag; /* Controls display of message popups. */

extern int                  errno,          /* system return error value      */
                            Dm_Id;          /* Display Manager SHM Id         */


int cleanup ( )
{
    short                   error,          /* return value from function calls */
                            disp_init = NO, /* YES, if a display was up         */
                            pos_indx;       /* position Id index                */

    D(printf("START cleanup\n"));
/*
 *  If the Displayer task is active, set the halt flag in shared memory.    Then
 *  call the check flag routine check for the displayer halt acknowledge flag
 *  to be set by the Displayer task.    If the Displayer task has not halted then
 *  kill the task.
 */

    if ( Dm_Address->display[Disp_Num].disp_init ||
         Dm_Address->display[Disp_Num].disp_pid != -1 ) {
        Dm_Address->display[Disp_Num].active_display = NO;
        Dm_Address->display[Disp_Num].halt = YES;
        disp_init = YES;

        exit_disp ( Disp_Num );
    }
```

```
/*
 *  Decrement the number of active display tasks for this workstation and the
 *  number of Display Managers for the position Id. Clear the position Id in
 *  the table if no more users for the position Id. If this is not the last
 *  display manager task, then set the display structure to inactive for this
 *  display.    Notify the Data Handler of a Displayer task halt by setting its
 *  clear flag. Call the check flt routine to delete this display from the
 *  flight information.
 */

    Dm_Address->process.disp_nbr--;
    pos_indx = Dm_Address->display[Disp_Num].pos_id_indx;
    Dm_Address->process.nbr_pos[pos_indx]--;
    if ( Dm_Address->process.nbr_pos[pos_indx] <= 0 ) {
        Dm_Address->process.pos_id[pos_indx][0] = 0;
    }
    if ( ( Dm_Address->process.disp_nbr > 0 ) && ( disp_init ) ) {
        Dm_Address->display[Disp_Num].dh_clear = YES;
        chk_flg ( &Dm_Address->display[Disp_Num].dh_clear, 5, 0 );
        Dm_Address->display[Disp_Num].disp_active = NO;
        chk_flt ( NO );
        Dm_Address->display[Disp_Num].disp_init = NO;
        Dm_Address->display[Disp_Num].halt = NO;
        Dm_Address->display[Disp_Num].disp_pid = -1;
    }

/*
 *  This is the last display manager in this workstation. Set the Data Handler
 *  halt flag.  Call the check flag routine to time the halt process. If the
 *  halt acknowledge flag is not set, the kill the process. Remove the Display
 *  Manager shm.
 */

    if ( Dm_Address->process.disp_nbr <= 0 ) {
        Dm_Address->process.dh_not_halted = YES;
        Dm_Address->process.disp_halt_nbr = Disp_Num;
        Dm_Address->process.dh_halt_ack = NO;
        error = chk_flg ( &Dm_Address->process.dh_halt_ack, 10, YES );
        if ( error )
            kill ( Dm_Address->process.dh_pid, SIGKILL );
    }

/*
 *  Turn off display of popups and advise of the Display Manager exiting.
 */

    tui_msg_control ( Msg_Popup_Flag = FALSE );
    EXmsg ( M_WHITE, "Display Manager %d exiting", Dm_Address->process.disp_nbr+1 );

/*
 *  Remove the main control panel window.
 */

    XtDestroyWidget ( Top );

/*
 *  If this is the last Display Manager, remove the shared memory segment.
 */

    if ( Dm_Address->process.disp_nbr <= 0 ) {
        shmdt ( Dm_Address );
        shmctl ( Dm_Id, IPC_RMID, 0 );
    }
```

```
/*
 *  Normal return.
 */

    D(printf("END cleanup\n"));
    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: clear.c
 *
 *  This function clears the display window, frees allocated memory for
 *  the given display, and resets world coordinates.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Richard Romeo - Ford Aerospace Corporation/Houston
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Ronnie Killough - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <X11/Xlib.h>
#include <constants.h>
#include <disp.h>
#include <DDdisp.h>
#include <DDplot.h>
#include <wex/EXmsg.h>


extern struct dm_shmemory   *Dm_Address;     /* ptr to DM shared memory      */
extern struct bg_recs       Bg_Rec;          /* ptr to background records    */
extern struct mtext_ent     *Mtext;          /* ptr to multi-level text recs */
extern struct fg_file_header    *Ffile;      /* ptr to fg ddf header         */
extern struct plot_ptrs     *Plot_info_ptr; /* ptr to plot records          */

extern short                Nbr_of_plots;   /* # of plot records            */


int clear(disp_num)

    short   disp_num;           /* number of display to clear               */

{
    struct graph_record *graph_ptr;     /* ptr thru bg graphics records     */
    struct rec_header   *bg_text_ptr;   /* ptr thru bg text records         */
    struct mtext_ent    *mtext_ptr;     /* ptr thru multitext items         */
    struct plot_ptrs    *plot_ptr;      /* ptr thru plot records            */
    struct lim_lines    *nline_ptr;     /* ptr thru nom/lim lines           */

    short               i;              /* loop counter                     */


    D(printf("START clear\n"));

/*
 *  Verify the display is active
 */

    if (Dm_Address->display[disp_num].disp_active == NO) {
        tui_msg(M_YELLOW, "Display %d not active", disp_num);
        return(0);
    }
```

```
/*
 *  Clear the window
 */

    XClearWindow(Dm_Address->xdisplay[disp_num], Dm_Address->window[disp_num]);

/*
 *  Destroy plot widgets for all plots.
 *  Deallocate memory allocated for plot records.
 *  Deallocate buffer memory for all previously active plots.
 */

    plot_ptr = Plot_info_ptr;

    for (i=0; i<Nbr_of_plots; i++) {
        plot_ptr = Plot_info_ptr + i;
        XtDestroyWidget(plot_ptr->draw_win);
        free((char *)plot_ptr->plt_decom);
        free((char *)plot_ptr->msids);
        free((char *)plot_ptr->axis);

        if (plot_ptr->header->nline_num > 0) {
            nline_ptr = plot_ptr->nline;

            for (i=0; i<plot_ptr->header->nline_num; i++) {
                free((char *)nline_ptr->plot_pts_ptr);
                nline_ptr++;
            }

            free((char *)plot_ptr->nline);
        }

        if (plot_ptr->header->lline_num > 0) {
            nline_ptr = plot_ptr->lline;

            for (i=0; i<plot_ptr->header->lline_num; i++) {
                free((char *)nline_ptr->plot_pts_ptr);
                nline_ptr++;
            }

            free((char *)plot_ptr->lline);
        }

        if (plot_ptr->prev_act_flg == YES)
            free((char *)plot_ptr->plot_data);

    }

/*
 *  Deallocate memory allocated for background
 *  graphics and text records.
 */

/* RLK 10/24/90 All have been deallocated except points arrays and vector text
                character arrays. */

    if (Bg_Rec.graph_num > 0) {
        graph_ptr = Bg_Rec.graph_rec;

        for (i=0; i < Bg_Rec.graph_num; i++) {
            free((char *)graph_ptr->graph_ptr);
            graph_ptr++;
        }
```

```
        free((char *)Bg_Rec.graph_rec);
    }

    if (Bg_Rec.char_num > 0) {
        bg_text_ptr = Bg_Rec.record;

        for (i = 0; i < Bg_Rec.char_num; i++) {
            free ((char *)bg_text_ptr->record_item);
            bg_text_ptr++;
        }

        free((char *)Bg_Rec.record);
    }

/*
 *  Deallocate memory allocated for
 *  foreground multi-level text records.
 */

    mtext_ptr = Mtext;

    for (i = 0; i < Ffile->Mltxt_Num; i++) {
        free ((char *)mtext_ptr->text_ptr);
        mtext_ptr++;
    }

/*
 *  Deallocate memory allocated for PBI labels,
 *  and ddd_msids, labels.
 */

/*
 *  Deallocate memory block allocated for
 *  foreground records.
 */

    free((char *)Ffile);

/*
 *  Restore the world coordinates to 0 and 100.
 */

    Dm_Address->display[disp_num].low_x = 0.0;
    Dm_Address->display[disp_num].low_y = 0.0;
    Dm_Address->display[disp_num].high_x = 100.0;
    Dm_Address->display[disp_num].high_y = 100.0;

    D(printf("END clear\n"));

    return (0);
}
```

```
/*****************************************************************************
 * MODULE NAME: clr_disp.c
 *
 *  This routine sets the clear display flag for the Data Handler task.
 *  The effective display is removed from the screen and all memory specific
 *  to that display freed via a call to exit_disp().
 *  In addition, the default PF keys are read back into the
 *  active PF keys file and the initialization screen is brought back up.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  K. Noonan       - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <signal.h>
#include <memory.h>
#include <constants.h>
#include <disp.h>
#include <pf_key.h>
#include <wex/EXmsg.h>


extern struct dm_shmemory   *Dm_Address;
extern struct pfkey_defs    Act_Pfkeys[PFKEY_COUNT],
                            Def_Pfkeys[PFKEY_COUNT],
                            Current_Com;

extern char                 **Msid_list_ddd,
                            **Msid_list_lim;

extern int                  Msid_num_ddd,
                            Msid_num_lim;

extern short                Pbi_Num,
                            Disp_Num;

int clr_disp ( )
{
    struct disp_info        *display;

    short                   error;

    D(printf("START clr_disp\n"));

/*
 *  If a display is currently active, then copy the default PF key definitions
 *  into the active structure.  Set the display clear flag and wait for a
 *  response.  If no response, advise and clear the flag anyway.
 */

    display = &Dm_Address->display[Disp_Num];

    if ( display->active_display == YES ) {
        memcpy ( (char  *)Act_Pfkeys, (char *)Def_Pfkeys,
```

```
                PFKEY_COUNT * sizeof ( struct pfkey_defs ) );

/*
 *  Call the exit_disp routine to remove the display from the screen
 *  and free all display-specific memory.
 */

        if (Dm_Address->display[Disp_Num].disp_init) {
            Dm_Address->display[Disp_Num].active_display = NO;
            Dm_Address->display[Disp_Num].halt = YES;

            exit_disp(Disp_Num);
        }

/*
 *  Set the Data Handler display clear flag and wait for a response.  If no
 *  response, advise and clear the flag.
 */

        display->dh_clear = YES;
        error = chk_flg ( &display->dh_clear, 20, 0 );
        if ( error ) {
            tui_msg ( M_YELLOW, "Error in Data Handler display clear for display %d",
                  Disp_Num );
        }
/*
 *  If we are clearing out an old display with PBIs, clear out the memory
 *  used by the PBIs.
 */

        if ( ( Current_Com.func_no == CLEAR_DISPLAY ) ||
             ( Current_Com.func_no == START_DISPLAY ) ||
             ( Current_Com.func_no == START_PDISPLAY ) ) {

            if ( Pbi_Num > 0 )
                pbi_free ( );

        }
/*
 *  Draw the initialization screen, if the clear command has been selected.
 *  Clear the display active flag.
 */

        if ( Current_Com.func_no == CLEAR_DISPLAY )
            display->active_display = NO;
    }

/*
 *  If a display is not active, then advise.
 */

    else {
        tui_msg ( M_WHITE, "No display has been initialized" );
    }

/*
 *  Free MSID lists and zero counts.
 */

    free ( (char *)Msid_list_lim );
    free ( (char *)Msid_list_ddd );

    Msid_num_lim = Msid_num_ddd = 0;
```

```
    D(printf("END clr_disp\n"));
    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: colorpal.c
 *
 *  The globals file contains only declaration statements of variables.
 *  There is no "executable code" in the file.  The purpose of this file is
 *  for an easy look up for all globals variables.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  C. Davis        - Ford Aerospace Corporation
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/

float           Colors[128][3] = {

    {0.0, 0.0, 0.0},     /* dark gray #116 */
    {0.1, 0.1, 0.1},
    {0.2, 0.2, 0.2},
    {0.25, 0.25, 0.25},
    {0.3, 0.3, 0.3},
    {0.4, 0.4, 0.4},
    {0.5, 0.5, 0.5},
    {0.55, 0.55, 0.55},

    {0.55, 0.55, 0.55}, /* light gray #115 */
    {0.65, 0.65, 0.65},
    {0.7, 0.7, 0.7},
    {0.75, 0.75, 0.75},
    {0.8, 0.8, 0.8},
    {0.9, 0.9, 0.9},
    {0.95, 0.95, 0.95},
    {1.0, 1.0, 1.0},

    {0.7, 0.1, 0.7},     /* light violet #114 */
    {0.8, 0.2, 0.8},
    {0.9, 0.4, 0.9},
    {0.9, 0.5, 0.9},
    {1.0, 0.6, 1.0},
    {1.0, 0.7, 1.0},
    {1.0, 0.8, 1.0},
    {1.0, 0.9, 1.0},

    {0.3, 0.1, 0.3},     /* dark violet #113 */
    {0.4, 0.1, 0.4},
    {0.5, 0.1, 0.5},
    {0.6, 0.2, 0.6},
    {0.6, 0.3, 0.6},
    {0.7, 0.4, 0.6},
    {0.8, 0.4, 0.7},
    {0.9, 0.4, 0.9},

    {1.0, 0.0, 0.0},     /* light red #112 */
    {1.0, 0.3, 0.3},
    {1.0, 0.4, 0.4},
    {1.0, 0.5, 0.5},
    {1.0, 0.6, 0.6},
```

```
{1.0, 0.7, 0.7},
{1.0, 0.8, 0.8},
{1.0, 0.9, 0.9},

{0.2, 0.0, 0.0},    /* dark red #111 */
{0.3, 0.0, 0.0},
{0.5, 0.0, 0.0},
{0.6, 0.0, 0.0},
{0.7, 0.0, 0.0},
{0.8, 0.0, 0.0},
{0.9, 0.0, 0.0},
{1.0, 0.0, 0.1},

{0.7, 0.2, 0.1},    /* orange  #110 */
{0.8, 0.2, 0.1},
{0.9, 0.2, 0.1},
{0.9, 0.3, 0.1},
{1.0, 0.4, 0.1},
{1.0, 0.5, 0.1},
{1.0, 0.6, 0.1},
{1.0, 0.7, 0.1},

{0.3, 0.1, 0.0},    /* brown #109 */
{0.4, 0.2, 0.0},
{0.5, 0.3, 0.0},
{0.6, 0.4, 0.0},
{0.7, 0.5, 0.0},
{0.8, 0.6, 0.0},
{0.9, 0.7, 0.0},
{1.0, 0.8, 0.0},

{0.25, 0.25, 0.1},  /* yellow green #108 */
{0.35, 0.35, 0.1},
{0.45, 0.45, 0.1},
{0.55, 0.55, 0.0},
{0.65, 0.65, 0.1},
{0.75, 0.75, 0.1},
{0.85, 0.85, 0.1},
{0.95, 0.95, 0.1},

{0.6, 0.6, 0.0},    /* yellow #107 */
{0.7, 0.7, 0.0},
{0.8, 0.8, 0.1},
{0.9, 0.9, 0.0},
{0.95, 0.95, 0.0},
{1.0, 1.0, 0.0},
{1.0, 1.0, 0.5},
{1.0, 1.0, 0.7},

{0.0, 0.5, 0.5},    /* grayish green #106 */
{0.1, 0.6, 0.5},
{0.3, 0.7, 0.55},
{0.2, 0.8, 0.5},
{0.5, 0.9, 0.5},
{0.5, 1.0, 0.5},
{0.7, 1.0, 0.7},
{0.8, 1.0, 0.8},

{0.2, 0.2, 0.0},    /* brownish green #105 */
{0.3, 0.3, 0.1},
{0.3, 0.5, 0.1},
{0.4, 0.6, 0.1},
{0.4, 0.75, 0.2},
{0.5, 0.8, 0.3},
```

```
        {0.6, 0.8, 0.4},
        {0.7, 0.9, 0.4},

        {0.0, 0.3, 0.0},    /* bright to dark green #104 */
        {0.0, 0.4, 0.0},
        {0.0, 0.5, 0.0},
        {0.0, 0.6, 0.0},
        {0.0, 0.7, 0.0},
        {0.0, 0.8, 0.0},
        {0.0, 0.9, 0.0},
        {0.0, 1.0, 0.0},

        {0.0, 0.2, 0.5},    /* blue-green #103 */
        {0.0, 0.3, 0.5},
        {0.0, 0.4, 0.5},
        {0.0, 0.5, 0.5},
        {0.0, 0.6, 0.5},
        {0.0, 0.7, 0.5},
        {0.0, 0.8, 0.5},
        {0.0, 0.9, 0.5},

        {0.3, 0.2, 0.8},    /* aqua to medium blue  #102 */
        {0.4, 0.3, 0.9},
        {0.5, 0.4, 1.0},
        {0.5, 0.5, 1.0},
        {0.5, 0.7, 1.0},
        {0.5, 0.8, 1.0},
        {0.4, 0.9, 1.0},
        {0.0, 1.0, 1.0},

        {0.0, 0.1, 0.2},    /* dark blue #101 */
        {0.1, 0.2, 0.4},
        {0.1, 0.2, 0.5},
        {0.0, 0.1, 0.7},
        {0.0, 0.1, 0.8},
        {0.2, 0.3, 1.0},
        {0.3, 0.4, 1.0},
        {0.4, 0.5, 1.0},
};
```

```
/*****************************************************************************
 * MODULE NAME: colors.c
 *
 *  This routine stores a color pallette composed of 128 different colors into
 *  memory.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  C. Davis        - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <X11/Xlib.h>
#include <X11/Intrinsic.h>
#include <constants.h>
#include <wex/EXmsg.h>


extern Widget   Top;
extern Colormap Main_cmap;
extern short    Pixels[128];
extern float    Colors[128][3];


int colors ( )
{
    XColor          color;

    Display         *display;

    int             i;

    D(printf("START colors\n"));
/*
 *  Save the display and set color flags.
 */

    display     = XtDisplay ( Top );
    color.flags = DoRed | DoGreen | DoBlue;

/*
 *  Store all the colors into the display manager color map. Note that each
 *  color in the (Pixels) array is offset by the number of colors used for
 *  Motif.
 */

    for ( i = 0; i < 128; i++ ) {
        color.red   = (int)( Colors[i][0] * 65535 );
        color.green = (int)( Colors[i][1] * 65535 );
        color.blue  = (int)( Colors[i][2] * 65535 );
        Pixels[i] = color.pixel = i + NUM_MOTIF_COLORS;

        XStoreColor ( display, Main_cmap, &color );
    }
```

```
    /*
     *  Normal return.
     */

        D(printf("END colors\n"));
        return ( 0 );
    }
```

```
/*****************************************************************************
 * MODULE NAME: command.c
 *
 *  This function is the main command processor. It accepts commands from menus
 *  and function keys and calls the correct function.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  C. Davis        - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <X11/Intrinsic.h>
#include <constants.h>
#include <disp.h>
#include <pf_key.h>
#include <wex/EXmsg.h>


extern struct dm_shmemory *Dm_Address;
extern struct pfkey_defs Current_Com;

extern short    Dm_Halt,                /* task halt flag                    */
                Disp_Num,               /* Display Manager number            */
                Pbi_Disable,            /* PBI enable/disable flag           */
                Flt_Selected,           /* Yes if flt info has been input    */
                Msg_Popup_Flag,
                Pbi_Hot_Ndx,            /* Pbi Hot index of the Pbi currently hot*/
                Pbi_Num;                /* Number of Pbi's defined for this disp */

extern int      Msid_num_ddd,
                Msid_num_lim;

extern float    Zoom_factor;            /* current zoom factor               */

extern Widget   Top,
                Pb_Alarm,
                Pb_Pbi,
                Pb_Log,
                Pb_Log_A,
                Pb_Msg,
                Pb_Pf;


int command ( menu_flag )

    int     menu_flag;                  /* YES if called from the menu; NO if
                                         * called from a function key.
                                         */
{
    short           indx;               /* position Id index                 */

    D(printf("START command\n"));
/*
 *  If selected command requires a display to be active, but no display is present,
```

```
 *  generate an error.
 */

    switch ( Current_Com.func_no ) {

    case CLEAR_DISPLAY  :
    case FREEZE_DISPLAY :
    case RESTART_DISPLAY:
    case HIST_TAB       :
    case LIM_LIST       :
    case LIM_MENU       :
    case CHG_LIM        :
    case DDD_UNLATCH    :
    case DDD_UNL_ALL    :
    case PLOT_LIST      :
    case PLOT_OVRLAY    :
    case SAVE_OVRLAY    :
    case ZOOM_DIS       :
        if ( Dm_Address->display[Disp_Num].disp_init == NO ) {
            tui_msg ( M_YELLOW, "Command rejected - No display initialized" );
            return ( 0 );
        }
    }

/*
 *  If selected command requires the flight and data type to be specified, but they
 *  have not been specified, generate an error.
 */

    switch ( Current_Com.func_no ) {

    case START_DISPLAY  :
    case START_PDISPLAY :
    case GDR_CHG        :
    case HIST_TAB       :
    case LIM_LIST       :
    case LIM_MENU       :
    case CHG_LIM        :
    case LIM_GRP        :
    case LIM_GRP_OFF    :
    case PLOT_LIST      :
    case PLOT           :
    case PLOT_OFF       :
    case PLOT_OVRLAY    :
    case SAVE_OVRLAY    :
    case PLOT_UNV       :
        if ( Flt_Selected == NO ) {
            tui_msg ( M_YELLOW, "Command rejected - Flight ID/data type not input" );
            return ( 0 );
        }
    }

/*
 *  All required data has been specified, so begin processing of the different
 *  commands.
 */

    switch ( Current_Com.func_no ) {

/*
 *  Process setting of the popup display flag.
 */

    case MSG_ON:
```

```
    case MSG_OFF:
        if ( menu_flag == ON )
            Msg_Popup_Flag = ( Msg_Popup_Flag      == YES    ) ? NO  : YES;
        else
            Msg_Popup_Flag = ( Current_Com.func_no == MSG_ON ) ? YES : NO;

        if ( Msg_Popup_Flag == YES )
            set_label ( Pb_Msg, "Disable Messages" );
        else
            set_label ( Pb_Msg, "Enable Messages" );

        tui_msg_control ( Msg_Popup_Flag );

        break;

/*
 *  Read and set the flight and datatype.
 */

    case SET_FLIGHT:
        flt_data ( );
        break;

/*
 *  Dump screen contents.
 */

    case SCRN_DUMP:
        tui_start_wait ( );
        system("/usr/local/Xdump/400/xwd.400 | /usr/local/Xdump/400/xpr.400 | lp -ograph"
;
        tui_stop_wait ( );
        break;

/*
 *  Display/Allow edit of colors.
 */

    case EDIT_COLORS:
        edit_colors ( );
        break;

/*
 *  Terminate the program.
 */

    case HALT_DISPLAY:
        Dm_Halt = YES;
        break;

/*
 *  Get the display name. If called from a menu, allow selection of the name from a
 *  list of files; otherwise, allow direct entry of a name via a prompt. If a valid
 *  name is entered, bring up the new display.
 */

   case START_DISPLAY:
       if ( menu_flag )
           sel_disp ( );
       else
           get_disp ( );
       if ( Current_Com.func_no == START_PDISPLAY )
           new_disp ( );
       break;
```

```
/*
 *  If a display name has been specified and the flight information has been
 *  selected, then call the new display routine.  If no error, then clear the
 *  check screen variable, otherwise, if a display is not active, then redraw
 *  the initialization screen.
 */

    case START_PDISPLAY:
        if ( Current_Com.disp_name[0] != '\0' )
            new_disp ( );
        else
            tui_msg ( M_YELLOW, "Command rejected - No display name specified" );
        break;

/*
 *  Remove the current display.
 */

    case CLEAR_DISPLAY:
        clr_disp ( );
        break;

/*
 *  Based on the current state, FREEZE or PAUSE a display.
 */

    case FREEZE_DISPLAY:
    case RESTART_DISPLAY:
        Dm_Address->display[Disp_Num].disp_pause =
            ( Dm_Address->display[Disp_Num].disp_pause ) ? NO : YES;
        if ( Dm_Address->display[Disp_Num].disp_pause )
            set_label ( Pb_Pf, "Restart Display" );
        else {
            set_label ( Pb_Pf, "Freeze Display"  );
            set_timer ( Disp_Num );
        }
        break;

/*
 *  Store the new display update rate.
 */

    case UPD_RATE:
        if ( menu_flag )
            upd_rate ( );
        else
            Dm_Address->display[Disp_Num].update_rate = Current_Com.rate;
        break;

/*
 *  Unlatch DDD's. Note that the DDD_UNL_ALL command is only generated via the menus.
 */

    case DDD_UNLATCH:
    case DDD_UNL_ALL:
        if ( Msid_num_ddd == 0 )
            tui_msg ( M_YELLOW, "No DDD's in current display" );
        else {
            if ( Current_Com.func_no == DDD_UNL_ALL ) {
                Dm_Address->display[Disp_Num].action  = ALL;
                Dm_Address->display[Disp_Num].unlatch = YES;
                unlatch ( );
            } else
```

```
                if ( menu_flag == YES )
                    ddd_msid ( NOT_READY );
                else
                    ddd_msid ( READY     );
            }
            break;

    /*
     *  Change GDR retrieval information.
     */

        case GDR_CHG:
            chg_gdr ( );
            break;

    /*
     *  Toggle the position Id alarm flag.  Get the index to the current position
     *  Id flag.  Store the action in the alarm flag.  Advise of the current
     *  status.
     */

        case POS_ALARM:
        case POS_ALARM_OFF:
            indx = Dm_Address->display[Disp_Num].pos_id_indx;
            if ( menu_flag == ON )
                Dm_Address->process.alarm[indx] =
                    ( Dm_Address->process.alarm[indx] == ON ) ? OFF: ON;
            else
                Dm_Address->process.alarm[indx] =
                    ( Current_Com.func_no == POS_ALARM ) ? ON : OFF;

            if ( Dm_Address->process.alarm[indx] == ON ) {
                set_label ( Pb_Alarm, "Disable Alarms" );
                tui_msg ( M_WHITE, "Position Id %s alarm enabled",
                      Dm_Address->process.pos_id[indx] );
            } else {
                set_label ( Pb_Alarm, "Enable Alarms" );
                tui_msg ( M_WHITE, "Position Id %s alarm disabled",
                      Dm_Address->process.pos_id[indx] );
            }
            break;

    /*
     *  Enable/Disable PBI's.
     */

        case PBI_ENABLE:
        case PBI_DISABLE:
            if ( menu_flag == YES )
                Pbi_Disable = ( Pbi_Disable          == ENABLED    ) ? DISABLED : ENABLED;
            else
                Pbi_Disable = ( Current_Com.func_no == PBI_ENABLE ) ? ENABLED : DISABLED;

            if ( Pbi_Disable == ENABLED ) {
                set_label ( Pb_Pbi, "Disable PBI's" );
                tui_msg ( M_WHITE, "PBI Input enabled"  );
            } else {
                set_label ( Pb_Pbi, "Enable PBI's"  );
                tui_msg ( M_WHITE, "PBI Input disabled" );
            }
            break;

    /*
     *  Process Display Log enable commands.
```

```
*/

    case LOGDISABLE_DISPLAY:
    case LOGENABLE_DISPLAY:
        if ( menu_flag == YES )
            Dm_Address->display[Disp_Num].log_enable =
                ( Dm_Address->display[Disp_Num].log_enable == YES ) ? NO : YES;
        else
            Dm_Address->display[Disp_Num].log_enable =
                ( Current_Com.func_no == LOGDISABLE_DISPLAY       ) ? NO : YES;

        if ( Dm_Address->display[Disp_Num].log_enable == NO ) {
            set_label ( Pb_Log, "Enable Logging" );
            tui_msg ( M_WHITE, "Display Logging disabled" );
        } else {
            set_label ( Pb_Log, "Disable Logging" );
            tui_msg ( M_WHITE, "Display Logging enabled" );
        }
        break;

/*
 *  Process Enable All  logging commands.
 */

    case LOGDISABLE_ALL:
    case LOGENABLE_ALL:
        if ( menu_flag == YES )
            Dm_Address->process.log_enable =
                ( Dm_Address->process.log_enable == YES ) ? NO : YES;
        else
            Dm_Address->process.log_enable =
                ( Current_Com.func_no == LOGDISABLE_ALL ) ? NO : YES;

        if ( Dm_Address->process.log_enable == NO ) {
            set_label ( Pb_Log_A, "Enable All Logging" );
            tui_msg ( M_WHITE, "All  Logging disabled" );
        } else {
            set_label ( Pb_Log_A, "Disable All Logging" );
            tui_msg ( M_WHITE, "All  Logging enabled" );
        }
        break;

/*
 *  List limit group files and allow user to turn one on or off.
 */

    case LIM_LIST:
        if ( Msid_num_lim == 0 )
            tui_msg ( M_YELLOW, "No MSID's appropriate for limits in current display" );
        else
            list_files ( YES, FALSE );
        break;

/*
 *  Enter MSID and limits to change and notify the Data Handler of the new limits.
 */

    case LIM_MENU:
    case CHG_LIM:
        if ( Msid_num_lim == 0 )
            tui_msg ( M_YELLOW, "No MSID's appropriate for limits in current display" );
        else
            chg_lim ( );
        break;
```

```
/*
 *  Turn on or off a limit group. These commands will only come from function keys and
 *  PBI's.
 */

    case LIM_GRP:
    case LIM_GRP_OFF:
        if ( Msid_num_lim == 0 )
            tui_msg ( M_YELLOW, "No MSID's appropriate for limits in current display" );
        else
            lim_grp ( );
        break;

/*
 *  Process the list plot command.
 */

    case PLOT_LIST:
        list_files ( NO, FALSE );
        break;

/*
 *  Process the plot stop and start commands and the plot overlay command. These
 *  commands will only come from function keys and PBI's.
 */

    case PLOT:
    case PLOT_OFF:
        get_plot ( );
        break;

/*
 *  Process the display and save overlay commands.
 */

    case PLOT_OVRLAY:
    case SAVE_OVRLAY:
        if ( menu_flag )
            plot_ovl ( NOT_READY );
        else
            plot_ovl ( READY     );
        break;

/*
 *  Process the universal plot definition command.
 */

    case PLOT_UNV:
        unv_plot ( );
        break;

/*
 *  Zoom the display/Reset display to original view
 */

    case ZOOM_DIS:
        Dm_Address->display[Disp_Num].dd_zoom = ZOOM_DIS;
        zoom ( Disp_Num );
        break;

/*
 *  Unzoom (reset display to original view).
 */
```

```
    case ZOOM_RES:
        Dm_Address->display[Disp_Num].dd_zoom = ZOOM_RES;
        zoom ( Disp_Num );
        break;

/*
 *  Change the zoom factor
 */

    case ZOOM_FAC:
        if ( menu_flag )
            chg_zoom ( );
        else
            Dm_Address->display[Disp_Num].dd_zfact = Zoom_factor = Current_Com.factor;

        break;

/*
 *  Do history table.
 */

    case HIST_TAB:
        if ( menu_flag )
            hist_tab ( );
        else {
            strcpy ( Dm_Address->display[Disp_Num].display_name, Current_Com.disp_name );
            strcpy ( Dm_Address->display[Disp_Num].plot_overlay, Current_Com.ovr_name  );
            Dm_Address->display[Disp_Num].dh_htab = YES;
        }
        break;

/*
 *  Call the gdr next routine to process the GDR get next command. This command is not
 *  available from any menu.
 */

    case GDR_GETNEXT:
        gdr_next ( );
        break;

/*
 *  Execute a UNIX command. This command is not available from any menu.
 */

    case UNIX_COMMAND:
        system ( Current_Com.mesg_ptr );
        break;

/*
 *  Send a message to another process. This command is not available from any menu.
 */

    case EXMSG_SEND:
        ex_msgsnd ( &Current_Com );
        break;

    default:
        break;

    }

/*
 *  Normal return.
```

```
 */

    D(printf("END command\n"));
    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: date_chek.c
 *
 *  This function outputs an error if the version is out of date.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  C. Davis        - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <constants.h>
#include <sys/types.h>
#include <sys/timeb.h>
#include <wex/EXmsg.h>


#define SEPT_1_1990 900000000


int date_chek ( )
{
    struct timeb    current_time;

    ftime ( &current_time );
    if ( current_time.time > SEPT_1_1990 ) {
        tui_msg ( M_YELLOW, "This Program Version is out of date after September 1, 1990"
);
        tui_msg ( M_YELLOW, "Get a new version" );
        return ( -1 );
    } else {
        return ( 0 );
    }
}
```

```
/*****************************************************************************
 * MODULE NAME: dcm_ent.c
 *
 *  This routine locates the proper entry point into the decom and
 *  data buffers and calls extract to extract the data.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Richard Romeo   -   Ford Aerospace Corporation
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <memory.h>
#include <sys/types.h>
#include <X11/Xlib.h>
#include <sys/timeb.h>
#include <wex/EXmsg.h>
#include <constants.h>
#include <disp.h>
#include <DDfg_graph.h>
#include <DDdisp.h>


extern union p_data     Data;                   /* Local ptr for union structure.      */
extern int              Offset;                 /* Offset into the old data array.     */

extern short            No_Change;              /* No change of data flag.             */

extern long             Status_Color;           /* Status color for dynamic line.      */

extern unsigned char    Graph_New_Data[60000],  /* New Data Array                      */
                        Graph_Old_Data[60000];  /* Old Data Array                      */


int dcm_ent ( decom_entry, data_buf, msid_info, first_pass )

    struct shm_decom    *decom_entry;           /* Local ptr to decom buffer.          */

    register char       *data_buf;              /* Ptr to data buffer.                 */

    struct msid_ent     *msid_info;             /* Msid entry table local ptr.         */

    short               first_pass;             /* First pass buffer flag.             */
{
    long                status;                 /* Local status variable for msid.     */

    int                 sample_size,            /* Sample size of one sample.          */
                        retval,                 /* Return value of memcmp.             */
                        skip_amt;               /* Number of bytes to skip.            *

    short               dcm_error_flg;          /* Local decom error flag.             */

    unsigned char       *start_of_sample;       /* Start of sample in decom buffer.    */
```

```
/*
 *  Check for decom error in local decom buffer and calculate number of bytes that have
 *  to be skipped per sample
 */

    dcm_error_flg = NO;

    if ( decom_entry->error == NULL ) {

        if ( decom_entry->num_samps > 0 ) {
            sample_size = decom_entry->sample_size;

            if ( msid_info->Sample == -1 )
                skip_amt = ( decom_entry->num_samps - 1 ) * sample_size;
            else {

                if ( msid_info->Sample > decom_entry->num_samps )
                    dcm_error_flg = YES;
                else
                    skip_amt = ( msid_info->Sample - 1 ) * sample_size;
            }

            if ( dcm_error_flg == NO ) {
                start_of_sample = ( unsigned char * )
                     ( data_buf + decom_entry->offset + skip_amt );

/*
 *              Check data buffer for status of bit 11, set change bit and
 *              call extract.
 */

                if ( first_pass == 0 ) {
                    memcpy ( (char *)&Graph_New_Data[Offset],
                             (char *)start_of_sample, sample_size );
                    Offset += sample_size;
                } else {
                    retval = memcmp ( (char *)&Graph_New_Data[Offset],
                                      (char *)&Graph_Old_Data[Offset], sample_size );

                    if ( retval != 0 ) {
                        memcpy ( (char *)&Graph_Old_Data[Offset],
                                 (char *)&Graph_New_Data[Offset], sample_size );
                        status = extract ( &Graph_New_Data[Offset], decom_entry );
                        Status_Color = stat_col ( status, msid_info );
                        Offset += sample_size;
                        No_Change = NO;
                    }
                }

            }
        }
    } else
        dcm_error_flg = YES;

    return ( dcm_error_flg );
}
```

```
/*****************************************************************************
 * MODULE NAME: ddd.c
 *
 *  This routine will determine whether a primitive is to be
 *  latched by extracting the value associated with the ddd msid and comparing
 *  the value to the latched value given in the display definition table.  If
 *  the value matches the latched value, then the msid is latched, a latched
 *  flag is set, and the latched color is stored.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  R. Romeo        - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <X11/Xlib.h>
#include <constants.h>
#include <disp.h>
#include <DDdisp.h>


extern union p_data     Data;           /* Pointer for data union structure.
extern struct data_info *Dh_Address;    /* Displayer shared memory.          */
extern struct msid_ent  *Msid;          /* Msid entry table pointer.         */
extern struct ddd_ent   *Ddd;           /* Ddd entry table pointer.          */
extern short            Disp_Num;       /* Display number.                   */


long ddd ( dcm_buf, data_buf, indx, first_pass )

    register struct shm_decom   *dcm_buf;   /* ptr to dcm buffer             */
    register char               *data_buf;  /* ptr to data buffer            */
    int                         indx;       /* index from primitive to ddd   */
    short                       first_pass; /* first pass buffer flag        */
{
    struct msid_ent             *msid_info; /* msid entry table pointer      */
    struct shm_decom            *decom_entry;/* local ptr to decom buffer    */
    struct ddd_ent              *ddd_ptr;   /* Ddd entry table pointer       */

    int                         i,          /* Ctr flag.                     */
                                error;      /* Error flag.                   */
    long                        new_color = INVALID;
                                /* determined color of the primitive         */
    short                       latchable = YES,
                                /* YES, primitive can be latched             */
                                *loc_ddd_ptr;

/*
 *  Initialize ddd parameters.
 */

    ddd_ptr = Ddd + indx - 1;
    loc_ddd_ptr = ddd_ptr->ddd_app_ptr;
    if ( ( !ddd_ptr->zero_locked ) && ( !ddd_ptr->one_locked ) )
```

```
        latchable = NO;

    for ( i = 0; i < ddd_ptr->ddd_msids; i++ ) {
        msid_info = Msid + *loc_ddd_ptr - 1;

/*
 *  If the msid is already latched, then get the ddd latch color and set the latched
 *  flag.
 */

        if ( ( msid_info->ddd0_latch ) && ( ddd_ptr->zero_locked ) ) {
            new_color = ddd_ptr->zero_val_cor;
            break;
        } else if ( ( msid_info->ddd1_latch ) && ( ddd_ptr->one_locked ) ) {
            new_color = ddd_ptr->one_val_cor;
            break;
        } else {
            if ( Dh_Address->msid_index[Disp_Num][*loc_ddd_ptr - 1] >= 0 ) {
                decom_entry = dcm_buf +
                    Dh_Address->msid_index[Disp_Num][*loc_ddd_ptr - 1];

/*
 *              Call decom buffer entry routine to extract the value of the status
 *              word in order to get the binary discrete for the ddd logic.
 */

                error = dcm_ent ( decom_entry, data_buf, msid_info, first_pass );
                if ( error == NULL ) {

/*
 *                  The locked value is a 0.
 */

                    if ( Data.sldata[0] == 0 ) {
                        new_color = ddd_ptr->zero_val_cor;
                        if ( ddd_ptr->zero_locked ) {
                            msid_info->ddd0_latch = YES;
                            break;
                        }

/*
 *                  The locked value is a 1.
 */

                    } else {
                        new_color = ddd_ptr->one_val_cor;
                        if ( ddd_ptr->one_locked ) {
                            msid_info->ddd1_latch = YES;
                            break;
                        } else if ( latchable == NO )
                            break;
                    }
                } else if ( new_color == INVALID )
                    new_color = msid_info->Dead_Color;

            } else if ( new_color == INVALID )
                new_color = msid_info->Dead_Color;
        }
        loc_ddd_ptr++;
    }
    return ( new_color );
}
```

```
/*****************************************************************************
 * MODULE NAME: ddd_msid.c
 *
 *  This function allows the user to unlatch as specific DDD MSID.
 *
 *
 * INTERNAL FUNCTIONS:
 *
 *      o   ddd_menu     -   This function presents the menu which allows the
 *                           DDD to be selected.
 *
 *      o   cb_ddd       -   This function handles all callbacks generated by
 *                           the menu.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  C. Davis         - Ford Aerospace Corporation
 *
 *
 *  MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <X11/Shell.h>
#include <Xm/Xm.h>
#include <Xm/mwm.h>
#include <Xm/Text.h>
#include <constants.h>
#include <disp.h>
#include <DDdisp.h>
#include <pf_key.h>
#include <user_inter.h>
#include <wex/EXmsg.h>


static Widget               t_msid;
static int                  flag;

extern Widget               Top;
extern struct msid_ent      *Msid;
extern struct pfkey_defs    Current_Com;     /* current commands definition */
extern struct dm_shmemory   *Dm_Address;     /* display manager shm         */
extern short                Disp_Num;        /* display manager number      */
extern char                 **Msid_list_ddd,
                            *Src_list[];
extern int                  Msid_num_ddd;



int ddd_msid ( ready )

    int             ready;                  /* If TRUE, indicates that display of a
                                               * menu is not required.
                                               */
{
    D(printf("START ddd_msid\n"));
```

```
/*
 *  If called from menu, display the popup.
 */

    if ( ready == NO )
        ddd_menu ( );

/*
 *  If called from a function key or if values were entered via the menu, store the MSID
 *  and source values in shared memory.
 */

    if ( ready == YES || flag == YES ) {
        strcpy ( Dm_Address->display[Disp_Num].msid_name, Current_Com.limit_change.msid );
        strcpy ( Dm_Address->display[Disp_Num].src,       Current_Com.limit_change.src  );
        tui_msg ( M_BLUE, "msid <%s> src <%s>", Current_Com.limit_change.msid,
                                                Current_Com.limit_change.src );
        Dm_Address->display[Disp_Num].action  = UNLATCH_MSID;
        Dm_Address->display[Disp_Num].unlatch = YES;
        unlatch ( );
    }

    D(printf("END ddd_msid\n"));
    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: ddd_menu
 *
 *  This function presents the menu which allows the DDD to be selected.
 *****************************************************************************/

static int ddd_menu ( )
{
    register int    i;

    Arg             args[10];

    Widget          shell, form, f_msid, f_cmd;

    XtCallbackProc  cb_ddd();

    XEvent          event;

    D(printf("START ddd_menu\n"));
/*
 *  Create the shell widget. Note that setting the args in the create
 *  widget call does not seem to work, so I set them afterward.
 */

    i = 0;
    shell = tui_create_trans_shell ( "Change DDD MSID", args, i );

    i = 0;
    XtSetArg ( args[i], XmNmwmInputMode, MWM_INPUT_APPLICATION_MODAL ); i++;
    XtSetValues ( shell, args, i );

/*
 *  Create the main form.
 */

    i = 0;
    form   = tui_create_form ( shell, "form",   TRUE,  args, i );
    f_msid = tui_create_form ( form,  "f_msid", FALSE, args, i );
    f_cmd  = tui_create_form ( form,  "f_cmd",  FALSE, args, i );

/*
 *  Create all widgets.
 */

    i = 0;
    t_msid = tui_create_sel ( f_msid, "t_msid", Msid_list_ddd, Msid_num_ddd, "MSID's",
                              args, i );

    i = 0;
    XtManageChild ( XmCreateSeparator ( form, "sep0", args, i ) );

    i = 0;
    tui_create_pushbutton ( f_cmd, "Cancel", cb_ddd, (caddr_t)0, args, i );
    tui_create_pushbutton ( f_cmd, "OK",     cb_ddd, (caddr_t)1, args, i );
    tui_create_pushbutton ( f_cmd, "Help",   cb_ddd, (caddr_t)2, args, i );

/*
 *  Put all inputs in a tab group.
 */

    XmAddTabGroup ( t_msid );

/*
```

```
     *  Realize and popup the shell.
     */

        XtRealizeWidget ( shell );
        XtPopup ( shell, None );
        set_cmap ( shell );

    /*
     *  Wait until the user finishes with the popup.
     */

        flag = -1;
        while ( flag == -1 ) {
            XtNextEvent     ( &event );
            XtDispatchEvent ( &event );
        }

        XtDestroyWidget ( shell );

    /*
     *  Return the value selected by the user (0 is for not verified, 1 is for
     *  verified.
     */

        D(printf("END ddd_menu\n"));
        return ( flag );
    }
```

```
/*****************************************************************************
 * MODULE NAME: cb_ddd
 *
 *  This function handles all callbacks generated by the menu.
 *****************************************************************************/

/* ARGSUSED */
static XtCallbackProc cb_ddd ( w, closure, calldata )

    Widget      w;              /* Set to widget which in which callback originated.    */

    caddr_t     closure,        /* Indicates selected command.                          */
                *calldata;      /* Widget-specific information.                         */
{
    register int    i;

    XtCallbackProc  cb_help();

    char            *ptr,
                    new_src [4],
                    real_src[4];

    D(printf("START cb_ddd\n"));
/*
 *  Process OK button. First extract and verify the MSID.
 */

    if ( (int)closure == 1 ) {
        strcpy ( Current_Com.limit_change.msid, ptr = XmTextGetString ( t_msid ) );
        free ( ptr );

/*
 *      Verify that the MSID exists and set index into list of MSID's. If MSID is
 *      invalid, return.
 */

        if ( ( i = val_msid ( Msid_list_ddd, Msid_num_ddd,
                              Current_Com.limit_change.msid ) ) == 0 )
            return;

/*
 *      Save the corresponding source for the selected MSID.
 */

        strcpy ( new_src, (Msid+i)->Data_Src );
        if ( val_src ( new_src, real_src ) == 0 ) {
            tui_msg ( M_YELLOW, "Invalid data source" );
            return;
        }

        strcpy ( Current_Com.limit_change.src, new_src );
        flag = (int)closure;

/*
 *  Process CANCEL button.
 */

    } else if ( (int)closure == 0 ) {
        flag = (int)closure;

/*
 *  If help button was selected, display appropriate help text.
 */
```

```
    } else if ( (int)closure == 2 )
        cb_help ( (Widget)0, (caddr_t)9, (caddr_t)0 );

    D(printf("END cb_ddd\n"));
    return;
}
```

```
/*****************************************************************************
 * MODULE NAME: dec_val.c
 *
 *  This function validates a decimal value.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  K. Noonan       - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <ctype.h>
#include <constants.h>
#include <wex/EXmsg.h>


int dec_val ( char_str )

    char            *char_str;   /* integer or decimal character string */
{
    short            i,          /* loop counter */
                     decimal,    /* counter for number of decimal points */
                     valid;      /* set to YES if string is valid        */

    int              length;     /* length of character string           */

    D(printf("START dec_val\n"));
/*
 * Get the length of the character string
 */

    i = 0;
    length = strlen ( char_str );
    valid = YES;
    decimal = NO;

/*
 *  First character may be "+" or "-".  If not, then check for a decimal or
 *  a digit.
 */

    if ( ( * ( char_str + i ) == '+' ) || ( * ( char_str + i ) == '-' ) ) {
        i++;
    } else {
        if ( isdigit ( * ( char_str + i ) ) != 0 ) {
            i++;
        } else {
            if ( * ( char_str + i ) == '.' ) {
                decimal = YES;
                i++;
            } else {
                valid = NO;
            }
        }
    }
```

```
    /*
     *  Loop through until all characters are validated or a character is found
     *  invalid.
     */

        while ( i < length && valid == YES ) {
            if ( isdigit ( * ( char_str + i ) ) != 0 ) {
                i++;
            } else {
                if ( ( * ( char_str + i ) == '.' ) && ( decimal == NO ) ) {
                    decimal = YES;
                    i++;
                } else {
                    valid = NO;
                }
            }
        }

        D(printf("END dec_val\n"));
        return ( valid );
    }
```

```
/* LINTLIBRARY */

#include <stdio.h>
#include <X11/Xlib.h>
#include <constants.h>
#include <disp.h>
#include <DDdisp.h>
#include <DDplot.h>
#include <pf_key.h>
#include <wex/FCpbi.h>

int DDpbi_updt ( disp_num )
    short   disp_num;
{ return ( 0 ); }

int chg_gdr ( )
{ return ( 0 ); }

int gdr_menu ( )
{ return ( 0 ); }

int chg_lim ( )
{ return ( 0 ); }

int chgl_menu ( )
{ return ( 0 ); }

int chg_zoom ( )
{ return ( 0 ); }

int chg_zoom_menu ( )
{ return ( 0 ); }

int chk_flg ( flag, wait_count, value )
    short   *flag,
            wait_count,
            value;
{ return ( 0 ); }

int chk_flt ( add )
    short   add;
{ return ( 0 ); }

int chk_res ( access, pos_id )
    short   access;
    char    *pos_id;
{ return ( 0 ); }

int cleanup ( )
{ return ( 0 ); }

int clear ( disp_num )
    short   disp_num;
{ return ( 0 ); }

int clr_disp ( )
{ return ( 0 ); }

int colors ( )
{ return ( 0 ); }

int command ( menu_flag )
    int     menu_flag;
{ return ( 0 ); }
```

```
int date_chek ( )
{ return ( 0 ); }

int dcm_ent ( decom_entry, data_buf, msid_info, first_pass )
    struct shm_decom    *decom_entry;
    register char       *data_buf;
    struct msid_ent     *msid_info;
    short               first_pass;
{ return ( 0 ); }

int ddd ( dcm_buf, data_buf, indx, first_pass )
    register struct shm_decom   *dcm_buf;
    register char               *data_buf;
    int                         indx;
    short                       first_pass;
{ return ( 0 ); }

int ddd_msid ( ready )
    int     ready;
{ return ( 0 ); }

int ddd_menu ( )
{ return ( 0 ); }

int dec_val ( char_str )
    char    *char_str;
{ return ( 0 ); }

void draw_axs ( disp_num, plot_ptr, axis_ptr, x, y, width, height )
    short               disp_num;
    struct plot_ptrs    *plot_ptr;
    struct axis_info    *axis_ptr;
    short               x,
                        y,
                        width,
                        height;
{ return; }

int draw_ovl ( act_plot_ptr )
    struct plot_ptrs    *act_plot_ptr;
{ return ( 0 ); }

void draw_plt ( disp_num, plot_info_ptr, x, y, width, height )
    short               disp_num;
    struct plot_ptrs    *plot_info_ptr;
    short               x,
                        y,
                        width,
                        height;
{ return; }

int edit_colors ( )
{ return ( 0 ); }

int color_scales ( )
{ return ( 0 ); }

int color_select ( color )
    int     color;
{ return ( 0 ); }

int ex_msgsnd ( cmd_ptr )
    struct pfkey_defs   *cmd_ptr;
```

```
{ return ( 0 ); }

int exit_disp ( disp_num )
    short   disp_num;
{ return ( 0 ); }

int extract ( data_ptr, decom_ptr )
    char                *data_ptr;
    struct shm_decom    *decom_ptr;
{ return ( 0 ); }

int first_proc ( )
{ return ( 0 ); }

int flt_data ( )
{ return ( 0 ); }

int flt_data_menu ( )
{ return ( 0 ); }

int gdr_next ( )
{ return ( 0 ); }

int get_disp ( )
{ return ( 0 ); }

int get_fn ( dir_file_name, no_dir_fn )
    char        *dir_file_name,
                *no_dir_fn;
{ return ( 0 ); }

int get_plot ( )
{ return ( 0 ); }

int hist_tab ( )
{ return ( 0 ); }

int ht_init ( entry_num, htab_num )
    long            entry_num;
    long            htab_num;
{ return ( 0 ); }

int init ( )
{ return ( 0 ); }

int control_fpe ( )
{ return ( 0 ); }

int init_disp ( disp_num )
    short   disp_num;
{ return ( 0 ); }

int init_fg ( disp_num )
    short   disp_num;
{ return ( 0 ); }

int init_label ( )
{ return ( 0 ); }

int int_ln ( ulx, uly, lrx, lry, points, num_pts )
    short   ulx,
            uly,
            lrx,
            lry,
```

```
            num_pts;
    XPoint  points[];
{ return ( 0 ); }

int lim_grp ( )
{ return ( 0 ); }

void lim_ln ( disp_num, plot_info_ptr, line_ptr, tmplt_ptr )
    int                 disp_num;
    struct plot_ptrs    *plot_info_ptr;
    struct lim_lines    *line_ptr;
    struct plot_tmplt   *tmplt_ptr;
{ return; }

int limit_val ( limit )
    char    limit[];
{ return ( 0 ); }

int list_files ( limit_list, hist_flag )
    int     limit_list,
            hist_flag;
{ return ( 0 ); }

int main ( argc, argv )
    int     argc;
    char    **argv;
{ return ( 0 ); }

int new_disp ( )
{ return ( 0 ); }

int p_atimei ( t_string )
    char    t_string[];
{ return ( 0 ); }

int p_dataval ( decom_ptr )
    struct shm_decom    *decom_ptr;
{ return ( 0 ); }

void p_itimea ( time_int, time_char )
    int     time_int;
    char    time_char[15];
{ return; }

int parse_cmd ( cmd_struct, cmd_string, cmd_string_length, pbi_or_pfkey, version )
    struct pfkey_defs   *cmd_struct;
    char                *cmd_string;
    int                 cmd_string_length,
                        pbi_or_pfkey,
                        version;
{ return ( 0 ); }

int pbi_cmd ( )
{ return ( 0 ); }

int pbi_config ( disp_num, redraw_rect, pbi_changed, number_of_changes )
    short                   disp_num;
    struct pbi_redraw_rect  redraw_rect;
    struct pbi_changes      *pbi_changed;
    int                     number_of_changes;
{ return ( 0 ); }

int pbi_free ( )
{ return ( 0 ); }
```

```
int pbi_host ( )
{ return ( 0 ); }

int pbi_hot ( x, y )
    int     x,
            y;
{ return ( 0 ); }

int pbi_local ( )
{ return ( 0 ); }

int pbi_setup ( flt, datatype )
    char        *flt,
                *datatype;
{ return ( 0 ); }

int pbi_updt ( modified, pbi_start, pbi_entries )
    int             modified;
    PBI_ENTRY       *pbi_start;
    int             pbi_entries;
{ return ( 0 ); }

int pf_chk ( New_Com )
    struct pfkey_defs   *New_Com;
{ return ( 0 ); }

int plot_msid ( disp_num, act_plot_ptr )
    short               disp_num;
    struct plot_ptrs   *act_plot_ptr;
{ return ( 0 ); }

int plot_ovl ( ready )
    short       ready;
{ return ( 0 ); }

int ovl_menu ( )
{ return ( 0 ); }

int free_lists ( )
{ return ( 0 ); }

int proc_plt ( disp_num, plot_ptr )
    short               disp_num;
    struct plot_ptrs    *plot_ptr;
{ return ( 0 ); }

int read_disp ( )
{ return ( 0 ); }

int read_fgr ( disp_num, ddf_ffp )
    short   disp_num;
    FILE    *ddf_ffp;
{ return ( 0 ); }

int read_files ( limit_list )
    int     limit_list;
{ return ( 0 ); }

int read_ovls ( list_file )
    char    ***list_file;
{ return ( 0 ); }

int read_pbi ( )
```

```
{ return ( 0 ); }

int read_pf ( default_flag, disp_name )
    char    default_flag,
            disp_name[DNAME_LEN];
{ return ( 0 ); }

int read_plt ( disp_num, plot_ptr )
    short               disp_num;
    struct plot_ptrs    *plot_ptr;
{ return ( 0 ); }

int readbg ( disp_num )
    short   disp_num;
{ return ( 0 ); }

int readfg ( disp_num )
    short   disp_num;
{ return ( 0 ); }

int redraw ( disp_num, ulx, uly, lrx, lry )
    short   disp_num,
            ulx,
            uly,
            lrx,
            lry;
{ return ( 0 ); }

int redwbg ( disp_num, ulx, uly, lrx, lry )
    short   disp_num,
            ulx,
            uly,
            lrx,
            lry;
{ return ( 0 ); }

int redwfg ( disp_num, ulx, uly, lrx, lry )
    short   disp_num,
            ulx,
            uly,
            lrx,
            lry;
{ return ( 0 ); }

int sel_disp ( )
{ return ( 0 ); }

int set_cmap ( widget )
    Widget  widget;
{ return ( 0 ); }

unsigned long set_gc ( xdisplay, gc, gc_val, graph_col, line_type, line_wdth,
                       pat_type, pat_sizex, pat_sizey, font )

    Display     *xdisplay;
    GC          gc;
    XGCValues   *gc_val;
    short       graph_col,
                line_type,
                pat_type,
                pat_sizex,
                pat_sizey;
    float       line_wdth;
    Font        font;
```

```
{ return ( (unsigned long)0 ); }

int set_label ( widget, label )
    Widget  widget;
    char    *label;
{ return ( 0 ); }

int set_timer ( disp_num )
    short   disp_num;
{ return ( 0 ); }

int shm_creat ( )
{ return ( 0 ); }

int sort_msid ( msid_list, nbr_msids, nbr_recs )
    struct msid_record  *msid_list;
    short               nbr_msids,
                        nbr_recs;
{ return ( 0 ); }

int stat_col(status, msid_info)
    long            status;
    struct msid_ent *msid_info;
{ return ( 0 ); }

void tick_mk ( disp_num, plot_info_ptr, gc, xpos, ypos, length, xory )
    short               disp_num;
    struct plot_ptrs    *plot_info_ptr;
    GC                  gc;
    float               xpos,
                        ypos,
                        length;
    char                xory;
{ return; }

int time_val ( char_str )
    char    *char_str;
{ return ( 0 ); }

int ui_init ( argc, argv )
    int     argc;
    char    **argv;
{ return ( 0 ); }

int unlatch ( )
{ return ( 0 ); }

int unv_plot ( )
{ return ( 0 ); }

int unv_menu ( )
{ return ( 0 ); }

int display_msid ( )
{ return ( 0 ); }

int save_xy ( )
{ return ( 0 ); }

int save_msid ( )
{ return ( 0 ); }

int upd_rate ( )
{ return ( 0 ); }
```

```
int upd_rate_menu ( update_rate )
    int     update_rate;
{ return ( 0 ); }

int update ( disp_num )
    short   disp_num;
{ return ( 0 ); }

int updtbg ( disp_num )
    short   disp_num;
{ return ( 0 ); }

int updtfg ( disp_num, decom_ptr, lmsid, tab_info, status )
    short               disp_num;
    struct shm_decom    *decom_ptr;
    struct msid_ent     *lmsid;
    struct tabular_ent  *tab_info;
    long                status;
{ return ( 0 ); }

int updtht ( )
{ return ( 0 ); }

int short val_dt ( strm_type )
    char    strm_type[];
{ return ( 0 ); }

int val_fn ( file_name, chk_wex )
    char    *file_name;
    short   chk_wex;
{ return ( 0 ); }

int val_msid ( list, count, msid )
    char    **list,
            *msid;
    int     count;
{ return ( 0 ); }

int val_ppl ( file_name )
    char    *file_name;
{ return ( 0 ); }

int val_src ( data_src, real_src )
    char    *data_src,
            *real_src;
{ return ( 0 ); }

int valmsid ( msid )
    char    msid[];
{ return ( 0 ); }

int zoom ( )
{ return ( 0 ); }
```

```
/*****************************************************************************
 * MODULE NAME: draw_axs
 *
 *  This function draws axes, tick marks, and grid lines for the plot
 *  associated with the given axis information.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Tod Milam - Ford Aerospace Corporation/Houston
 *
 *
 * MODIFIED FOR X-WINDOWS BY:
 *
 *  Ronnie Killough -   Software Engineering Section
 *                      Data Systems Department
 *                      Automation and Data Systems Division
 *                      Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <string.h>
#include <wex/EXmsg.h>
#include <constants.h>
#include <disp.h>
#include <DDdisp.h>
#include <DDplot.h>


extern struct dm_shmemory   *Dm_Address;    /* ptr to DM shared memory      */


void draw_axs (disp_num, plot_ptr, axis_ptr, x, y, width, height)

    short   disp_num;               /* display number containing plot       */

    struct plot_ptrs    *plot_ptr;  /* ptr to plot template structure       */

    struct axis_info    *axis_ptr;  /* ptr to axis to draw                  */

    short   x, y, width, height;    /* coord of exposed plot area           */

{
    Display *xdisplay;              /* ptr to X display structure           */
    Window  xwindow;                /* XID of effective display window      */
    XPoint points[2];               /* axis endpoints                       */
    XGCValues   *gc_val;            /* ptr to GC values struct in DM sh mem */
    GC  gc;                         /* XID of GC in DM sh memory            */

    struct plot_tmplt   *tmplt_ptr; /* ptr to plot positional information   */

    double  factor_x, factor_y;     /* transformation factors               */

    float   space, space2,          /* spacing for maj/min tick mks & grid  */
            start, stop;            /* used for calculating tick marks      */

    unsigned long gc_mask;          /* GC values mask for XChangeGC         */

    int     count, count2,          /* loop control for counting tick marks */
            i;                      /* loop control                         */

    short   offset_x, offset_y,     /* zoom coordinate offset values        */
            major_tick, minor_tick, /* maj/min tick mk length in pixels     */
```

```
            major_offset,           /* offset location of major tick marks  */
            minor_offset;           /* offset location of major tick marks  */


    D(printf("START draw_axs\n"));

/*
 *  Setup X variables, transformation factors,
 *  and local pointers.
 */

    xdisplay = Dm_Address->xdisplay[disp_num];
    xwindow = XtWindow(plot_ptr->draw_win);
    gc_val = &Dm_Address->gc_val[disp_num];
    gc = Dm_Address->gc[disp_num];

    tmplt_ptr = plot_ptr->plot_pos;

    factor_x = tmplt_ptr->factor_x;
    factor_y = tmplt_ptr->factor_y;
    offset_x = tmplt_ptr->offset_x;
    offset_y = tmplt_ptr->offset_y;

/*
 *  Calculate the major tick mark length. Tick mark lengths
 *  are calculated using the X transformation only to prevent different
 *  length tick marks on the X and Y axes.
 */

    major_tick = (short) MAJOR_TICK_LEN * factor_x;
    minor_tick = (short) MINOR_TICK_LEN * factor_x;

/*
 *  Set the line parameters for the axes and tick marks
 */

    if (gc_mask = set_gc(xdisplay, gc, gc_val, axis_ptr->axis_col, 1, 1.0,
                         NO_CHANGE, NO_CHANGE, NO_CHANGE, NO_CHANGE))
        XChangeGC(xdisplay, gc, gc_mask, gc_val);

/*
 *  Draw the axis and tick marks only if the axis is visible
 */

    if (axis_ptr->vis_flag == 'V') {

/*
 *      If angle polar processing then draw
 *      x and y axes each at position 50
 */

        if (axis_ptr->axis_xory == 'Y' && axis_ptr->axis_type == POLAR) {
            points[0].x = 0;
            points[0].y = (short) ((100 - axis_ptr->axis_pos) * factor_y);
            points[1].x = (short) (100.0 * factor_x);
            points[1].y = points[0].y;

            XDrawLine(xdisplay, xwindow, gc, points[0].x, points[0].y,
                      points[1].x, points[1].y);

            points[0].x = (short) (axis_ptr->axis_pos * factor_x);
            points[0].y = 0;
            points[1].x = points[0].x;
            points[1].y = (short) (100.0 * factor_y);
```

```
            XDrawLine(xdisplay, xwindow, gc, points[0].x, points[0].y,
                        points[1].x, points[1].y);

/*
 *      If axis is not a polar distance axis, draw x or y
 *      axis for current axis record.
 */

        } else if (!(axis_ptr->axis_xory == 'X'
                && axis_ptr->axis_type == POLAR)) {

            if (axis_ptr->axis_xory == 'X') {
                points[0].x = 0;
                points[0].y = axis_ptr->cur_axis_pos;
                points[1].x = tmplt_ptr->drw_width;
                points[1].y = points[0].y;
            } else if (axis_ptr->axis_xory == 'Y') {
                points[0].x = axis_ptr->cur_axis_pos;
                points[0].y = 0;
                points[1].x = points[0].x;
                points[1].y = tmplt_ptr->drw_height;
            } else
                return;

            XDrawLine(xdisplay, xwindow, gc, points[0].x, points[0].y,
                            points[1].x, points[1].y);

/*
 *          If axis is near top or right of plot bounding box (pbx)
 *          align tick marks with axis using an offset. If axis is in the
 *          middle of the pbx, center the tick marks on the axis.  Else
 *          the axis is near the bottom or left of pbx and tick marks will
 *          align to axis w/o an offset.
 */

            if (axis_ptr->axis_xory == 'X') {

                if (axis_ptr->axis_pos < 5) {   /* axis at bottom of pbx */
                major_offset = major_tick * -1;
                minor_offset = minor_tick * -1;
                } else if (axis_ptr->axis_pos > 95) /* axis at top of pbx */
                    major_offset = minor_offset = 0;
                else {                              /* axis in midst of pbx */
                    major_offset = major_tick / -2;
                    minor_offset = minor_tick / -2;
                }

            } else {

                if (axis_ptr->axis_pos < 5) {   /* axis at left of pbx */
                    major_offset = minor_offset = 0;
                } else if (axis_ptr->axis_pos > 95) {   /* axis at top of pbx */
                    major_offset = major_tick * -1;
                    minor_offset = minor_tick * -1;
                } else {                            /* axis in midst of pbx */
                    major_offset = major_tick / -2;
                    minor_offset = minor_tick / -2;
                }
            }

/*
 *          Draw major tick marks
 */
```

```
            if (axis_ptr->maj_ticks > 0) {
                space = 100.0 / (axis_ptr->maj_ticks - 1);

                for (count = 0; count <= axis_ptr->maj_ticks; count++) {

                    if (axis_ptr->axis_xory == 'X')
                        tick_mk (disp_num, plot_ptr, gc,
                                (short)(count * space * factor_x) + offset_x,
                                axis_ptr->cur_axis_pos + major_offset,
                                major_tick, axis_ptr->axis_xory);
                    else if (axis_ptr->axis_xory == 'Y')
                        tick_mk (disp_num, plot_ptr, gc,
                            axis_ptr->cur_axis_pos + major_offset,
                            (short)((100.0 - (count * space))
                                                * factor_y) + offset_y,
                            major_tick, axis_ptr->axis_xory);
                }
            }


/*
 *          Draw minor tick marks
 */

            if (axis_ptr->min_ticks > 0) {
                space = 100.0 / (axis_ptr->maj_ticks - 1);

                for (count = 0; count < (axis_ptr->maj_ticks - 1); count++) {
                    start = (count) * space;
                    stop = (count + 1) * space;
                    space2 = (stop - start) / (axis_ptr->min_ticks + 1);

                    for (count2 = 0; count2 < axis_ptr->min_ticks ; count2++) {

                        if (axis_ptr->axis_xory == 'X')
                            tick_mk (disp_num, plot_ptr, gc,
                                    (short) (((count2 + 1) * space2 + start)
                                                * factor_x) + offset_x,
                                    axis_ptr->cur_axis_pos + minor_offset,
                                    minor_tick, axis_ptr->axis_xory);

                        else if (axis_ptr->axis_xory == 'Y')
                            tick_mk (disp_num, plot_ptr, gc,
                                    axis_ptr->cur_axis_pos + minor_offset,
                                    (short) ((100.0 - ((count2 + 1) * space2
                                            + start)) * factor_y) + offset_y,
                                    minor_tick, axis_ptr->axis_xory);
                    }
                }

            }   /* end if minor ticks > 0 */

        }   /* end if not distance polar axis */

    }   /* end if axes are visible */

/*
 *  Draw the grid if it is visible
 */

    if (axis_ptr->grid_flag == 'Y') {

        if (gc_mask = set_gc(xdisplay, gc, gc_val, axis_ptr->grd_color,
```

```
                                axis_ptr->grid_type, 1.0,
                                NO_CHANGE, NO_CHANGE, NO_CHANGE, NO_CHANGE))
            XChangeGC(xdisplay, gc, gc_mask, gc_val);


/* RLK 10/25/90 Polar axes not functional.

        if (axis_ptr->axis_type == POLAR)
            DDp_polar (axis_ptr);

        else {
*/

            space = 100.0 / (axis_ptr->grid_gran - 1);

            for (count = 0; count <= axis_ptr->grid_gran; count++) {

                if (axis_ptr->axis_xory == 'X')
                    tick_mk (disp_num, plot_ptr, gc,
                            (short) (count * space * factor_x) + offset_x,
                            0, tmplt_ptr->drw_height, axis_ptr->axis_xory);
                else if (axis_ptr->axis_xory == 'Y')
                    tick_mk (disp_num, plot_ptr, gc, 0,
                            (short) ((100.0 - (count * space))
                                                    * factor_y) + offset_y,
                            tmplt_ptr->drw_width, axis_ptr->axis_xory);
            }

/*
        }
*/
    }

    D(printf("END draw_axs\n"));
    return;
}
```

```
/*****************************************************************************
 * MODULE NAME: draw_ovl.c
 *
 *  This routine draws a plot overlay.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Richard Romeo   - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <X11/Xlib.h>
#include <fcntl.h>
#include <wex/EXmsg.h>
#include <constants.h>
#include <disp.h>
#include <DDdisp.h>
#include <DDplot.h>


extern struct dm_shmemory   *Dm_Address;    /* Pointer to Display Manager SHM.        */
extern struct plot_ptrs     *Plot_info_ptr; /* Ptr thru plot ptr files.               */

extern short                Disp_Num,       /* Display Number.                        */
                            End_of_file;    /* End of data file flag.                 */

extern int                  errno;          /* Error return value.                    */


int draw_ovl ( plot_ptr )

    struct plot_ptrs    *plot_ptr;      /* Ptr thru plot ptr files.                */
{
    struct shm_decom    *loc_dcm_ptr;       /* Ptr thru plot decom structure.      */
    struct plot_hdr     *plot_hdr_ptr;      /* Ptr thru plot header.               */
    struct shm_decom    *save_dcm_ptr;      /* Save decom buffer ptr.              */

    int                 k, m, i,            /* Loop count variable.                */
                        next_offset,        /* Nbr of bytes for offset cal.        */
                        ovr_plot_fp,        /* Local plot file pointer.            */
                        org_plot_fp,        /* Save plot data ptr.                 */
                        save_buf_size;      /* Save the data buffer size.          */

    char                *calloc ( ),        /* Get malloc as a pointer.            */
                        *malloc ( ),        /* Get malloc as a pointer.            */
                        *save_data_ptr,     /* Save the data buffer ptr.           */
                        plot_ovr[DNAME_LEN + 5];

    double              sav_seconds;        /* Save seconds elapsed.               */

    D(printf("START DRAW_OVL\n"));
/*
 *  Copy plot name from memory and store into local parameter.
 */
```

```
    strcpy ( plot_ovr, plot_ptr->plot_ovr );

/*
 *  Open plot overlay data file and read decom buffer.
 */

    strcat ( plot_ovr, ".ovr" );
    org_plot_fp = plot_ptr->plot_fp;
    plot_ptr->plot_fp = open ( plot_ovr, O_RDONLY );
    plot_ptr->ovr_flg = NO;
    if ( plot_ptr->plot_fp == INVALID ) {
        tui_msg ( M_YELLOW, "Error %d on opening the plot overlay data file", errno );
        plot_ptr->ovr_flg = NO;
        plot_ptr->plot_fp = org_plot_fp;
        return ( -1 );
    }
    ovr_plot_fp = plot_ptr->plot_fp;

/*
 *  Skip over decom buffer header and read decom buffer to memory.
 */

    lseek ( ovr_plot_fp, 80, 0 );
    save_dcm_ptr = plot_ptr->plt_decom;
    plot_hdr_ptr = plot_ptr->header;
    if ( plot_hdr_ptr->msid_num > 0 ) {
        plot_ptr->plt_decom = ( struct shm_decom * )
            calloc ( (unsigned)plot_hdr_ptr->msid_num, sizeof ( struct shm_decom ) );

        if ( plot_ptr->plt_decom == NULL ) {
            tui_msg ( M_YELLOW, "Error on allocating memory for plot decom" );
            plot_ptr->plot_fp = org_plot_fp;
            plot_ptr->plt_decom = save_dcm_ptr;
            return ( -1 );
        }
        loc_dcm_ptr = plot_ptr->plt_decom;
        save_buf_size = plot_ptr->buf_size;
        plot_ptr->buf_size = 0;
        next_offset = 0;

        for ( m = 0; m < plot_hdr_ptr->msid_num; m++ ) {
            read ( ovr_plot_fp, &loc_dcm_ptr->size, sizeof ( int ) );
            read ( ovr_plot_fp, &loc_dcm_ptr->length, sizeof ( int ) );
            read ( ovr_plot_fp, &loc_dcm_ptr->num_samps, sizeof ( short ) );
            read ( ovr_plot_fp, &loc_dcm_ptr->attribute, sizeof ( char ) );
            read ( ovr_plot_fp, &loc_dcm_ptr->error, sizeof ( char ) );
            lseek ( ovr_plot_fp, 12, 1 );

#ifdef FAC
            if ( loc_dcm_ptr->error != NULL ) {
                loc_dcm_ptr->num_samps = 1;
                loc_dcm_ptr->length = 4;
            }
#endif

            loc_dcm_ptr->sample_size = loc_dcm_ptr->size / loc_dcm_ptr->num_samps;
            plot_ptr->buf_size = plot_ptr->buf_size + 2 + loc_dcm_ptr->size;

            loc_dcm_ptr->offset = next_offset;
            next_offset = loc_dcm_ptr->size + loc_dcm_ptr->offset + 2;
            loc_dcm_ptr++;
        }
```

```
        loc_dcm_ptr = plot_ptr->plt_decom;
        for ( k = 0; k < plot_hdr_ptr->msid_num; k++ ) {
            /*EXmsg ( M_YELLOW, "size %x",        loc_dcm_ptr->size        );*/
            /*EXmsg ( M_YELLOW, "length %x",      loc_dcm_ptr->length      );*/
            /*EXmsg ( M_YELLOW, "num_samps %x",   loc_dcm_ptr->num_samps   );*/
            /*EXmsg ( M_YELLOW, "attr %x",        loc_dcm_ptr->attribute   );*/
            /*EXmsg ( M_YELLOW, "error %x ",      loc_dcm_ptr->error       );*/
            /*EXmsg ( M_YELLOW, "offset %x",      loc_dcm_ptr->offset      );*/
            /*EXmsg ( M_YELLOW, "sample size %x", loc_dcm_ptr->sample_size );*/
            loc_dcm_ptr++;
        }

    /*
     *      Allocate space for data buffer and set ovr plot active flag.
     */

        save_data_ptr = plot_ptr->plot_data;
        plot_ptr->plot_data = malloc ( plot_ptr->buf_size );
        if ( plot_ptr->plot_data == NULL ) {
            tui_msg ( M_YELLOW, "Error %d on creating data buffer space", errno );
            plot_ptr->plot_fp = org_plot_fp;
            plot_ptr->plt_decom = save_dcm_ptr;
            plot_ptr->buf_size = save_buf_size;
            plot_ptr->plot_data = save_data_ptr;
            return ( -1 );
        }
        sav_seconds = plot_ptr->seconds_elapsed;
        plot_ptr->seconds_elapsed = 0;

    /*
     *          Reset all first point flags for drawing overlay.
     */

            for (i=0; i<plot_ptr->header->actual_msids; i++)
                (plot_ptr->msids + i)->first_pt = YES;

    /*
     *      While not end of overlay data file continue
     *      to plot overlay points.
     */

        End_of_file = NO;
        plot_ptr->ovl_color_flg = YES;
        while ( End_of_file == NO )
            proc_plt ( Disp_Num, plot_ptr );
        plot_ptr->ovl_color_flg = NO;

    /*
     *          Reset all first point flags for redraw of plot.
     */

            for (i=0; i<plot_ptr->header->actual_msids; i++)
                (plot_ptr->msids + i)->first_pt = YES;

    /*
     *      Call original file to plot the current data.
     */

        /*if ( plot_ptr->act_flg != YES )
            DDorg_file ( plot_ptr );*/

    /*
     *  Restore back the information overwritten for overlay processing.
     */
```

```
        close ( plot_ptr->plot_fp );
        free ( plot_ptr->plot_data );
        plot_ptr->plot_fp = org_plot_fp;
        plot_ptr->plt_decom = save_dcm_ptr;
        plot_ptr->buf_size = save_buf_size;
        plot_ptr->plot_data = save_data_ptr;
        plot_ptr->seconds_elapsed = sav_seconds;
        plot_ptr->ovr_flg = YES;

    } else {
        close ( plot_ptr->plot_fp );
        plot_ptr->plot_fp = org_plot_fp;
    }

    D(printf("END DRAW_OVL\n"));
    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: draw_plt.c
 *
 *  This function calls the functions which draw the axes and
 *  limit/nominal lines of a plot.  This function draws the plot
 *  axis labels.
 *
 * DEVELOPMENT NOTES:
 *
 *  o   Logarithmic and Polar axes have not been tested.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Tod Milam - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Ronnie L. Killough - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <X11/Xlib.h>
#include <math.h>
#include <constants.h>
#include <disp.h>
#include <DDdisp.h>
#include <DDplot.h>
#include <wex/EXmsg.h>


extern struct dm_shmemory *Dm_Address;  /* ptr to DM shared memory          */


void draw_plt (disp_num, plot_ptr, x, y, width, height)

    short               disp_num;       /* display # containing plot        */

    struct plot_ptrs    *plot_ptr;      /* ptr to plot record               */

    short               x, y, width, height;
                                        /* coord. of exposed plot area      */

{
    double  sqrt(),                     /* square root fn used for logs     */
            fabs ();                    /* absolute value fn                */

    XPoint      point;                  /* coord of label to be drawn       */
    XGCValues   *gc_val;                /* ptr to GC values in DM sh mem    */
    GC          gc;                     /* XID of GC in DM shared memory    */
    Display     *xdisplay;              /* ptr to X display in DM sh mem    */
    Window      xwindow;                /* XID of X window of display       */
    Font        label_font;             /* font of axes labels              */

    struct axis_info    *x_ptr;         /* ptr thru x axes                  */
    struct axis_info    *y_ptr;         /* ptr thru y axes                  */
    struct lim_lines    *nl_ptr;        /* ptr thru nom & limit plot lines  */

    double  factor_x, factor_y,         /* plot transformation factors      */
```

```
            low_value, high_value,      /* axis low/high scale values       */
            space,                      /* space between labels             */
            increment;                  /* label value increment            */

    float   diffval,                    /* difference of low and high value */
            loglabel,                   /* logarithmic label to output      */
            fxl, fyl,                   /* temp world coordinates of label  */
            string_offset,              /* offset applied to the coord. of
                                           each label to properly place it
                                           w/ respect to the axis           */
            percent_of_y;               /* used to compute label font size  */

    unsigned long   gc_mask;            /* GC values mask for XChangeGC     */

    int     i,                          /* loop counter                     */
            count,                      /* loop counter for label count     */
            char_width, char_height;    /* width/height of label font       */

    short   multiply = 1,               /* low/high value multiplier        */
            major_tick;                 /* major tick mk length in pixels   */

    char    label[15];                  /* label text                       */


    D(printf("START draw_plt\n"));

/*
 *  Setup X variables and transformation factors
 */

    xdisplay = Dm_Address->xdisplay[disp_num];
    xwindow = XtWindow(plot_ptr->draw_win);
    gc_val = &Dm_Address->gc_val[disp_num];
    gc = Dm_Address->gc[disp_num];

    factor_x = plot_ptr->plot_pos->factor_x;
    factor_y = plot_ptr->plot_pos->factor_y;

/*
 *  Calculate the major tick mark length. Tick mark lengths
 *  are calculated using the X transformation only to prevent different
 *  length tick marks on the X and Y axes.
 */

    major_tick = (short) MAJOR_TICK_LEN * factor_x;

/*
 *  Draw all x axes for this plot
 */

    x_ptr = plot_ptr->axis;

    for (i = 0; i < plot_ptr->header->xaxes_num; i++) {

        if (x_ptr->axis_type == CARTESIAN)
            x_ptr->scale_ratio = 100.0 / (x_ptr->high_value - x_ptr->low_value);

        else if (x_ptr->axis_type == LOGARITHMIC) {
            x_ptr->scale_ratio = 1.0;

            if (x_ptr->low_value > x_ptr->high_value)
                multiply = -1;
            else
                multiply = 1;
```

```
            low_value = x_ptr->low_value * multiply;
            high_value = x_ptr->high_value * multiply;
            diffval = high_value - low_value;
            x_ptr->logval = (float) sqrt ((double) diffval);

        } else  /* POLAR */
            x_ptr->scale_ratio =
                        100.0 / ((x_ptr->high_value - x_ptr->low_value) * 2);

        draw_axs(disp_num, plot_ptr, x_ptr, x, y, width, height);

        x_ptr++;
    }

/*
 *  Draw all y axes for this plot
 */

    y_ptr = plot_ptr->axis + plot_ptr->header->xaxes_num;

    for (i = 0; i < plot_ptr->header->yaxes_num; i++) {

        if (y_ptr->axis_type == LOGARITHMIC) {
            y_ptr->scale_ratio = 1.0;

            if (y_ptr->low_value > y_ptr->high_value)
                multiply = -1;
            else
                multiply = 1;

            low_value = y_ptr->low_value * multiply;
            high_value = y_ptr->high_value * multiply;
            diffval = high_value - low_value;
            y_ptr->logval = (float) sqrt ((double) diffval);

        } else  /* CARTESIAN or POLAR */
            y_ptr->scale_ratio = 100.0 / (y_ptr->high_value - y_ptr->low_value);

        draw_axs (disp_num, plot_ptr, y_ptr, x, y, width, height);
        y_ptr++;
    }

/*
 * Draw nominal value lines for this plot
 */

    nl_ptr = plot_ptr->nline;

    for (i = 0; i < plot_ptr->header->nline_num; i++) {
        lim_ln (disp_num, plot_ptr, nl_ptr);
        nl_ptr++;
    }

/*
 * Draw limit lines for this plot
 */

    nl_ptr = plot_ptr->lline;
    for (i = 0; i < plot_ptr->header->lline_num; i++) {
        lim_ln (disp_num, plot_ptr, nl_ptr);
        nl_ptr++;
    }
```

```
/*
 *  Setup font style for labels.
 */

    percent_of_y = plot_ptr->plot_pos->bb_height * .30;

    char_width = 0;      /* font size based on calculated height only */
    char_height = (int) percent_of_y;

    label_font = font_num(disp_num, LABEL_STYLE, char_width, char_height);

/*
 *  Draw x axes labels
 */

    x_ptr = plot_ptr->axis;

    for (i = 0; i < plot_ptr->header->xaxes_num; i++) {
        low_value = x_ptr->low_value;
        high_value = x_ptr->high_value;

/*
 *      Set the text parameters
 */

        if (gc_mask = set_gc(xdisplay, gc, gc_val,
                             x_ptr->axis_col, NO_CHANGE, -1.0,
                             NO_CHANGE, NO_CHANGE, NO_CHANGE, label_font))
            XChangeGC(xdisplay, gc, gc_mask, gc_val);

/*
 *      Axis-type-dependent calculations
 */

        if (x_ptr->axis_type == CARTESIAN) {
            space = 100.0 / (x_ptr->grad_vals - 1);
            increment = (high_value - low_value) / (x_ptr->grad_vals - 1);

/*
 *          If x axis is near top of plot, place labels under the
 *          axis, else place labels over the axis.  Goal is to have
 *          labels on the interior of the plot
 */

            if (x_ptr->axis_pos > 95)
                string_offset = major_tick * -1;
            else
                string_offset = major_tick;

        } else if (x_ptr->axis_type == LOGARITHMIC) {

/*
 *          If the low value is greater then the high value, then set
 *          multiply to -1. This will keep us from trying to square
 *          root a negative value. Later the value will be multiplied
 *          again to get the correct label sign to output.
 */

            if (low_value > high_value)
                multiply = -1;
            else
                multiply = 1;

            high_value = high_value * multiply;
```

```
            low_value = low_value * multiply;

        } else {    /* POLAR */

            if (x_ptr->grad_vals != 0) {
                space = 50.0 / (x_ptr->grad_vals);
                increment = (high_value - low_value) / (x_ptr->grad_vals);
            }
        }


/*
 *      For each graduation, calculate the coordinates and value
 *      of the label and draw it.  Suppress drawing the first axis
 *      label to avoid confusion at axis crossing. Suppress drawing
 *      the last label since it would be drawn outside the plot
 *      bounding box anyway.
 */

        for (count = 1; count < (x_ptr->grad_vals - 1); count++) {

/*
 *          Calculate coordinates of this label
 */

/* RLK 9/26/90  point.y will actually need to be the axis position in pixels
                less the height of the font */

/* RLK 9/26/90  need to account for font height here if axis is near top
                of plot and having to place label under the axis. */

            if (x_ptr->axis_type == POLAR) {
                fx1 = (float) (x_ptr->axis_pos + 30);
                fy1 = 50.0 - (count + 1) * space;
            } else {
                point.x = (short) (count * space * factor_x)
                                        + plot_ptr->plot_pos->offset_x;
                point.y = x_ptr->cur_axis_pos - string_offset;
                            /* - <font_height> if top axis */
            }

/*
 *          Calculate label and convert to string
 */

            /* if processing logarithmic axis, then set up label based on
                logarithmic formula  */

            if (x_ptr->axis_type == CARTESIAN) {

                if (x_ptr->scal_type == 'T')
                    p_itimea((int) (low_value + count * increment), label);
                else
                    sprintf(label, "%.2f", (float)
                            ((low_value + count * increment) * multiply));

            } else if (x_ptr->axis_type == LOGARITHMIC) {
                loglabel = ((x_ptr->logval * ((space * count) / 100))
                    * (x_ptr->logval * ((space * count) / 100))) + low_value;

                if (x_ptr->scal_type == 'T')
                    p_itimea ((int) (loglabel * multiply + 0.5), label);
                else
                    sprintf(label, "%.2f", (float) (loglabel * multiply));
```

```
            } else {    /* POLAR */

                if (x_ptr->scal_type == 'T')
                    p_itimea((int) (low_value + (count + 1.0)
                                            * increment + 0.5), label);
                else
                    sprintf(label, "%.2f", (float)
                            ((low_value + (count + 1) * increment) * multiply));
            }

/*
 *          Draw label on plot
 */

            XDrawString(xdisplay, xwindow, gc, point.x, point.y, label,
                        strlen(label));
        }

        x_ptr++;
    }

/*
 *  Draw Y axis labels
 */

    y_ptr = plot_ptr->axis + plot_ptr->header->xaxes_num;

    for (i = 0; i < plot_ptr->header->yaxes_num; i++) {

        low_value = y_ptr->low_value;
        high_value = y_ptr->high_value;

/*
 *      Set the text parameters for this y axis
 */

        if (gc_mask = set_gc(xdisplay, gc, gc_val,
                             y_ptr->axis_col, NO_CHANGE, -1.0,
                             NO_CHANGE, NO_CHANGE, NO_CHANGE, label_font))
            XChangeGC(xdisplay, gc, gc_mask, gc_val);

/*
 *      Axis-type-dependent calculations for this y axis
 */

        /* if processing logarithmic then set up the log value */

        if (y_ptr->axis_type == CARTESIAN) {
            space = 100.0 / (y_ptr->grad_vals - 1);
            increment = (high_value - low_value) / (y_ptr->grad_vals - 1);

/*
 *          If y axis is near the right of the plot, place label
 *          to the left of the axis, else place to the right of the
 *          axis.  Goal is to have labels on the interior of the plot.
 */

            if (y_ptr->axis_pos > 95)
                string_offset = major_tick * -1;
            else
                string_offset = major_tick;

        } else if (y_ptr->axis_type == LOGARITHMIC) {
```

```
/*
 *          If the low value is greater then the high value, then set
 *          multiply to -1. This will keep us from trying to sqaure
 *          root a negative value. Later the value will be multiplied
 *          again to get the correct label sign to output.
 */

            if (low_value > high_value)
                multiply = -1;
            else
                multiply = 1;

            high_value = high_value * multiply;
            low_value = low_value * multiply;

        } else {    /* POLAR */

            if (y_ptr->grad_vals != 0) {
                space = 50.0 / (y_ptr->grad_vals);
                increment = (high_value - low_value) / (y_ptr->grad_vals);
            }
        }

/*
 *      For each graduation, calculate the value and coordinates
 *      of the label and draw it.  Suppress drawing the first axis
 *      label to avoid confusion at axis crossing. Suppress drawing
 *      the last label since it will be drawn outside the plot
 *      bounding box anyway.
 */

        for (count = 1; count < (y_ptr->grad_vals - 1); count++) {

/*
 *          Calculate the value of this y axis label and convert to string
 */

            if (y_ptr->axis_type == LOGARITHMIC) {
                loglabel = ((y_ptr->logval * ((space * count) / 100))
                    * (y_ptr->logval * ((space * count) / 100))) + low_value;

                if (y_ptr->scal_type == 'T')
                    p_itimea ((int) (loglabel * multiply), label);
                else
                    sprintf(label, "%.2f", (float) (loglabel * multiply));

            } else {    /* CARTESIAN */

                if (y_ptr->scal_type == 'T')
                    p_itimea ((int) (low_value + count * increment), label);
                else
                    sprintf(label, "%.2f", (float)
                                ((low_value + count * increment) * multiply));
            }

/*
 *          Calculate the coordinates of this label for this y axis
 */

/* RLK 9/26/90  May need to use XTextExtents to calculate the width
 *              of the string and use this to place the string right
 *              justified to the inside of a y axis near the right of
 *              the plot bounding box.
```

```
 */

            point.x = y_ptr->cur_axis_pos + string_offset;
            point.y = (short) ((100.0 - count * space) * factor_y)
                                            + plot_ptr->plot_pos->offset_y;

/*
 *          If axis is not of polar type, draw label to plot
 */

            if (y_ptr->axis_type != POLAR)
                XDrawString(xdisplay, xwindow, gc, point.x, point.y, label,
                                strlen(label));
        }

        y_ptr++;
    }

    D(printf("END draw_plt\n"));
    return;
}
```

```
/*****************************************************************************
 * MODULE NAME: edit_colors.c
 *
 *  This file contains functions needed to control and allow manipulation of
 *  the color map.
 *
 *
 * INTERNAL FUNCTIONS:
 *
 *  o   cb_color_manager    -   Main callback event handler for interactive
 *                              color editing.
 *
 *  o   cb_event_color      -   Callback event handler for input events in
 *                              the window used to draw colors.
 *
 *  o   cb_expose_color     -   Callback event handler for expose events in
 *                              the window used to draw colors.
 *
 *  o   color_scales        -   Updates the scale widgets used to manipulate
 *                              RGB values.
 *
 *  o   color_select        -   Draws a highlight rectangle around the current
 *                              color.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <string.h>
#include <memory.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Shell.h>
#include <Xm/mwm.h>
#include <Xm/Xm.h>
#include <Xm/DrawingA.h>
#include <Xm/Form.h>
#include <Xm/Label.h>
#include <Xm/Scale.h>
#include <Xm/ScrolledW.h>
#include <constants.h>
#include <user_inter.h>
#include <wex/EXmsg.h>


static Widget       Shell,      /* Shell widget which is the parent of the
                                 * color select popup.
                                 */
                    Sc_rgb_red, /* Scale widget used to allow the user to ad-
                                 * just the amount of red in a color.
                                 */
                    Sc_rgb_green,
                                /* Scale widget used to allow the user to ad-
                                 * just the amount of green in a color.
                                 */
                    Sc_rgb_blue;/* Scale widget used to allow the user to ad-
                                 * just the amount of blue in a color.
```

```
                                */

static Window       Win;        /* Set to the window in which the color map is
                                 * displayed.
                                 */

static XColor       Cmap_orig[MAX_COLORS],
                                /* Array used to retain the color map in its
                                 * state prior to editing. Needed to restore
                                 * the color map if necessary.
                                 */
                    *Cmap_cur;  /* Pointer to the array of colors which is
                                 * currently available for editing.
                                 */

static int          Num_colors, /* Set to the number of colors currently being
                                 * edited.
                                 */
                    Cur_pixel,  /* The index in the current color map for the
                                 * pixel currently being edited.
                                 */
                    Flag;       /* Flag used to determine when the user is
                                 * finished with the color selection popup.
                                 */

Display             *Disp;
int                 Scr;
GC                  gc;

extern Widget       Top;        /* The top level widget which is the parent
                                 * of transient shells.
                                 */

extern Colormap     Main_cmap;
```

```
/*****************************************************************************
 * MODULE NAME: edit_colors
 *
 *  This function presents a popup which allows the user to manipulate colors.
 *  The popup presents a grid of the current colors and a set of scales which
 *  allows the colors to be modified.
 *****************************************************************************/

int edit_colors ( )
{
    register int            i;

    static XtCallbackRec    cb[] = {
        { (XtCallbackProc)NULL, (caddr_t)NULL },
        { (XtCallbackProc)NULL, (caddr_t)NULL }
    };

    Widget                  form, f_rgb, f_clr, f_cmd,
                            sw_color, da_color;

    XSetWindowAttributes    attributes;

    Arg                     args[10];

    XEvent                  event;

    XColor                  cmap_rgb[1];

    XtCallbackProc          cb_color_manager(),
                            cb_expose_color (),
                            cb_event_color  (),
                            cb_help         ();

    unsigned long           mask;

    D(printf("START EDIT_COLORS\n"));

/*
 *  Save the display and the screen.
 */

    Disp = XtDisplay ( Top );
    Scr  = DefaultScreen ( Disp );

/*
 *  Save a local copy of the current color map.
 */

    for ( i = 0; i < MAX_COLORS; i++ )
        Cmap_orig[i].pixel = i;
    XQueryColors ( Disp, Main_cmap, Cmap_orig, MAX_COLORS );

/*
 *  Save the current color map, number of colors.
 */

    Cmap_cur   = cmap_rgb;
    Num_colors = 1;
    memcpy ( (char *)&Cmap_cur[0], (char *)&Cmap_orig[NUM_MOTIF_COLORS],
             sizeof(XColor) );

/*
 *  Create the shell widget with an argument which specifies the minimum width
```

```
 *  of the shell.
 */

    i = 0;
    XtSetArg ( args[i], XmNminWidth, 308 ); i++;
    Shell = tui_create_trans_shell ( "Color Editor", args, i );

/*
 *  Create the main form which is the parent of all other widgets and child
 *  forms.
 */

    i = 0;
    form = tui_create_form ( Shell, "form", TRUE, args, i );

/*
 *  Create the area used to display the color map. This area consists of a
 *  form which includes a label and a scrolled window which in turn contains
 *  the drawing area used for the color map.
 */

    i = 0;
    XtSetArg ( args[i], XmNleftPosition, CLR_LEFT_RGB ); i++;
    f_clr = tui_create_form  ( form,  "f_clr", FALSE, args, i );

    i = 0;
    tui_create_label ( f_clr, "l_colors", "Colormap", args, i );

    i = 0;
    XtManageChild ( sw_color  =
            XmCreateScrolledWindow ( f_clr, "sw_colors", args, i ) );

/*
 *  Create the drawing area widget. Set the width and height to the required
 *  size of the colors area and add callback for expose events. Note that
 *  input events (mouse selection) is not allowed during overlay color
 *  editing.
 */

    D(printf("  Create drawing area\n"));
    i = 0;
    XtSetArg ( args[i], XmNwidth,  CLR_TOTAL ); i++;
    XtSetArg ( args[i], XmNheight, CLR_TOTAL ); i++;
    XtManageChild ( da_color =
            XmCreateDrawingArea ( sw_color, "drawclr", args, i ) );

    XtAddCallback ( da_color, XmNexposeCallback, cb_expose_color, 0 );
    XtAddCallback ( da_color, XmNinputCallback,  cb_event_color,  0 );

/*
 *  Create the form and scale widgets needed to allow the user to adjust the
 *  RGB contents of a color. Changing a scale value causes a callback to
 *  the (cb_color_manager) function.
 */

    D(printf("  Create scales\n"));
    i = 0;
    f_rgb = tui_create_form ( f_clr, "f_rgb", FALSE, NULL, i );

    cb[0].callback = (XtCallbackProc)cb_color_manager;
    cb[0].closure  = (caddr_t)-1;

    i = 0;
    XtSetArg ( args[i], XmNvalueChangedCallback, cb ); i++;
```

```
    XtManageChild ( Sc_rgb_red   =
            XmCreateScale ( f_rgb, "sc_rgb_red",   args, i ) );
    XtManageChild ( Sc_rgb_green =
            XmCreateScale ( f_rgb, "sc_rgb_green", args, i ) );
    XtManageChild ( Sc_rgb_blue  =
            XmCreateScale ( f_rgb, "sc_rgb_blue",  args, i ) );

    i = 0;
    tui_create_label ( f_rgb, "l_rgb_red",   "Red",   args, i );
    tui_create_label ( f_rgb, "l_rgb_green", "Green", args, i );
    tui_create_label ( f_rgb, "l_rgb_blue",  "Blue",  args, i );

/*
 *  Create a separator to go between the main widgets and the form with the
 *  command widgets.
 */

    D(printf("  Create separator\n"));
    i = 0;
    tui_create_separator ( form, "sep", args, i );

/*
 *  Create the form and 4 command widgets to allow the user to save, cancel,
 *  restore, and get help.
 */

    D(printf("  Create commands\n"));
    i = 0;
    f_cmd = tui_create_form ( form, "f_cmd", FALSE, args, i );
    tui_create_pushbutton ( f_cmd, "OK",      cb_color_manager,
                        (caddr_t)-2, args, i );
    tui_create_pushbutton ( f_cmd, "Cancel",  cb_color_manager,
                        (caddr_t)-3, args, i );
    tui_create_pushbutton ( f_cmd, "Restore", cb_color_manager,
                        (caddr_t)-4, args, i );
    tui_create_pushbutton ( f_cmd, "Help",    cb_help,
                        3,           args, i );

/*
 *  Realize and pop up the shell widget.
 */

    XtRealizeWidget ( Shell );
    XtPopup         ( Shell, None );
    set_cmap ( Shell );

/*
 *  Set the attributes necessary to create the actual window used for the
 *  available colors.
 */

    attributes.save_under       = 0;
    attributes.backing_store    = NotUseful;
    attributes.border_pixel     = 1;
    attributes.background_pixel = 0;
    attributes.bit_gravity      = NorthWestGravity;

    mask = CWBackingStore | CWSaveUnder | CWBackPixel |
           CWBorderPixel | CWBitGravity;

/*
 *  Create and save the window for the drawing area widget.
 */
```

```
    XtCreateWindow ( da_color, CopyFromParent, DefaultVisual ( Disp, Scr ), mask,
                     &attributes );
    Win = XtWindow ( da_color );

    gc = XCreateGC ( Disp, Win, 0, 0 );

/*
 *  Call the color manager to cause the color for the first overlay color to
 *  be highlighted.
 */

    cb_color_manager ( (Widget)NULL, (caddr_t)0, (caddr_t)NULL );

/*
 *  Loop and process events until the global flag (flag) is set to a value
 *  other than 01. This will occur when the user selects the OK or CANCEL
 *  command.
 */

    Flag = -1;
    while ( Flag == -1 ) {
        XtNextEvent     ( &event );
        XtDispatchEvent ( &event );
    };

    XtDestroyWidget ( Shell );

/*
 *  If the user selected CANCEL (Flag = 0), then the color map needs to needs
 *  to be restored to its pre-edit state.
 */

    if ( Flag == 0 )
        XStoreColors ( Disp, Main_cmap,
                &Cmap_orig[NUM_MOTIF_COLORS], MAX_COLORS-NUM_MOTIF_COLORS );

/*
 *  Normal return.
 */

    D(printf("END EDIT_COLORS\n"));
    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: cb_color_manager
 *
 *  This function processes the majority of the callbacks occurring in the
 *  color manager. The values of (closure), the callbacks, and the actions
 *  taken include:
 *
 *      o   ( -1 ) Called because a scale was moved. This changes the RGB
 *          contents of the current color.
 *
 *      o   ( -2 ) Called because the OK button was selected. Set global flag
 *          to cause the main loop to terminate and then save changes.
 *
 *      o   ( -3 ) Called because the CANCEL button was selected. Set global
 *          flag to cause the main loop to terminate and abort changes.
 *
 *      o   ( -4 ) Called because the RESTORE button was selected. This
 *          resets the color map to its entry state.
 *****************************************************************************/

/* ARGSUSED */
static XtCallbackProc cb_color_manager ( widget, closure, calldata )

    Widget      widget;         /* Set to widget which in which callback originated.    */

    caddr_t     closure,        /* Indicates selected command.                          */
                *calldata;      /* Widget-specific information.                         */
{
    register int    i;

    int             value;

    D(printf("START CB_COLOR_MANAGER\n"));
/*
 *  If called to initialize the current color, select it and set scales.
 */

    if ( (int)closure >= 0 ) {
        Cur_pixel = (int)closure;
        color_select ( (int)Cmap_cur[Cur_pixel].pixel );
        color_scales ( );

/*
 *  If called because a color scale was adjusted, update the RGB
 *  contents of the current color. Retrieve the positions of each of the scales
 *  and use to update the current color. Note that the position is shifted
 *  left 8 bits. When the color is ready, store it in the actual X color
 *  map to cause the effect to be seen. Note that the RGB values in the
 *  XColor structure are shorts and the return value from (XmScaleGetValue) is
 *  an int.
 */

    } else if ( (int)closure == -1 ) {
        XmScaleGetValue ( Sc_rgb_red,   &value );
        Cmap_cur[Cur_pixel].red   = value << 8;
        XmScaleGetValue ( Sc_rgb_green, &value );
        Cmap_cur[Cur_pixel].green = value << 8;
        XmScaleGetValue ( Sc_rgb_blue,  &value );
        Cmap_cur[Cur_pixel].blue  = value << 8;

        XStoreColor ( Disp, Main_cmap, &Cmap_cur[Cur_pixel] );

/*
```

```
 *  If called because the user selected OK or CANCEL, set the global flag to
 *  indicate that the main loop should terminate. The value assigned to (Flag)
 *  indicates whether or not changes should be saved.
 */

    } else if ( (int)closure == -2 ) {
        Flag = 1;

    } else if ( (int)closure == -3 ) {
        Flag = 0;

/*
 *  If called because the RESTORE command was selected, restore the color mapo
 *  to the original colors. First restore the actual X color map (don't send
 *  the first few colors because they can't be changed). Next restore each of
 *  the colors in the current color map.
 */

    } else if ( (int)closure == -4 ) {
        XStoreColors ( Disp, Main_cmap,
                &Cmap_orig[NUM_MOTIF_COLORS], MAX_COLORS-NUM_MOTIF_COLORS );
        for ( i = 0; i < Num_colors; i++ )
            memcpy ( (char *)&Cmap_cur[i],
                     (char *)&Cmap_orig[Cmap_cur[i].pixel], sizeof(XColor) );

/*
 *      Update the scales to reflect the (possibly) changed current color
 *      value.
 */

        color_scales ( );
    }

/*
 *  Normal return.
 */

    D(printf("END CB_COLOR_MANAGER\n"));
    return;
}
```

```
/*****************************************************************************
 * MODULE NAME: cb_event_color
 *
 *  This function is called to handle events in the available colors window.
 *****************************************************************************/

/* ARGSUSED */
static XtCallbackProc cb_event_color ( widget, closure, calldata )

    Widget      widget;         /* Set to widget which in which callback originated.    */

    caddr_t     closure;        /* Not used.                                            */

    XmDrawingAreaCallbackStruct
                *calldata;      /* Widget-specific data.                                */
{
    register int    color;

    XButtonEvent    *button;

    D(printf("START CB_EVENT_COLOR\n"));
/*
 *  If the event was a button press, return. The only events processed are
 *  button releases.
 */

    button = &calldata->event->xbutton;

    if ( button->state == 0 )
        return;

/*
 *  Extract the x and y position from the event structure and use to compute
 *  the corresponding color.
 */

    color = ( button->y / (CLR_SIZE+CLR_SPACE) ) * CLR_NUM +
            ( button->x / (CLR_SIZE+CLR_SPACE) );

/*
 *  If the x and y position is outside of the valid range (indicated by the
 *  color), return.
 */

    if ( color > CLR_NUM*CLR_NUM || color < NUM_MOTIF_COLORS )
        return;

/*
 *  Indicate the newly selected color (draw an outline around the selected
 *  color).
 */

    color_select ( (int)color );

/*
 *  Update the current colormap entry to the selected color. Update the color
 *  scales to represent the new color.
 */

    Cmap_cur[Cur_pixel].pixel = color;
    XQueryColor ( Disp, Main_cmap, &Cmap_cur[Cur_pixel] );

    color_scales ( );
```

```
/*
 *  Normal return.
 */

    D(printf("END CB_EVENT_COLOR\n"));
    return;
}
```

```
/*****************************************************************************
 * MODULE NAME: cb_expose_color
 *
 *  This function is called when the color table window is exposed. It draws
 *  a grid consisting of the available colors. It only handles a color map
 *  containing 256 colors (grid dimensions of 16 x 16).
 *****************************************************************************/

/* ARGSUSED */
static XtCallbackProc cb_expose_color ( widget, closure, calldata )

    Widget  widget;             /* Set to the widget which initiated this
                                 * callback function.
                                 */

    caddr_t closure;            /* Callback specific data.  This parameter
                                 * is not used by this function.
                                 */
    XmDrawingAreaCallbackStruct *calldata;
                                /* Specifies any callback-specific data the
                                 * widget needs to pass to the client.
                                 */
{
    register int        x, y;

    XExposeEvent        *expose;

    D(printf("START CB_EXPOSE_COLOR\n"));
/*
 *  Save pointer to the expose event.  If another expose event is pending,
 *  exit from function.
 */

    expose = &calldata->event->xexpose;

    if ( expose->count != 0 )
        return;

/*
 *  Draw black background throughout the whole window to give a background
 *  for the actual colors.
 */

    XSetForeground ( Disp, gc, BlackPixel ( Disp, Scr ) );
    XFillRectangle ( Disp, Win, gc, 0, 0, CLR_TOTAL+1, CLR_TOTAL+1 );

/*
 *  Draw a square for each of the available colors. This algorithm will only
 *  work for 256 cell color maps. Note that CLR_SPACE is the amount of gap
 *  between the color squares. It is set to 3 to allow the highlight rec-
 *  tangle to contrast with both the background and the color itself.
 */

    for ( y = 0; y < CLR_NUM; y++ )
        for ( x = 0; x < CLR_NUM; x++ ) {
            XSetForeground ( Disp, gc, CLR_NUM * y + x );
            XFillRectangle ( Disp, Win, gc,
                             CLR_SPACE + x * ( CLR_SIZE + CLR_SPACE ),
                             CLR_SPACE + y * ( CLR_SIZE + CLR_SPACE ),
                             CLR_NUM - 1, CLR_NUM - 1 );
        }

/*
```

```
     * Draw the highlight rectangle around the current color.
     */

        color_select ( (int)Cmap_cur[Cur_pixel].pixel );

    /*
     *  Normal return.
     */

        D(printf("END CB_EXPOSE_COLOR\n"));
        return;
    }
```

```
/*****************************************************************************
 * MODULE NAME: color_scales
 *
 *  This function updates the scale widgets when a new color is selected
 *  either via the color map.
 *****************************************************************************/

static int color_scales ( )
{
    D(printf("START COLOR_SCALES\n"));
/*
 *  Set the position of the three scales to correspond to the color. Note that
 *  the RGB values are in the range of 0 to 65535, so they must be shifted
 *  right (divided by 256) first. This is correct because the LSB of the
 *  RGB value is always zero.
 */

    XmScaleSetValue ( Sc_rgb_red,   Cmap_cur[Cur_pixel].red   >> 8 );
    XmScaleSetValue ( Sc_rgb_green, Cmap_cur[Cur_pixel].green >> 8 );
    XmScaleSetValue ( Sc_rgb_blue,  Cmap_cur[Cur_pixel].blue  >> 8 );

/*
 *  Normal return.
 */

    D(printf("END COLOR_SCALES\n"));
    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: color_select
 *
 *  This function highlights the rectangle containing the current color in the
 *  RGB color map area.
 *****************************************************************************/

static int color_select ( color )

    int     color;              /* Specifies the color (and the area) to
                                 * highlight.
                                 */
{
    static int      cur_color = -1;

    D(printf("START COLOR_SELECT\n"));
/*
 *  If an old highlight rectangle is displayed, remove it by redrawing it in
 *  the black pixel. Note that there are always three spaces between each of
 *  the color rectangles. The highlight color goes in the middle pixel of the
 *  three so that it never touches the actual color.
 */

    if ( cur_color != -1 ) {
        XSetForeground ( Disp, gc, BlackPixel ( Disp, Scr ) );
        XDrawRectangle ( Disp, Win, gc,
                    CLR_SPACE+( cur_color % CLR_NUM )*(CLR_SIZE+CLR_SPACE) - 2,
                    CLR_SPACE+( cur_color / CLR_NUM )*(CLR_SIZE+CLR_SPACE) - 2,
                    CLR_NUM+2, CLR_NUM+2 );
    }

/*
 *  Draw the new rectangle around the current color.
 */

    cur_color = color;
    XSetForeground ( Disp, gc, WhitePixel ( Disp, Scr ) );
    XDrawRectangle ( Disp, Win, gc,
                     CLR_SPACE+( color % CLR_NUM )*(CLR_SIZE+CLR_SPACE) - 2,
                     CLR_SPACE+( color / CLR_NUM )*(CLR_SIZE+CLR_SPACE) - 2,
                     CLR_NUM+2, CLR_NUM+2 );

/*
 *  Normal return.
 */

    D(printf("END COLOR_SELECT\n"));
    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: ex_msgsnd.c
 *
 *  This function uses WEX to send a message to another process.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  S. Zrubek       - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <constants.h>
#include <pf_key.h>
#include <wex/EXmsg.h>


int ex_msgsnd ( cmd_ptr )

    struct pfkey_defs   *cmd_ptr;
{
    struct msgbuf       *message;       /* Structure to send to tui_msgsnd.          */

    char                *process_name,  /* Name of the process to send the message to.*/
                        *temp_mesg;     /* Temporary message variable.               */

    int                 i = 0,          /* Loop counter.                             */
                        msg_lnth,       /* Length of the message to send.            */
                        pid,            /* Process id.                               */
                        rc = 0;         /* Error code.                               */

    D(printf("START ex_msgsnd\n"));
/*
 *  Set up initial values
 */

    msg_lnth = strlen ( cmd_ptr->mesg_ptr );
    message = (struct msgbuf *)calloc ( (unsigned)1, sizeof ( struct msgbuf ) );

/*
 *  Parse out the name of the process to which the message is being sent
 */

    while ( cmd_ptr->mesg_ptr[i] != ' ' ) {
        if ( i <= MAXFILENAMESIZ )
            i++;
        else
            break;
    }

    process_name = ( char * ) calloc ( (unsigned)1, i );
    strncpy ( process_name, cmd_ptr->mesg_ptr, i );
```

```
/*
 *  Find the start of the actual message and parse it out of the input string
 */

    i++;
    msg_lnth -= i;
    temp_mesg = ( char * ) calloc ( (unsigned)1, msg_lnth + 1 );
    temp_mesg = & ( cmd_ptr->mesg_ptr[i] );
    temp_mesg[msg_lnth] = '\0';

/*
 *  Get the process ID of the process to which the message is being sent
 */

    pid = EXgetpid ( process_name, SEARCH_ALL );

/*
 *  Set flags and send the message
 */

    message->mtype = 1;
    strncpy ( message->mtext, temp_mesg, strlen ( temp_mesg ) );

    if ( pid > 0 ) {
        rc = EXmsgsnd ( pid, message, msg_lnth, IPC_NOWAIT );
        if ( rc < 0 )
            tui_msg ( M_YELLOW, "ERROR %d in tui_msgsnd function", rc );
    } else
        tui_msg ( M_YELLOW, "ERROR Process %s not found for tui_msgsnd", process_name );

    D(printf("END ex_msgsnd\n"));
    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: exit_disp.c
 *
 *  This routine removes the display and frees memory via a call to
 *  clear.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Tod Milam - Ford Aerospace Corporation/Houston
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Ronnie Killough - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <X11/Xlib.h>
#include <X11/Intrinsic.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <constants.h>
#include <disp.h>
#include <DDdisp.h>
#include <wex/EXmsg.h>


extern struct dm_shmemory   *Dm_Address;     /* ptr to DM shared memory          */


exit_disp(disp_num)

    short   disp_num;                        /* display # of display to remove   */
{

    D(printf("START exit_disp\n"));
/*
 *  Verify this display is active
 */

    if (Dm_Address->display[disp_num].disp_active == NO) {
        tui_msg(M_YELLOW, "Display number %d not active", disp_num);
        return(0);
    }

/*
 *  Clear display and free memory
 */

    clear(disp_num);

/*
 *  Deactivate this display number
 */

    Dm_Address->display[disp_num].disp_active = NO;

/*
```

```
     *  Destroy display window. MDC - zero shell.
     */

        XtDestroyWidget(Dm_Address->shell[disp_num]);
        Dm_Address->shell[disp_num] = NULL;

    /*
     *  Set exit response flag, negate halt flag, negate display init flag
     */

        Dm_Address->display[disp_num].halt          = NO;
        Dm_Address->display[disp_num].disp_init     = NO;

        D(printf("END exit_disp\n"));
        return(0);
    }
```

```
/*****************************************************************************
 * MODULE NAME: extract.c
 *
 *  This function extracts a data value from the data buffer using the
 *  decom buffer information.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Tod Milam - Ford Aerospace Corporation/Houston
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Ronnie Killough - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <X11/Xlib.h>
#include <wex/EXmsg.h>
#include <sys/types.h>
#include <constants.h>
#include <disp.h>
#include <DDdisp.h>


extern union p_data Data;       /* union structure for decom'd data values  */
extern char     Cdata[256];     /* buffer for character data from the DH    */


extract(data_ptr, decom_ptr)

    char   *data_ptr;               /* ptr to the raw data buffer           */
    struct shm_decom    *decom_ptr; /* ptr to the decom buffer              */
{
    long    status;                 /* status word for incoming data value  */

    int     count;                  /* count of number of char data bytes   */

/*
 * Extract the status word first
 */

    status = *(long*)(data_ptr);
    Data.ldata[0] = *(long*)(data_ptr + 4);
    Data.ldata[1] = *(long*)(data_ptr + 8);

/*
 *  Extract data from raw data buffer into union structure
 *  according to the decom attribute.
 */

    switch ( decom_ptr->attribute ) {

        case ( 'D' ) :      /*   double precision real                  */
        case ( 11 ) :       /*   binary coded decimal time variable     */
        case ( 13 ) :       /*   binary coded hexadecimal time variable */
        case ( 15 ) :       /*   bit weighted time variable             */

            Data.uldata[0] = *(long*)(data_ptr + 4);
```

```
        Data.uldata[1] = *(long*)(data_ptr + 8);
        break;

    case ( 'E' ) :      /*   single precision real                    */

        Data.sfdata[0] = *(float*)(data_ptr + 4);
        break;

    case ( 9 ) :        /*   binary coded decimal tacan range         */
    case ( 12 ) :       /*   binary coded decimal analog varable      */
    case ( 14 ) :       /*   binary coded hexadecimal analog var.     */
    case ( 16 ) :       /*   bit weighted analog variable             */

        Data.uldata[0] = *(long*)(data_ptr + 4);
        break;

    case ( 'F' ) :      /*   integer - signed                         */
    case ( 2 ) :        /*   integer - signed                         */
    case ( 3 ) :        /*   integer - no compliment                  */
    case ( 4 ) :        /*   integer - no compliment - overflow bit   */

        if ( decom_ptr->length == 16 )
            Data.ssdata[0] = *(short*)(data_ptr + 4);
        else
            Data.sldata[0] = *(long*)(data_ptr + 4);
        break;

    case ( 'B' ) :      /*   Discrete                                 */
    case ( 24 ) :       /*   Discrete                                 */

        Data.sldata[0] = *(long*)(data_ptr);
        Data.sldata[0] &= 01;
        break;

    case ( 'P' ) :      /*   Discrete Parent                          */
    case ( 'L' ) :      /*   Natural - Unsigned                       */
    case ( 5 ) :        /*   Natural - Unsigned                       */
    case ( 6 ) :        /*   Discrete Parent                          */

        if ( decom_ptr->length <= 32 )
            Data.uldata[0] = *(unsigned long*)(data_ptr + 4);
        else {
            Data.uldata[0] = *(unsigned long*)(data_ptr + 4);
            Data.uldata[1] = *(unsigned long*) (data_ptr + 8);
        }
        break;

    case ( 'A' ) :      /*   ASCII                                    */
    case ( 23 ) :       /*   ASCII character string                   */

        for ( count = 0; count < decom_ptr->length; count++ )
            Cdata[count] = (char)*(data_ptr + 4 + count);
        break;

    case ( 1 ) :        /*   Real                                     */

        if ( decom_ptr->length == 16 )
            Data.ssdata[0] = *(short*)(data_ptr + 4);

        else if ( decom_ptr->length == 32 )
            Data.sldata[0] = *(long*)(data_ptr + 4);

        else {
            Data.sldata[0] = *(long*)(data_ptr + 4);
```

```
            Data.sldata[1] = *(long*)(data_ptr + 8);
        }

        break;

    case ( 7 ) :        /*   binary coded decimal - FORMAT X    */
    case ( 8 ) :        /*   binary coded decimal - FORMAT Y    */
    case ( 10 ) :       /*   binary coded decimal GMT - days/hrs */

        Data.usdata[0] = *(long*)(data_ptr + 4);
        break;

    case ( 17 ) :       /*   bit weighted clock time            */

        Data.uldata[0] = ( *(short*)(data_ptr + 4)*30 );
        Data.uldata[1] = *(long *)(data_ptr + 8);
        break;

    case ( 18 ) :       /*   bit weighted GMT/MET               */

        Data.uldata[0] = *(unsigned long*)(data_ptr + 4);
        Data.uldata[1] = *(unsigned long*)(data_ptr + 8);
        break;

    case ( 19 ) :       /*   spacelab floating point            */

        if ( decom_ptr->length <= 32 )
            Data.uldata[0] = *(long*)(data_ptr + 4);
        else {
            Data.uldata[0] = *(short*)(data_ptr + 4);
            Data.uldata[1] = *(long*)(data_ptr + 8);
        }
        break;

    case ( 20 ) :       /*   experiment I/O GMT - TYPE X        */
    case ( 21 ) :       /*   experiment I/O GMT - TYPE H        */
    case ( 22 ) :       /*   EBCDIC character string            */

        Data.uldata[0] = *(short*)(data_ptr + 4);
        Data.uldata[1] = *(long*)(data_ptr + 8);
        break;

    default:

        Data.ddata = *(double*)(data_ptr + 4);
        break;

}

return (status);
}
```

```
/*****************************************************************************
 * MODULE NAME: first_proc.c
 *
 *  This routines attempts to attach to the Display Manager Shared Memory.
 *  If memory does not exist, then this is the first Display Manager for this
 *  workstation.  If the memory does exist, then a system call is made to obtain
 *  the number of processes attached to the Display Manager shared memory.  If
 *  no processes are attached, then this is the first Display Manager for this
 *  workstation.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  K. Noonan       - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <fcntl.h>
#include <stdio.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <constants.h>
#include <disp.h>
#include <wex/EXmsg.h>


extern struct dm_shmemory   *Dm_Address;
struct shmid_ds             buf;        /* shared memory status information       */

extern int                  errno,      /* error return value                     */
                            Dm_Id;      /* shared memory Id                       */


int first_proc ( )
{
    short       first_copy = NO;        /* flag signifying first Display Manager  */

    int         retval,                 /* system error return value              */
                shmctl ( );             /* shm status call                        */

    D(printf("START first_proc\n"));
/*
 *  Determine if this is the first Display Manager activated. Get the shm Id.
 *  If an error occurs, then this is the first process.  Otherwise do a shmctl
 *  call to retrieve the number of attaches to the shared memory.  If the
 *  number attached is greater than zero, then this is not the first Display
 *  Manager in the workstation.  If the number attached is zero, then this
 *  is the first Display Manager, but the shared memory exists, so don't
 *  recreate the shared memory.  Attach to the shared memory in the case of
 *  the existence of the shared memory.
 */

    if ( ( Dm_Id = shmget ( DM_SHM_KEY, sizeof(struct dm_shmemory), 0666 ) ) == -1 ) {
        first_copy = CREAT_SHM;
```

```
    } else {
        retval = shmctl ( Dm_Id, IPC_STAT, &buf );
        if ( retval == -1 ) {
            tui_msg ( M_YELLOW, "Error %d on shmctl call", errno );
            return ( -1 );
        } else {
            if ( buf.shm_nattch > 0 )
                first_copy = NO;
            else
                first_copy = NO_CREAT;
            if ( ( Dm_Address = ( struct dm_shmemory * ) shmat ( Dm_Id, 0, 0 ) ) == NULL )
{
                tui_msg ( M_YELLOW, "Error %d on shared memory attach", errno );
                return ( -1 );
            }
        }
    }

    D(printf("END first_proc\n"));
    return ( first_copy );
}
```

```
/*****************************************************************************
 * MODULE NAME: flt_data.c
 *
 *  This function allows the user to set the flight and data type.
 *
 *
 * INTERNAL FUNCTIONS:
 *
 *      o   flt_data_menu   -   This function displays the menu which allows
 *                              the user to enter the flight and data type.
 *
 *      o   cb_fd           -   This function processes all callback generated
 *                              by the menu.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  K. Noonan - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <X11/Intrinsic.h>
#include <X11/Shell.h>
#include <Xm/Xm.h>
#include <Xm/Text.h>
#include <constants.h>
#include <disp.h>
#include <pf_key.h>
#include <user_inter.h>
#include <wex/EXmsg.h>


static Widget               t_fid, r_data;
static int                  flag;

extern Widget               Top;

extern struct dm_shmemory   *Dm_Address;    /* Display Manager shared memory      */

extern short                Flt_Selected,   /* Yes if flt info has been input     */
                            Disp_Num,       /* display number of Display Manager  */
                            Good_Strm;      /* Yes, if good data type input       */


int flt_data ( )
{
    D(printf("START flt_data\n"));
/*
 *  If a flight has already been selected, deselect it.
 */

    if ( Flt_Selected == YES )
        chk_flt ( NO );

/*
 *  Display and get input from a popup.
```

```
*/

    flt_data_menu ( );

    D(printf("END flt_data\n"));
    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: flt_data_menu
 *
 *  This function displays the menu which allows the user to enter the flight
 *  and data type.
 *****************************************************************************/

static int flt_data_menu ( )
{
    register int    i;

    static char     *datatypes[] = { "RR", "R1", "R2", "SR", "S1", "S2" };

    Arg             args[10];

    Widget          shell, form, f_data, f_cmd;

    XtCallbackProc  cb_fd();

    XEvent          event;

    D(printf("START flt_data_menu\n"));
/*
 *  Create the shell widget. Note that setting the args in the create
 *  widget call does not seem to work, so I set them afterward.
 */

    i = 0;
    shell = tui_create_trans_shell ( "Set Flight/Data", args, i );

/*
 *  Create the main form.
 */

    i = 0;
    form   = tui_create_form ( shell, "form",   TRUE,  args, i );
    f_data = tui_create_form ( form,  "f_data", FALSE, args, i );
    f_cmd  = tui_create_form ( form,  "f_cmd",  FALSE, args, i );

/*
 *  Create all widgets.
 */

    i = 0;
    tui_create_label ( f_data, "l_fid",  "Flight ID", args, i );
    tui_create_label ( f_data, "l_data", "Data Type", args, i );
    t_fid  = tui_create_text  ( f_data, "t_fid",  Dm_Address->display[Disp_Num].flight_id,
                                3, XmSINGLE_LINE_EDIT, TRUE, args, i );
    r_data = tui_create_rb  ( f_data, "r_data", datatypes, 6,
                              Dm_Address->display[Disp_Num].strm_type, args, i );

    i = 0;
    XtManageChild ( XmCreateSeparator ( form, "sep0", args, i ) );

    i = 0;
    tui_create_pushbutton ( f_cmd, "Cancel", cb_fd, (caddr_t)0, args, i );
    tui_create_pushbutton ( f_cmd, "OK",     cb_fd, (caddr_t)1, args, i );
    tui_create_pushbutton ( f_cmd, "Help",   cb_fd, (caddr_t)2, args, i );

/*
 *  Put all inputs in a tab group.
 */
```

```
    XmAddTabGroup ( t_fid  );
    XmAddTabGroup ( r_data );

/*
 *  Realize and popup the shell.
 */

    XtRealizeWidget ( shell );
    XtPopup ( shell, None );
    set_cmap ( shell );

/*
 *  Wait until the user finishes with the popup.
 */

    flag = -1;
    while ( flag == -1 ) {
        XtNextEvent     ( &event );
        XtDispatchEvent ( &event );
    }

    XtDestroyWidget ( shell );

/*
 *  Return the value selected by the user (0 is for not verified, 1 is for
 *  verified.
 */

    D(printf("END flt_data_menu\n"));
    return ( flag );
}
```

```
/*****************************************************************************
 * MODULE NAME: cb_fd
 *
 *  This function processes all callback generated by the menu.
 *****************************************************************************/

/* ARGSUSED */
static XtCallbackProc cb_fd ( w, closure, calldata )

    Widget          w;          /* Set to widget which in which callback originated.    */

    caddr_t         closure,    /* Indicates selected command.                          */
                    *calldata;  /* Widget-specific information.                         */
{
    XtCallbackProc  cb_help();

    int             val_dt();

    char            *flight_id,
                    *strm_type;

    D(printf("START cb_fd\n"));
/*
 *  Process OK button.
 */

    if ( (int)closure == 1 ) {
        flight_id = XmTextGetString ( t_fid  );
        strm_type = tui_radio_get_value ( r_data );

/*
 *      Copy the data type into shared memory. Save and validate the data type.
 */

        strncpy ( Dm_Address->display[Disp_Num].flight_id, flight_id, 3 );
        Dm_Address->display[Disp_Num].flight_id[3] = 0;

        val_dt ( strm_type );

/*
 *      If the flight or the data type is invalid, set flag. Otherwise, select the flight
 *      if valid.
 */

        if ( ( Dm_Address->display[Disp_Num].flight_id[0] == '\0' ) ||
             ( Dm_Address->display[Disp_Num].flight_id[0] == ' ' )  ||
             ( Good_Strm == NO ) ) {
            Flt_Selected = NO;
        } else {
            chk_flt ( YES );
            flag = (int)closure;
        }

/*
 *  Process CANCEL button.
 */

    } else if ( (int)closure == 0 ) {
        flag = (int)closure;

/*
 *  If help button was selected, display appropriate help text.
 */
```

```
    } else if ( (int)closure == 2 )
        cb_help ( (Widget)0, (caddr_t)1, (caddr_t)0 );

    D(printf("END cb_fd\n"));
    return;
}
```

```
/*****************************************************************************
 * MODULE NAME: font_num.c
 *
 *  This function returns the X Font ID corresponding to the given
 *  dimensions and style of the font.  Any font needed which has
 *  not been loaded is loaded.  The dimensions and style of the font
 *  those generated by the Display Builder application and are translated
 *  into fixed font sizes in ranges.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Richard Romeo   - Ford Aerospace Corporation/Houston
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Ronnie Killough - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <X11/Xlib.h>
#include <constants.h>
#include <disp.h>
#include <DDdisp.h>


extern struct dm_shmemory   *Dm_Address;    /* ptr to DM shared memory      */


Font font_num(disp_num, font_style, horz_size, vert_size)

    short   disp_num;           /* display # of requested font              */
    char    font_style[5];      /* requested font style (regl, bold, ital)  */
    int     horz_size;          /* horizontal font size                     */
    int     vert_size;          /* vertical font size                       */
{

    Display *xdisplay;          /* ptr to X display struct of display       */
    Font    font_nm;            /* X Font number                            */


    xdisplay = Dm_Address->xdisplay[disp_num];

/*
 *  Break fonts down by style.
 */

    switch (font_style[0]) {

    case 'r':       /* Font Style is regular   */

/* RLK original range value for 9x11 font
        if ((horz_size >= 95) || (vert_size >= 220))
*/

        if ((horz_size >= 155) || (vert_size >= 245))
            font_nm = XLoadFont(xdisplay, R_24);
        else if ((horz_size >= 130) || (vert_size >= 220))
            font_nm = XLoadFont(xdisplay, R_18);
```

```
        else if ((horz_size >= 105) || (vert_size >= 195))
            font_nm = XLoadFont(xdisplay, R_14);
        else if ((horz_size >= 80) || (vert_size >= 170))
            font_nm = XLoadFont(xdisplay, R_12);
        else if ((horz_size >= 55) || (vert_size >= 145))
            font_nm = XLoadFont(xdisplay, R_10);
        else
            font_nm = XLoadFont(xdisplay, R_08);

        break;

    case 'b':       /* Font Style is bold      */

        if ((horz_size >= 155) || (vert_size >= 245))
            font_nm = XLoadFont(xdisplay, B_24);
        else if ((horz_size >= 130) || (vert_size >= 220))
            font_nm = XLoadFont(xdisplay, B_18);
        else if ((horz_size >= 105) || (vert_size >= 195))
            font_nm = XLoadFont(xdisplay, B_14);
        else if ((horz_size >= 80) || (vert_size >= 170))
            font_nm = XLoadFont(xdisplay, B_12);
        else if ((horz_size >= 55) || (vert_size >= 145))
            font_nm = XLoadFont(xdisplay, B_10);
        else
            font_nm = XLoadFont(xdisplay, B_08);

        break;

    case 'i':       /* Font Style is italic    */

        if ((horz_size >= 155) || (vert_size >= 245))
            font_nm = XLoadFont(xdisplay, I_24);
        else if ((horz_size >= 130) || (vert_size >= 220))
            font_nm = XLoadFont(xdisplay, I_18);
        else if ((horz_size >= 105) || (vert_size >= 195))
            font_nm = XLoadFont(xdisplay, I_14);
        else if ((horz_size >= 80) || (vert_size >= 170))
            font_nm = XLoadFont(xdisplay, I_12);
        else if ((horz_size >= 55) || (vert_size >= 145))
            font_nm = XLoadFont(xdisplay, I_10);
        else
            font_nm = XLoadFont(xdisplay, I_08);

        break;

    default:
        font_nm = XLoadFont(xdisplay, R_12);
        break;
    }
    return (font_nm);
}
```

```
/*****************************************************************************
 * MODULE NAME: gdr_next.c
 *
 *  This function processes the GDR next command.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  R. Romeo        - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <constants.h>
#include <disp.h>
#include <pf_key.h>
#include <wex/EXmsg.h>


extern struct dm_shmemory *Dm_Address;      /* ptr to display manager shm      */
extern struct pfkey_defs Current_Com;       /* current command structure       */

extern char     Disp_Path[DNAME_LEN];       /* default display path name       */


int gdr_next ( )
{
    short           i,                      /* index counter                   */
                    match = NO;             /* YES, if match found on filename */

    char            ppl_fn[60];             /* ppl filename                    */

    D(printf("START gdr_next\n"));
/*
 *  Search for a match on the PPL filename that the get next command is to be
 *  done for.  If a match is found, then set the get next command flag in the
 *  PPL record information and in shared memory for the Data Handler.   If no
 *  match is found, then advise.
 */

    if ( Current_Com.disp_name[0] != '/' ) {
        strcpy ( ppl_fn, Disp_Path );
        strcat ( ppl_fn, Current_Com.disp_name );
    } else
        strcpy ( ppl_fn, Current_Com.disp_name );

    i = 0;
    while ( i < NUM_GDR && match == NO ) {
        if ( strcmp ( ppl_fn, Dm_Address->ppl_recs[i].ppl_filename ) == 0 ) {
            match = YES;
            Dm_Address->ppl_recs[i].get_next = YES;
            Dm_Address->process.gdr_get_next = YES;
        } else
            i++;
    }

    if ( match == NO )
```

```
            tui_msg ( M_YELLOW, "PPL file %s is not currently active", Current_Com.disp_name )
    ;

        D(printf("END gdr_next\n"));
        return ( 0 );
    }
```

```
/*******************************************************************************
 * MODULE NAME: get_disp.c
 *
 *  This function prompts for a display name.
 *
 *
 * INTERNAL FUNCTIONS:
 *
 *      o   process_ok  -   Returns TRUE if the user enters a valid display
 *                          name.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  C. Davis         - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *******************************************************************************/


#include <X11/Intrinsic.h>
#include <Xm/Text.h>
#include <constants.h>
#include <disp.h>
#include <pf_key.h>
#include <user_inter.h>
#include <wex/EXmsg.h>


extern Widget               Top;

extern struct pfkey_defs    Current_Com;    /* current commands definition        */
extern struct dm_shmemory   *Dm_Address;    /* Display Manager shared memory ptr  */

extern short                Disp_Num;       /* Display Manager number             */


int get_disp ( )
{
    int         process_ok();

    short       flag;

    D(printf("START get_disp\n"));
/*
 *  Display a popup waiting for user to enter a display name.
 */

    flag = tui_get_prompt ( Top, "Get Display", "Display Name", Current_Com.disp_name, -1,
                            process_ok, NULL, 0 );

/*
 *  Return the status of the popup.
 */

    D(printf("END get_disp\n"));
    return ( flag );
}
```

```
/*****************************************************************************
 * MODULE NAME: process_ok
 *
 *  Returns TRUE if the user enters a valid display name.
 *****************************************************************************/

static int process_ok ( string )

    char            *string;
{

    D(printf("START process_ok\n"));
/*
 *  Process OK button.
 */

    strcpy ( Current_Com.disp_name, string );
    if ( val_fn ( string, YES ) ) {
        Current_Com.func_no = START_PDISPLAY;
        return ( 1 );
    }

    D(printf("END process_ok\n"));
    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: get_fn.c
 *
 *  The Get Filename function strips off the directory from a filename and
 *  returns the filename without the directory specified.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  C. Davis        - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <constants.h>
#include <wex/EXmsg.h>


int get_fn ( dir_file_name, no_dir_fn )

    char            *dir_file_name,  /* pointer to the dir. spec. filename  */

                     *no_dir_fn;     /* ptr to the no directory filename    */
{
    int              len;            /* length of the filename              */

    short            i;              /* index cntr                           */

    D(printf("START get_fn\n"));
/*
 * Get the length of the passed in filename.  Then search for the first
 * occurrance of a "/."   Copy the characters from the slash to the end of the
 * filename into the variable to be passed back.  This variable will contain
 * the filename without the directory.
 */

    len = strlen ( dir_file_name );
    i = len - 1;
    while ( ( i >= 0 ) && ( ( * ( dir_file_name + i ) != '/' ) ) ) {
        i--;
    }
    strcpy ( no_dir_fn, ( dir_file_name + i + 1 ) );

    D(printf("END get_fn\n"));
    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: get_plot.c
 *
 *  This function is invoked when the user selects either a predefined PF
 *  key or a menu selection for a plot start or stop.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  K. Noonan       - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <fcntl.h>
#include <unistd.h>
#include <constants.h>
#include <disp.h>
#include <DDdisp.h>
#include <DDplot.h>
#include <pf_key.h>
#include <wex/EXmsg.h>


extern struct dm_shmemory   *Dm_Address;
extern struct pfkey_defs    Current_Com;
extern struct plot_ptrs     *Plot_info_ptr; /* Ptr thru plot ptr files.          */

extern int                  errno;          /* Error return value.               */

extern short                Nbr_of_plots,   /* Number of plots to display.       */
                            Disp_Num;

extern char                 Disp_Path[DNAME_LEN];


int get_plot ( )
{
    struct disp_info        *display;
    struct shm_decom        *loc_dcm_ptr;   /* Ptr thru plot decom structure.    */
    struct plot_hdr         *plot_hdr_ptr;  /* Ptr thru plot header.             */
    struct plot_ptrs        *plot_ptr;      /* Ptr thru plot ptr files.          */
    struct axis_info        *axis_ptr;      /* Local ptr to axis information.    */
    struct msid_info        *msid_ptr;      /* Local ptr to msid information.    */

    static short            access[MAX_PLOTS];  /* access restriction code for each plot*/

    int                     k, m,           /* Loop count variable.              */
                            next_offset,    /* Nbr of bytes for offset cal.      */
                            plot_fp;        /* Local plot file pointer.          */


    short                   match = NO,     /* YES, if match found on filename   */
                            i,              /* Index counter                     */
                            empty_start,    /* Slot to add start plot name       */
```

```
                        match_nbr,      /* Slot nbr for a match in act plot list    */
                        disp_num,       /* Effective display number                 */
                        nbr_axes;       /* Total x and y axes.                      */

    FILE                *fopen ( ),
                        *fp;            /* File ptr to plot file.                   */

    char                plot_name[DNAME_LEN+4], /* Plot name with .plt extension.   */
                        *malloc();      /* Get malloc as a pointer.                 */

    D(printf("START get_plot\n"));

/*
 *  Save the display number and pointer to current display structure.
 */
    disp_num = Disp_Num;
    display  = &Dm_Address->display[Disp_Num];

/*
 *  Build the complete path of the plot if the path is not already given.
 */

    if ( Current_Com.disp_name[0] != '/' ) {
        strcpy ( plot_name, Disp_Path );
        strcat ( plot_name,             Current_Com.disp_name );
        strcpy ( Current_Com.disp_name, plot_name             );
    } else
        strcpy ( plot_name,             Current_Com.disp_name );

/*
 *  Search the active plot table for a match on the plot name.  If a match is found
 *  and the action is to start a plot, then advise and return. Check for an empty slot
 *  to store the plot to be activated.
 */

    i = 0;
    match = NO;
    empty_start = -1;
    while ( ( i < MAX_PLOTS ) && ( match == NO ) ) {
        if ( ( strcmp ( Dm_Address->plots.act_plots[i], plot_name ) ) == 0 ) {
            match = YES;
            match_nbr = i;
            if ( Current_Com.func_no == PLOT ) {
                tui_msg ( M_YELLOW, "Plot %s is already active", Current_Com.disp_name );
                return ( -1 );
            }
        } else {
            if ( Dm_Address->plots.act_plots[i][0] == 0 )
                empty_start = i;
            i++;
        }
    }

/*
 *  If no match was found and the user is trying to turn a plot off, output an error
 *  and return.
 */

    if ( ( match == NO ) && ( Current_Com.func_no == PLOT_OFF ) ) {
        tui_msg ( M_YELLOW, "Plot %s not active", plot_name );
        return ( -1 );
    }

/*
```

```
 *  If the plot is to be started, then check to see if the workstation limit
 *  for active plots has been exceeded. If so, advise and return. Otherwise, open the
 *  plot file and check the access restriction code.
 */

    if ( Current_Com.func_no == PLOT ) {
        if ( empty_start == -1 ) {
            tui_msg ( M_YELLOW, "Workstation limit for active plots has been exceeded" );
            return ( -1 );
        }
        strncat ( plot_name, ".plt\0", 5 );
        fp = fopen ( plot_name, "r" );
        if ( fp == NULL ) {
            tui_msg ( M_YELLOW, "Error %d on opening plot file %s", errno, plot_name);
            return ( -1 );
        }
        fscanf ( fp, "%*72c" ); /* skip to access record */
        fscanf ( fp, "%hd", &access[i] );
        fclose ( fp );

/*
 *      Check the access restriction code to see if the plot is either a Medical
 *      or Payload restricted plot. If the plot is access restricted and the
 *      position Id does not match the access restriction, then exit out of this
 *      routine.
 */

        if ( chk_res ( access[i], display->pos_id ) )
            return ( -1 );
    }

/*
 *  Set up the shared memory flags for the Data Handler. Monitor the response
 *  from the Data Handler.
 */

    display->action = ( Current_Com.func_no == PLOT ) ? STRT_PLOT : STOP_PLOT;
    display->dh_plot_ack = NO;

    strcpy ( display->plot_name, Current_Com.disp_name );

    if ( Current_Com.func_no == PLOT ) {
        strcpy ( Dm_Address->plots.act_plots[empty_start], Current_Com.disp_name );
        display->dh_plot = STRT_PLOT;
    } else
        display->dh_plot = STOP_PLOT;

    if ( chk_flg ( &display->dh_plot_ack, 20, 1 ) ) {
        tui_msg ( M_YELLOW, "Data Handler unable to process plot %s",
                        Current_Com.disp_name );
        if ( Current_Com.func_no == PLOT )
            Dm_Address->plots.act_plots[empty_start][0] = 0;
        return ( -1 );
    }

/*
 *  If turning off a plot, zero out the name in the active plots list.
 */

    if ( Current_Com.func_no == PLOT_OFF )
        Dm_Address->plots.act_plots[match_nbr][0] = 0;

/*
 *  Find plot in plot info list.
```

```
 */

    match = NO;
    for (i=0; i < Nbr_of_plots; i++) {
        plot_ptr = Plot_info_ptr + i;

        if (!strcmp(plot_ptr->plot_name, display->plot_name)) {
            match = YES;
            break;
        }
    }

    if (match == NO) {
        tui_msg (M_YELLOW, "Plot %s is not specified in this display", display->plot_name)
;
        return ( -1 );
    }

    if ( display->action == STRT_PLOT ) {
        strcpy(plot_name, plot_ptr->plot_name);

/*
 *      Open plot data file and read decom buffer
 */

        strcpy(plot_name, plot_ptr->plot_data_file);
        strcat(plot_name, ".pdt");
        plot_ptr->plot_fp = open(plot_name, O_RDONLY);

        if (plot_ptr->plot_fp == INVALID) {
            tui_msg(M_YELLOW, "Error %d opening plot data file %s", errno, plot_name);
            return(-1);
        }

        plot_fp = plot_ptr->plot_fp;

/*
 *      Skip over decom buffer header and read
 *      decom buffer to memory.
 */

        lseek(plot_fp, 80, SEEK_SET);
        loc_dcm_ptr = plot_ptr->plt_decom;
        plot_hdr_ptr = plot_ptr->header;

        plot_ptr->buf_size = 0;
        next_offset = 0;

        for (m = 0; m < plot_hdr_ptr->msid_num; m++) {
            read(plot_fp, &loc_dcm_ptr->size, sizeof(int));
            read(plot_fp, &loc_dcm_ptr->length, sizeof(int));
            read(plot_fp, &loc_dcm_ptr->num_samps, sizeof(short));
            read(plot_fp, &loc_dcm_ptr->attribute, sizeof(char));
            read(plot_fp, &loc_dcm_ptr->error, sizeof(char));
            lseek(plot_fp, 12, SEEK_CUR);

#ifdef FAC
            if (loc_dcm_ptr->error != NULL) {
                loc_dcm_ptr->num_samps = 1;
                loc_dcm_ptr->length = 4;
            }
#endif

/*
```

```
 *          Calculate sample size and data buffer size.
 *          Store offset into memory.
 */

            loc_dcm_ptr->sample_size = loc_dcm_ptr->size / loc_dcm_ptr->num_samps;
            plot_ptr->buf_size = plot_ptr->buf_size + 2 + loc_dcm_ptr->size;

            loc_dcm_ptr->offset = next_offset;
            next_offset = loc_dcm_ptr->size + loc_dcm_ptr->offset + 2;
            loc_dcm_ptr++;
        }

/*
 *      Allocate space for data buffer
 *      and set plot active flag
 */

        plot_ptr->plot_data = malloc(plot_ptr->buf_size);

        if (plot_ptr->plot_data == NULL) {
            tui_msg(M_YELLOW, "Error %d on creating data buffer space", errno);
            return (-1);
        }

        plot_ptr->act_flg = YES;
        plot_ptr->seconds_elapsed = 0;

/*
 *      If plot was previously active, restore plot to
 *      original state (first point flags, scale factors, etc).
 */

        if (plot_ptr->prev_act_flg) {

            nbr_axes = plot_ptr->header->xaxes_num + plot_ptr->header->yaxes_num;
            axis_ptr = plot_ptr->axis;
            msid_ptr = plot_ptr->msids;

/*
 *          Set all msid first point flags
 */

            for (k=0; k<plot_ptr->header->actual_msids; k++) {
                msid_ptr->first_pt = YES;
                msid_ptr++;
            }

/*
 *          Restore original high and low scale values
 */

            for (k = 0; k < nbr_axes; k++) {
                axis_ptr->low_value = axis_ptr->org_low_val;
                axis_ptr->high_value = axis_ptr->org_high_val;
                axis_ptr++;
            }

/*
 *          Erase the plot using the coordinates of the plot
 *          and the background color. Redraw the plot, and if
 *          an overlay is applicable draw the overlay.
 */
```

```
            XClearArea(Dm_Address->xdisplay[disp_num], XtWindow(plot_ptr->draw_win),
                       0, 0, plot_ptr->plot_pos->drw_width,
                       plot_ptr->plot_pos->drw_height, False);

            draw_plt(Disp_Num, plot_ptr);

            if (plot_ptr->ovr_flg)
                draw_ovl(plot_ptr);

/*
 *      If plot was not previously active, set
 *      previously active flag since it is being
 *      activated now.
 */

        } else
            plot_ptr->prev_act_flg = YES;

    } else if (display->disp_init && plot_ptr->act_flg == YES) {
            plot_ptr->act_flg = NO;
            free(plot_ptr->plot_data);
    }

    D(printf("END get_plot\n"));
    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: globals.c
 *
 *  The globals file contains only declaration statements of variables.
 *  There is no "executable code" in the file.  The purpose of this file is
 *  for an easy look up for all globals variables.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  C. Davis        - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <sys/types.h>
#include <sys/timeb.h>
#include <termio.h>
#include <stdio.h>
#include <X11/Intrinsic.h>
#include <constants.h>
#include <disp.h>
#include <pf_key.h>
#include <DDdisp.h>
#include <DDfg_graph.h>
#include <wex/FCpbi.h>
#include <wex/EXmsg.h>

/*
 *  New globals added.
 */

Widget              Verytop,
                    Top,
                    Shell,
                    Scrl_Win,
                    Draw_Win,
                    Plt_Scrl_Win,
                    Plt_Draw_Win,
                    Pb_Alarm,
                    Pb_Pbi,
                    Pb_Log,
                    Pb_Log_A,
                    Pb_Msg,
                    Pb_Pf;

short               Pixels[128];
float               Colors[128][3];

Colormap            Main_cmap;

/*
 *  Define the strings corresponding to all functions. These are used when the
 *  command must be verified.
 */

char                *Func_Desc[] = {
```

```
    "",
    "Start Display",
    "Start Particular Display",
    "Clear Display",
    "Screen Dump",
    "Obsolete Function",
    "Halt Display",
    "Change Limits",
    "Change Limits",
    "Obsolete Function",
    "Obsolete Function",
    "Change Update Rate",
    "Enable Limit Group",
    "Enable Alarms",
    "Start Plot",
    "Plot Overlay",
    "History Tables",
    "Zoom Display",
    "Reset Zoom",
    "Set Zoom Factor",
    "Enable Display Log",
    "Disable Display Log",
    "Enable All Display Log",
    "Disable All Display Log",
    "GDR Get Next",
    "Enable PBI",
    "Disable PBI",
    "Send Message",
    "Unlatch DDD MSID",
    "Unlatch All DDD's",
    "Set Flight/Datatype",
    "Change GDR",
    "Disable Limit Group",
    "List Limits",
    "Stop Plot",
    "List Plots",
    "Save Overlay",
    "Define Universal Plot",
    "Disable Alarm",
    "Display",
    "Enable Messages",
    "Disable Messages"
};

/*
 *  Define the list of valid datatypes.
 */

char *Src_list[] = { "MTM", "GDR", "EVN", "NDM", "PPM", "PTM", "USR" };

/*
 *  Define list which will point to the current set of MSID's for DDD and limit
 *  functions.
 */

char    **Msid_list_lim,
        **Msid_list_ddd;

int     Msid_num_lim,
        Msid_num_ddd;

/*
 *  Define the lists of limits and plot files (and descriptions). These lists are
 *  used to present available limit and plot files to the user.
```

```
 */

char    **List_plot,
        **List_limit,
        **List_ht;

int     Num_plot,
        Num_limit,
        Num_ht;

/*
 *  This variable contains the function key definition for the function keys
 *  defined to correspond to this active display.
 */

struct pfkey_defs   Act_Pfkeys[PFKEY_COUNT];

/*
 *  This variable contains the current command that the Display Manager is to
 *  process.    These commands are either PF key defined or mouse entered from
 *  the main menu.
 */

struct pfkey_defs   Current_Com;

/*
 *  This variable contains the function key definition for the default function
 *  keys defined in "default.pf" for this workstation.
 */

struct pfkey_defs   Def_Pfkeys[PFKEY_COUNT];

/*
 *  This variable contains a pointer to all the display file names and
 *  descriptions within a directory.
 */

char                *Disp_Info = NULL;

/*
 *  This variable contains the slot number in the display information structure
 *  where information for Display Manager and related Display are contained.
 */

short               Disp_Num;

/*
 *  This variable contains path of the display when no path name is specified.
 */

char                Disp_Path[DNAME_LEN];

/*
 *  This variable contains the Display Manager and Data Handler Shared Address.
 */

struct data_info    *Dh_Address;
struct dm_shmemory  *Dm_Address;

/*
 *  Variable set to true when time to exit.
 */

short               Dm_Halt;
```

```
/*
 *  This variable contains the Display Manager Shared Memory ID.
 */

int                 Dm_Id;

/*
 *  This variable contains a one if this is the first Display Manager in the
 *  workstation.
 */

short               First_Copy = NO;

/*
 *  This variable contains a one if the Flight Id and Data Type have been
 *  input.
 */

short               Flt_Selected = NO;

/*
 *  This variable contains a one if the data type is good.
 */

short               Good_Strm = NO;

/*
 *  Flag used to control display of popup messages.
 */

short               Msg_Popup_Flag = NO;

/*
 *  This variable contains the default plot path for universal and plot data
 *  files.
 */

char                Plot_Path[DNAME_LEN];

/*
 *  These variables are necessary for creating and processing the PBI environ-
 *  ment.
 */

short               Pbi_Disable;        /* Pbi Disable flag to disable all Pbi input    */
short               Pbi_Hot_Ndx;        /* Pbi Index which indicates which entry is hot */
short               Pbi_Num = 0;        /* Number of Pbi entries for the current display*/
int                 Pbi_Env_Id;         /* Pbi Environment Id for interfacing with WSA  */
int                 Pbi_Toggle_Dir = 1; /* Pbi Forward/Reverse toggle dir ( def=FORW )  */
struct              pbi_def *Pbi_Def;   /* Pointer to the Pbi display definition entries*/
struct              PBI_ENTRY *Pbi_Ptr; /* Pointer to the PBI header table for WSA proc.*/
struct              PBI_TABLE *Pbi_Table;/* Ptr to the entry table entries for WSA prc  */

/*
 *  This variable is to hold the current zoom value
 */

float               Zoom_factor = 1.5;

struct limit_fil    *First_lim_ptr = NULL;
struct limit_file   *Last_lim_ptr = NULL;
int                 Screen_color;
```

```
/*
 *  Plot data file end-of-file flag...for plot redraw only
 */

short End_of_file = NO;

/*
 * History tab globals.
 */

struct hist_tab     *Htab;
struct ht_files     *Ht_files;

/*
 *  Structure templates.
 */

struct file_header  *Bg_hdr_file;       /* Ptr to background file hdr struct          */
struct file_header  *File1;             /* Ptr to background file hdr struct          */
struct rec_header   *Record;            /* Ptr to background rec. structure           */
struct graph_record *graph_rec;         /* Ptr to background graph structure          */
struct graph_record *Graph_Rec;         /* Ptr to background graph structure          */
struct graph_record *loc_graph_ptr;     /* Ptr to background graph structure          */
struct fg_file_header *Ffile;
                                        /* Ptr to foreground file header              */
struct msid_ent     *Msid;              /* Ptr to foreground msid rec struct          */
struct tabular_ent  *Tab;               /* Ptr to Tabular record structure            */
struct limit_ent    *Limit;             /* Ptr to Limit record structure              */
struct mtext_ent    *Mtext;             /* Ptr to Multilevel txt    record stru       */
struct pbi_ent      *Pbi;               /* Ptr to PBI record structure                */
struct limit_ent    *Limit_info;        /*  local ptr for limit structure             */
struct timeb        Last_Update;        /*  last ds_getparm time call                 */
union p_data        Data;               /*  local ptr to union structure              */
struct val_txt      *text_ptr;          /*  Text pointer for multilevel txt           */

struct bg_recs      Bg_Rec;             /* ptr thru all background records            */
struct fg_recs      Fg_rec;             /* ptr thru all foreground records            */
struct fgr_record   *Loc_fgr_ptr;       /* ptr to start of foreground records         */
struct label_ent    *Lab;               /* beginning ptr to label records             */
struct label_ent    *label;
struct scale_ent    *Scale;             /* beginning ptr to scale records             */
struct ddd_ent      *Ddd;               /* beginning ptr to ddd records               */
struct line_record      *line_ptr;      /* ptr thru line records                      */
struct rectangle_record *rect_ptr;      /* ptr thru rect records                      */
struct polygon_record   *poly_ptr;      /* ptr thru poly records                      */
struct circle_record    *circle_ptr;    /* ptr thru circle records                    */
struct arc_record       *arc_ptr;       /* ptr thru arc records                       */
struct ellipse_record   *ellipse_ptr;   /* ptr thru ellipse records                   */
struct ell_arc_record   *ell_arc_ptr;   /* ptr thru elliptical arc records            */
struct curve_record     *curve_ptr;     /* ptr thru curve records                     */

/*
 *  pointers to pat files.
 */

struct pat_file_header  *Pfile;         /* ptr to pat header file                     */
struct pat_msid_entry   *Pat_file;      /* ptr to pat msid file                       */

/*
 *  Information related to fonts.
 */

double              Horz_Size[MAX_FONTS];
double              Vert_Size[MAX_FONTS];
```

```
/*
 *  Variable for year to be displayed.
 */

int                 Year;
int                 Year_Cat;

/*
 *  Character array to hold the incoming ASCII string.
 */

char                Cdata[256];

/*
 *  Plot information.
 */

char                plot_name[DISP_NAME_LEN];
struct              plot_ptrs *Plot_info_ptr;
struct plot_tmplt   *Tmplt;

/*
 *  Data array used for comparison with new data.
 */

unsigned char       New_Data[60000];
unsigned char       Old_Data[60000];

unsigned char       Graph_New_Data[60000];
unsigned char       Graph_Old_Data[60000];

short               Nbr_of_plots;

short               Ovrlay_Drawn;

/*
 *  X Windows declarations.
 */

float           Font65_height = 0.65;
float           Font80_height = 0.80;
float           Font100_height = 1.00;

unsigned long   Show;
unsigned long   Mod;

int             Offset;
int             Change;
long            Status_Color;
short           Unlatch;
short           No_Change;

/*
 *  Logging variables.
 */

int             Logging_On = NO;        /* Displayer logging flag          */
int             Log_File_Id;            /* WEX Logging identifier          */
int             Log_Pid;                /* WEX Logging identifier          */

/*
 * Redraw variables.
 */
```

```
short           Resize;                 /* flag whether a graphic will resize the window
short           Redraw_flag=NO;         /* flag whether to redraw shapes or to just size*/
float           Redraw_ulx = 0.0;       /* left x for global redraw box                 */
float           Redraw_lry = 0.0;       /* lower y for global redraw box                */
float           Redraw_lrx = 0.0;       /* right x for global redraw box                */
float           Redraw_uly = 0.0;       /* upper y for global redraw box                */
```

```
/*****************************************************************************
 * MODULE NAME: hist_tab.c
 *
 *  This function builds a list of limit or plot files and allows the user to
 *  turn a plot/limit file on or off.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  D. Rice          - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <constants.h>
#include <disp.h>
#include <DDdisp.h>
#include <pf_key.h>
#include <stdio.h>
#include <wex/EXmsg.h>


extern Widget                           Top;
extern struct fg_file_header            *Ffile;
extern struct hist_tab                  *Htab;
extern struct dm_shmemory               *Dm_Address;
extern struct pfkey_defs                Current_Com;

extern short                            Disp_Num;

extern int                              Num_ht,
                                        Num_plot,
                                        Num_limit;

extern char                             **List_ht,
                                        **List_plot,
                                        **List_limit;


int hist_tab ( )
{
    register int    i,
                    j;

    int             flag;

    char            *s,
                    filename[DNAME_LEN + 1],
                    *malloc();

    D(printf("START hist_tab\n"));
/*
 *  If it is already known that this display does not have any history table entries
 *  for available limit files, generate an error and return.
 */
```

```
    if ( Num_ht == -1 ) {
        tui_msg ( M_YELLOW, "No limit files for the history tabs in display" );
        return ( -1 );
    }

/*
 *  If a new display has been initialized, generate a new list of limits.
 */

    if ( Num_ht == 0 ) {
        tui_start_wait ( );

/*
 *      Call list_files to read the directory of limit files.
 */

        list_files ( TRUE, TRUE );
        if ( Num_limit == 0 ) {
            tui_msg ( M_YELLOW, "No limit files available for history tables" );
            tui_stop_wait ( );
            return ( -1 );
        }

/*
 *      Build a list which consists of only the limit files which are referenced within
 *      the history table entries in the display file. First allocate enough memory for
 *      the list of pointers. If this fails, log an error and return.
 */

        if ( ( List_ht = (char **)malloc ( Num_limit * sizeof ( char * ) ) ) == NULL ) {
            tui_msg ( M_YELLOW, "Unable to allocate memory for limit/history table list" )
;
            tui_stop_wait ( );
            return ( -1 );
        }

/*
 *      For each filename in the list of limit files, check if it is referenced in the
 *      history tab file. If a match is found, set pointer in the new list.
 */

        for ( i = 0; i < Num_limit; i++ ) {
            for ( j = 0; j < Ffile->Htab_Num; j++ ) {
                if ( strncmp ( (Htab + j)->file_name, *(List_limit + i), 6 ) == 0 ) {
                    *(List_ht + Num_ht) = (Htab + j)->file_name;
                    printf ( "FILE: %s\n",*(List_ht + Num_ht) );
                    Num_ht++;
                    break;
                }
            }
        }

/*
 *      If no matches were found then there are no limit files available for the history
 *      table entries in the display. Generate an error and return.
 */

        tui_stop_wait ( );
        if ( Num_ht == 0 ) {
            tui_msg ( M_YELLOW, "No limit files for the history tabs in display" );
            Num_ht = -1;
            return ( -1 );
        }
```

```
    }

/*
 *  Present the list of names to the user and wait for a response.
 */

    flag = tui_get_list ( Top, List_ht, Num_ht, filename, "History Table",
                          "Limit Files", FALSE, -1, NULL, 0 );

/*
 *  If the user canceled the pop up, return.
 */

    if ( flag == 0 )
        return ( 0 );

/*
 *  Remove any trailing blanks from the filename and save in command buffer.
 */

    strncpy ( Current_Com.disp_name, filename, 8 );
    if ( s = index ( Current_Com.disp_name, ' ' ) )
        *s = '\0';
    else
        Current_Com.disp_name[8] = '\0';

/*
 *  Copy the filename into shared memory and set flag for the Data Handler.
 */

    strcpy ( Dm_Address->display[Disp_Num].display_name, Current_Com.disp_name );
    strcpy ( Dm_Address->display[Disp_Num].plot_overlay, Current_Com.disp_name );
    Dm_Address->display[Disp_Num].dh_htab = YES;

/*
 *  Normal return.
 */

    D(printf("END hist_tab\n"));
    return ( 0 );
}
```

```
/*******************************************************************************
 * MODULE NAME: ht_init.c
 *
 *  This function initializes the display's history tabs.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  T. Milam        - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *******************************************************************************/


#include <stdio.h>
#include <string.h>
#include <constants.h>
#include <disp.h>
#include <DDdisp.h>
#include <wex/EXmsg.h>


extern struct msid_ent *Msid;           /* msid structure pointer                */
extern struct hist_tab *Htab;           /* history tab structure pointer         */
extern struct tabular_ent *Tab;         /* Tabular entry table local ptr         */
extern struct dm_shmemory *Dm_Address;  /* Display Manager shared mem.           */
extern short    Disp_Num;               /* display number                        */
extern struct ht_files *Ht_files;       /* the array of file names and pointers  */


int ht_init ( entry_num, htab_num )

    long            entry_num,          /* number of msid entries                */
                    htab_num;           /* number of history tab entries         */
(
    struct hist_tab *first,             /* points to first history tab entry     */
                    *curr,              /* points to current history tab entry   */
                    *prev,              /* points to previous history tab entry  */
                    *prev_prev,         /* points to history tab two prior to curr */
                    *htab;              /* pointer to loop through htab list     */
    struct msid_ent *msid_ptr;          /* local msid pointer                    */
    struct ht_files *file_struct;       /* local pointer to file array           */
    struct tabular_ent *tab_ptr;        /* local tabular pointer                 */

    FILE            *file_ptr;          /* pointer to the history tab file       */

    short           i, j,               /* loop counters                         */
                    flag,               /* flag if prev file name is > curr      */
                    flag2,              /* flag if prev msid is > curr           */
                    version,            /* version read from the history tab file */
                    access;             /* access rest code from the hist tab file */

    long            status;             /* status variable                       */

    int             size,               /* size read from the hist tab file      */
                    length,             /* length read from the hist tab file    */
                    num_samps;          /* # of samples read from the hist tab file */
```

```
    char            flight_id[5],       /* flight id read from the hist tab file    */
                    strm_type[3],       /* stream type read from the hist tab file  */
                    file[14],           /* file name of the history tab file        */
                    ht_file_name[50],   /* local history tab file name to open      */
                    msid[MSID_LENGTH],  /* msid name read from the hist tab file    */
                    sample[5],          /* sample read from the hist tab file       */
                    source[3],          /* source read from the hist tab file       */
                    attribute,          /* attribute read from the hist tab file    */
                    error,              /* error read from the hist tab file        */
                    *value;             /* value read from the hist tab file        */

    double          lolimit,            /* low limit read from the hist tab file    */
                    hilimit;            /* high limit read from the hist tab file   */

    D(printf("START HT_INIT\n"));
/*
 *  Set up the msid_indices of the hist_tab struct.
 */

    for ( i = 0; i < entry_num; i++ ) {
        if ( ( Msid + i ) ->hist_ind != 0 )
             ( Htab + ( Msid + i ) ->hist_ind - 1 ) ->msid_index = i + 1;
    }

/*
 *  Group the history tab records by file name, msid name, and sorted
 *  by the time in descending order.
 */

    first = Htab;

/*
 *  After this sort the history tab records will be grouped by file
 *  then, within each file grouping they will be sorted by msid,
 *  and within each msid they will be sorted by time in descending
 *  order.
 */

/*
 *  Outside loop decrements for each number > 1.
 */

    for ( i = htab_num; i > 1; i-- ) {

/*
 *      Reset pointers for next pass.
 */

        curr = first->next_ptr;
        prev = first;
        prev_prev = NULL;

/*
 *      Inside loop goes to number on outside loop.
 */

        for ( j = 1; j < i; j++ ) {

/*
 *          Set flag on file name comparison.
 */

            flag = strcmp ( prev->file_name, curr->file_name );
```

```
/*
 *          Set flag on msid name comparison.
 */

            flag2 = strcmp ( ( Msid + ( prev->msid_index ) - 1 ) ->MSID,
                       ( Msid + ( curr->msid_index ) - 1 ) ->MSID );

/*
 *          If prev file name is greater then swap.
 */

            if ( flag > 0 ) {
                prev->next_ptr = curr->next_ptr;
                curr->next_ptr = prev;
                if ( prev_prev != NULL )
                    prev_prev->next_ptr = curr;
                else
                    first = curr;
                curr = prev;
                if ( prev_prev != NULL )
                    prev = prev_prev->next_ptr;
                else
                    prev = first;
            }

/*
 *          If prev msid name is greater then swap.
 */

            else if ( flag == 0 && flag2 > 0 ) {
                prev->next_ptr = curr->next_ptr;
                curr->next_ptr = prev;
                if ( prev_prev != NULL )
                    prev_prev->next_ptr = curr;
                else
                    first = curr;
                curr = prev;
                if ( prev_prev != NULL )
                    prev = prev_prev->next_ptr;
                else
                    prev = first;
            }

/*
 *          If prev time is less than current then swap.
 */

            else if ( ( flag == 0 ) && ( flag2 == 0 ) &&
                      ( curr->time_cntr > prev->time_cntr ) ) {
                prev->next_ptr = curr->next_ptr;
                curr->next_ptr = prev;
                if ( prev_prev != NULL )
                    prev_prev->next_ptr = curr;
                else
                    first = curr;
                curr = prev;
                if ( prev_prev != NULL )
                    prev = prev_prev->next_ptr;
                else
                    prev = first;
            }

/*
 *          Set up pointers for next pass through.
```

```
 */

            prev = prev->next_ptr;
            if ( prev_prev != NULL )
                prev_prev = prev_prev->next_ptr;
            else
                prev_prev = first;
            curr = curr->next_ptr;
        }
    }

/*
 *  Reset global to point to first in list.
 */

    Htab = first;

/*
 *  Make all entries sequential with dummy entries and set the values to NULL.
 */

    curr = Htab;
    while ( curr->next_ptr != NULL ) {
        if ( strcmp ( ( Msid + curr->msid_index - 1 ) ->MSID,
                ( Msid + curr->next_ptr->msid_index - 1 ) ->MSID ) == 0 ) {
            if ( curr->time_cntr > ( curr->next_ptr->time_cntr + 1 )
                && curr->time_cntr != 0 ) {
                i = curr->time_cntr - curr->next_ptr->time_cntr - 1;
                prev = ( struct hist_tab * ) calloc ( i, sizeof ( struct hist_tab ) );
                for ( j = 0; j < i; j++ ) {
                     ( prev + j ) ->next_ptr = prev + j + 1;
                     ( prev + j ) ->time_cntr = curr->time_cntr - j - 1;
                     ( prev + j ) ->htab_entr = INVALID;
                     ( prev + j ) ->msid_index = curr->msid_index;
                    strcpy ( ( prev + j ) ->file_name, curr->file_name );
                }
                 ( prev + i - 1 ) ->next_ptr = curr->next_ptr;
                curr->next_ptr = prev;
            }   /* end of if */
        }

        else if ( curr->time_cntr > 1 ) {   /* if the sequence didn't start at 1 */
            i = curr->time_cntr - 1;
            prev = ( struct hist_tab * ) calloc ( i, sizeof ( struct hist_tab ) );
            for ( j = 0; j < i; j++ ) {
                 ( prev + j ) ->next_ptr = prev + j + 1;
                 ( prev + j ) ->time_cntr = curr->time_cntr - j - 1;
                 ( prev + j ) ->htab_entr = INVALID;
                 ( prev + j ) ->msid_index = curr->msid_index;
                strcpy ( ( prev + j ) ->file_name, curr->file_name );
            }
             ( prev + i - 1 ) ->next_ptr = curr->next_ptr;
            curr->next_ptr = prev;
        }       /* end of else */
        curr->value = NULL;
        curr = curr->next_ptr;
    }

    if ( curr->time_cntr > 1 ) {    /* if the sequence didn't start at 1 */
        i = curr->time_cntr - 1;
        prev = ( struct hist_tab * ) calloc ( i, sizeof ( struct hist_tab ) );
        for ( j = 0; j < i; j++ ) {
             ( prev + j ) ->next_ptr = prev + j + 1;
             ( prev + j ) ->time_cntr = curr->time_cntr - j - 1;
```

```
                ( prev + j ) ->htab_entr = INVALID;
                ( prev + j ) ->msid_index = curr->msid_index;
               strcpy ( ( prev + j ) ->file_name, curr->file_name );
                ( prev + j ) ->value = NULL;
            }
             ( prev + i - 1 ) ->next_ptr = curr->next_ptr;
            curr->next_ptr = prev;
        }
        curr->value = NULL;

    /*
     *  If the history tab data files already exist then update the screen
     *  and memory with the data.
     */

        curr = Htab;

    /*
     *  Loop through the htab list creating space for each file.
     */

        while ( curr != NULL ) {

    /*
     *      Create the filename to see if it exists or not.
     */

            strcpy ( file, curr->file_name );
            if ( file[0] != '/' ) {
                strcpy ( ht_file_name, Dm_Address->display[Disp_Num].plot_path );
                strcat ( ht_file_name, file );
            } else {
                strcpy ( ht_file_name, file );
            }
            strcat ( ht_file_name, ".htb" );

    /*
     *      Open the file for read to see if it exists.
     */

            file_ptr = fopen ( ht_file_name, "rb" );
            if ( file_ptr != NULL ) {    /* The file exists */

    /*
     *          If this is the first existing data file set up the global.
     */

                if ( Ht_files == NULL ) {
                    Ht_files = ( struct ht_files * ) calloc ( 1, sizeof ( struct ht_files ) );
                    file_struct = Ht_files;
                    if ( file_struct == NULL ) {
                        tui_msg ( M_YELLOW, "error allocating history tab file struct" );
                        return ( -1 );
                    }
                    file_struct->ht_rec_ptr = curr;


    /*
     *          If this is not the first existing data file.
     */

                } else {

    /*
```

```
 *              Add the file to the list of open data files.
 */

                file_struct->next_ptr = ( struct ht_files * )
                    calloc ( 1, sizeof ( struct ht_files ) );
                if ( file_struct->next_ptr == NULL ) {
                    tui_msg ( M_YELLOW, "error allocating history tab file struct" );
                    return ( -1 );
                }
                file_struct = file_struct->next_ptr;
                file_struct->ht_rec_ptr = curr;

            }

/*
 *          Store the file pointer and read in the header information.
 */

            file_struct->file_ptr = file_ptr;
            fread ( ( void * ) &version, 2, 1, file_struct->file_ptr );
            fread ( ( void * ) flight_id, 5, 1, file_struct->file_ptr );
            fread ( ( void * ) strm_type, 3, 1, file_struct->file_ptr );
            fread ( ( void * ) &file_struct->num_entries, 4, 1, file_struct->file_ptr );
            fread ( ( void * ) &access, 2, 1, file_struct->file_ptr );

            if ( version > VERSION ) {
                tui_msg ( M_YELLOW, "version %hd of the history tab file is not supported
with this version of Display Manager" );
                return ( -1 );
            }

/*
 *          Read in the data for the current data file and store the values in temp.
 */

            while ( feof ( file_ptr ) == NO ) {

/*
 *              Read the msid, sample, and source.
 */

                fread ( ( void * ) msid, MSID_LENGTH, 1, file_struct->file_ptr );
                if ( feof ( file_ptr ) != NO ) {
                    break;
                }
                fread ( ( void * ) sample, 5, 1, file_struct->file_ptr );
                fread ( ( void * ) source, 3, 1, file_struct->file_ptr );

/*
 *              Read the decom information.
 */

                fread ( ( void * ) &size, 4, 1, file_struct->file_ptr );
                fread ( ( void * ) &length, 4, 1, file_struct->file_ptr );
                fread ( ( void * ) &num_samps, 4, 1, file_struct->file_ptr );
                fread ( ( void * ) &attribute, 1, 1, file_struct->file_ptr );
                fread ( ( void * ) &error, 1, 1, file_struct->file_ptr );

/*
 *              Read the limits and value.
 */

                fread ( ( void * ) &lolimit, 8, 1, file_struct->file_ptr );
                fread ( ( void * ) &hilimit, 8, 1, file_struct->file_ptr );
```

```
                value = ( char * ) malloc ( size );
                fread ( ( void * ) value, size, 1, file_struct->file_ptr );

/*
 *              Check to see if the msid is in the history tab list.
 */

                htab = file_struct->ht_rec_ptr;
                while ( htab != NULL && strcmp ( msid, ( Msid + htab->msid_index - 1 ) ->M
SID )
                        != 0 && ( strcmp ( htab->file_name, file ) == 0 ) )
                    htab = htab->next_ptr;

/*
 *              If the msid is found in the history tab list.
 */

                if ( htab != NULL && ( strcmp ( htab->file_name, file ) == 0 ) ) {

/*
 *                  Move values along the sequence of hist tab entries.
 */

                    while ( htab->next_ptr != NULL &&
                        strcmp ( msid, (Msid+htab->next_ptr->msid_index-1)->MSID ) == 0 &&
                        htab->time_cntr > 1 &&
                        strcmp ( htab->file_name, file ) == 0 ) {

/*
 *                      If the value exists then free up this memory.
 */

                        if ( htab->value != NULL )
                            free ( htab->value );

/*
 *                      If the previous value exists then copy it to current.
 */

                        if ( htab->next_ptr->value != NULL ) {

/*
 *                          Allocate space for value and copy it.
 */

                            htab->value = (char *)malloc ( htab->next_ptr->decom_ent.size
);
                            memcpy ( htab->value, htab->next_ptr->value, htab->next_ptr->
                                   decom_ent.size );

/*
 *                          Copy the decom information into current struct.
 */

                            htab->decom_ent.length = htab->next_ptr->decom_ent.length;
                            htab->decom_ent.size = htab->next_ptr->decom_ent.size;
                            htab->decom_ent.offset = htab->next_ptr->decom_ent.offset;
                            htab->decom_ent.num_samps = htab->next_ptr->decom_ent.num_sam
s;
                            htab->decom_ent.attribute = htab->next_ptr->decom_ent.attribut
e;
                            htab->decom_ent.error = htab->next_ptr->decom_ent.error;
                            htab->decom_ent.sample_size = htab->next_ptr->
                                decom_ent.sample_size;
```

```
                    }
                    htab = htab->next_ptr;
                }

/*
 *                  Update the most recent history tab value.
 */

                htab->value = ( char * ) malloc ( size );
                memcpy ( htab->value, value, size );
                free ( value );

/*
 *                  Update the most recent history tab decom buffer.
 */

                htab->decom_ent.size = size;
                htab->decom_ent.length = length;
                htab->decom_ent.offset = 0;
                htab->decom_ent.num_samps = num_samps;
                htab->decom_ent.attribute = attribute;
                htab->decom_ent.error = error;
                htab->decom_ent.sample_size = size / num_samps;

/*
 *                  Loop through the history tab limit entries.
 */

                while ( htab->next_ptr != NULL && htab->next_ptr->time_cntr == 0 ) {
                    htab = htab->next_ptr;
                    if ( htab->value != NULL )
                        free ( htab->value );
                    htab->value = ( char * ) malloc ( 20 ); /* allocate length 20
                                                * for string */
                    if ( htab->value == NULL ) {
                        tui_msg ( M_YELLOW, "Error allocating space for history tab va
lue" );
                        return ( -1 );
                    }
                    if ( htab->llimit_flag == 'Y' )
                        sprintf ( htab->value, "%f", lolimit );
                    else if ( htab->ulimit_flag == 'Y' )
                        sprintf ( htab->value, "%f", hilimit );
                }   /* end of while */
            }   /* end of if msid found in htab list */
        }   /* end of while not end of file */
    }       /* end of if file exists */

/*
 *      Move the history tab pointer past the file entries.
 */

        while ( curr && strcmp ( file, curr->file_name ) == 0 )
            curr = curr->next_ptr;
    }

/*
 *  Loop through the history tab limit entries.
 */

    curr = Htab;
    i = 0;
    while ( curr != NULL ) {
```

```
        if ( curr->value != NULL && curr->htab_entr != INVALID ) {
            i++;
            msid_ptr = Msid + curr->msid_index - 1;
            tab_ptr = Tab + msid_ptr->Tab_Index - 1;
            status = extract ( curr->value, &curr->decom_ent );
            msid_ptr->Wid_Ind = i;
            updtfg ( Disp_Num, &curr->decom_ent, msid_ptr,
                        tab_ptr, status);
        }
        curr = curr->next_ptr;
    }

    D(printf("END HT_INIT\n"));
    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: init.c
 *
 *  This function performs all one-time initialization.
 *
 *
 * INTERNAL FUNCTIONS:
 *
 *      o   control_fpe -   Called when a floating point exception occurs.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  K. Noonan       - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <fcntl.h>
#include <signal.h>
#include <X11/Xlib.h>
#include <X11/cursorfont.h>
#include <X11/Intrinsic.h>
#include <sys/types.h>
#include <constants.h>
#include <disp.h>
#include <pf_key.h>
#include <wex/EXmsg.h>
#include <wex/EXerror.h>
#include <wex/EXexec.h>
#include <wex/EXWEX.h>


extern struct data_info *Dh_Address;        /* DH shared memory */
extern struct dm_shmemory *Dm_Address;      /* DM shared memory */
extern struct pfkey_defs Current_Com;

extern int      errno;                      /* error return value                   */

extern short    Disp_Num,                   /* slot nbr where the display info is   */
                First_Copy,                 /* set to a 1 if first Display Manager  */
                Good_Strm,                  /* set to YES if data type is valid     */
                Flt_Selected;               /* Yes, if flight/data type input       */

extern char     Disp_Path[DNAME_LEN],
                Plot_Path[DNAME_LEN];


int init ( )
{
    int             error,                  /* return value for function calls      */
                    pid,                    /* process Id for new processes         */
                    first_copy,             /* flag signifying first Display Manager */
                    i,                      /* counter                              */
                    wex_err = NO,           /* error flag set if error on flt id or */
                    new,                    /* contains new position Id index       */
```

```
                    match,                  /* set to YES for pos. Id match          */
                    control_fpe ( ),
                    val_fn      ( );

    char            access_mode[20],        /* WEX access mode                       */
                    flight_id[9];           /* flt id obtained from WEX              */


    D(printf("START init\n"));
/*
 *  Set up the signal handler for floating point exceptions.
 */

    signal ( SIGFPE, control_fpe );

/*
 *  Call a routine to determine if this is the first Display Manager in the w/s.
 */

    if ( ( first_copy = first_proc ( ) ) == -1 )
        return ( -1 );

/*
 *  Check the version date.
 */

    date_chek ( );

/*
 *  Initialize WEX.
 */

    EXwexinit ( WEX_MSG_Q );

/*
 *  If first copy and the shared memory does not exist then call a routine
 *  to create the shared memory.   Get the access mode of the workstation.  Clear flags
 *  in shared memory.  Set up flags and start the Data Handler task and monitor the
 *  time for the initialization to complete. Store the Data Handler process Id.  Attach
 *  to the Data Handler shared memory.
 */

    if ( first_copy != NO ) {
        if ( first_copy == CREAT_SHM ) {
            if ( ( error = shm_creat ( ) ) == -1 )
                return ( -1 );
        }

        Dm_Address->process.disp_nbr = 1;

/*
 *      Get the access mode of the workstation (development/operational)
 */

        error = EXaccess ( access_mode );

        if ( error == -1 ) {
            tui_msg ( M_YELLOW, "Error %d in obtaining the access mode of workstation",
                  errno );
            return ( -1 );
        } else {
            if ( ( strcmp ( access_mode, "DEVELOPMENT" ) ) == 0 )
                Dm_Address->process.wex_mode = DEV;
            else
```

```
            Dm_Address->process.wex_mode = OPS;
        }

        First_Copy = YES;

/*
 *      Initialize all flags in the Display Manager shared memory
 *      since this is the first Display Manager on this workstation.
 */

        for ( i = 0; i < MAX_DISP; i++ ) {
            Dm_Address->dm_pid[i] = -1;
            Dm_Address->display[i].disp_active = NO;
            Dm_Address->display[i].active_display = NO;
            Dm_Address->display[i].disp_init = NO;
            Dm_Address->display[i].disp_pid = -1;
            Dm_Address->display[i].clear = NO;
            Dm_Address->display[i].dh_clear = NO;
            Dm_Address->display[i].halt = NO;
            Dm_Address->display[i].get_lim = NO;
            Dm_Address->display[i].upd_lim = NO;
            Dm_Address->display[i].dh_new_disp = NO;
            Dm_Address->display[i].disp_pause = NO;
            Dm_Address->display[i].new_display = NO;
            Dm_Address->display[i].upd_lim = NO;
            Dm_Address->display[i].grp_lim = NO;
            Dm_Address->display[i].dh_plot = NO;
            Dm_Address->display[i].read_plot = NO;
            Dm_Address->display[i].dd_strt = NO;
            Dm_Address->display[i].dd_stop = NO;
            Dm_Address->display[i].log_enable = NO;
            Dm_Address->display[i].update_rate = 1000;
            Dm_Address->display[i].low_x = 0.0;
            Dm_Address->display[i].low_y = 0.0;
            Dm_Address->display[i].high_x = 100.0;
            Dm_Address->display[i].high_y = 100.0;
        }

        for ( i = 0; i < MAX_POS_ID; i++ ) {
            Dm_Address->process.pos_id[i][0] = 0;
            Dm_Address->process.nbr_pos[i] = 0;
            Dm_Address->process.alarm[i] = YES;
        }

        for ( i = 0; i < MAX_FLTS; i++ ) {
            strncpy ( Dm_Address->strm[i].flt_id, "    \0", 4 );
            strncpy ( Dm_Address->strm[i].strm_type, "  \0", 3 );
            Dm_Address->strm[i].nbr_conn = 0;
        }

        Dm_Address->process.nbr_streams = 0;
        Dm_Address->process.dh_not_halted = NO;
        Dm_Address->process.dh_initialized = NO;
        Dm_Address->process.disp_init = NO;
        Dm_Address->process.dh_ack_evnt = NO;
        Dm_Address->process.dh_evnt = NO;
        Dm_Address->pbi_shmemory.disp_num = -1;
        Dm_Address->pbi_shmemory.number_of_changes = 0;

/*
 *      Start the Data Handler.  Verify assignment of a valid
 *      process id.  Wait for the Data Handler to set its
 *      initialization flag, then record it's PID in DM shared
 *      memory.
```

```
 */

#ifdef SUN
        pid = EXexec ( "/WEX/Exec/dh_sun",
                ARG_LIST, "dh_sun",  END_OF_LIST, END_OF_LIST );
#else
        pid = EXexec ( "/WEX/Exec/dh_mass",
                ARG_LIST, "dh_mass", END_OF_LIST, END_OF_LIST );
#endif

        if ( pid == -1 ) {
            tui_msg ( M_YELLOW, "Error %d on starting Data Handler", errno );
            return ( -1 );
        }

        error = chk_flg ( &Dm_Address->process.dh_initialized, DDH_LOOP, 1 );
        if ( error == -1 ) {
            tui_msg ( M_YELLOW, "Display Manager - Data Handler task not initialized" );
            return ( -1 );
        }

        Dm_Address->process.dh_pid = pid;

#ifdef STUB

        pid = EXexec ( "/WEX/Exec/pdt_feed",
                ARG_LIST, "pdt_feed", END_OF_LIST, END_OF_LIST );

#endif

/*
 *  If not the first Display Manager process, assure First_Copy is negated
 *  and increment the Display Manager process counter.
 */

    } else {
        Dm_Address->process.disp_nbr++;
        First_Copy = NO;
    }

/*
 *  Attach to the Data Handler shared memory
 */

    Dh_Address = (struct data_info *)shmat (Dm_Address->process.data_shm_id, 0, 0);
    if ( (int) Dh_Address == -1 ) {
        tui_msg ( M_YELLOW, "Error %d on Data Handler shared memory attach", errno );
        return ( -1 );
    }

/*
 * Get the flight Id, data type, and position Id from WEX and store with the display
 * information.  If errors occurs on the flight or data type call, set flight or data
 * type to null and set an internal error flag to YES. If an error occurs on the
 * position call, default the position Id to GNC.
 */

    if ( wex_err == NO ) {
        error = EXflight ( Dm_Address->dm_pid[Disp_Num], flight_id );
        if ( error != 0 ) {
            tui_msg ( M_YELLOW, "Error %d on WEX EXflight call", errno );
            Dm_Address->display[Disp_Num].flight_id[0] = '\0';
            wex_err = YES;
        } else {
```

```
            strncpy ( Dm_Address->display[Disp_Num].flight_id, flight_id, 3 );
            Dm_Address->display[Disp_Num].flight_id[3] = 0;
        }

        error = EXdatatype ( Dm_Address->dm_pid[Disp_Num],
                             Dm_Address->display[Disp_Num].strm_type );
        if ( error != 0 ) {
            tui_msg ( M_YELLOW, "Error %d on WEX EXdatatype call", errno );
            Dm_Address->display[Disp_Num].strm_type[0] = '\0';
            wex_err = YES;
        } else {

            if ( Dm_Address->display[Disp_Num].strm_type[0] != 0 ) {
                val_dt ( Dm_Address->display[Disp_Num].strm_type );
                if ( Good_Strm == NO )
                    wex_err = YES;
            }
        }

        if ( ( Dm_Address->display[Disp_Num].flight_id[0] == 0 ) || ( Good_Strm != YES ) )
{
            wex_err = YES;
        } else {
            Flt_Selected = YES;
        }

        error = EXposition ( Dm_Address->dm_pid[Disp_Num],
                             Dm_Address->display[Disp_Num].pos_id );
        if ( error != 0 ) {
            tui_msg ( M_YELLOW, "Error %d on WEX EXposition call- default to GNC ", errno
);
            strncpy ( Dm_Address->display[Disp_Num].pos_id, "GNC\0", 4 );

        } else if ( Dm_Address->display[Disp_Num].pos_id[0] == NULL ) {
            tui_msg ( M_YELLOW, "Position Id not available - default to GNC", errno );
            strncpy ( Dm_Address->display[Disp_Num].pos_id, "GNC\0", 4 );
        }

    } else {
        tui_msg ( M_WHITE, "Default position Id to GNC", errno );
        strncpy ( Dm_Address->display[Disp_Num].pos_id, "GNC\0", 4 );
    }

/*
 * Search the active position Id table and see if this is a new position Id or a
 * currently existing position.  Set an index which points to the position Id and the
 * position Id alarm flag. Increment the number of users for this position Id.
 */

    i = match = new = 0;
    while ( ( i < MAX_POS_ID ) && ( match == NO ) ) {
        if ( ( strcmp ( Dm_Address->display[Disp_Num].pos_id,
            Dm_Address->process.pos_id[i] ) == 0 ) ) {
            match = YES;
            Dm_Address->process.nbr_pos[i]++;
            Dm_Address->display[Disp_Num].pos_id_indx = i;
        } else
            if ( Dm_Address->process.nbr_pos[i] <= 0 )
                new = i;

        i++;
    }

    if ( match == NO ) {
```

```
        Dm_Address->process.nbr_pos[new] = 1;
        Dm_Address->display[Disp_Num].pos_id_indx = new;
        strcpy ( Dm_Address->process.pos_id[new],
                    Dm_Address->display[Disp_Num].pos_id );
        Dm_Address->process.alarm[new] = YES;
    }

/*
 *  RLK 9/19/90 Need to make this display-specific.  Actually, each gets
 *              its own copy (see below), so why the globals?
 *
 * Generate the default path name of the display universal plots and data files.
 * Then copy the default path names into shared memory.
 */

    strcpy ( Disp_Path, "/WEX/Datafiles/display/" );
    strcpy ( Plot_Path, "/user/display/" );

    strcat ( Disp_Path, Dm_Address->display[Disp_Num].pos_id );
    strcat ( Plot_Path, Dm_Address->display[Disp_Num].pos_id );

    strcat ( Disp_Path, "/" );
    strcat ( Plot_Path, "/" );

    strcpy ( Dm_Address->display[Disp_Num].plot_path, Plot_Path );
    strcpy ( Dm_Address->display[Disp_Num].disp_path, Disp_Path );

/*
 *  RLK 9/19/90 Need to make this display-specific
 *
 *  Call a routine to validate the flight id and stream tag.
 */

    if ( wex_err == NO )
        wex_err = chk_flt ( YES );

/*
 * Read in the default PF key definitions for the workstation.
 */

    read_pf ( YES, NONE );

/* RLK 9/19/90  Placed here temporarily until get "Current_Display" or whatever working */

    Disp_Num = 0;

    tui_msg ( M_WHITE, "Display Manager %d initialized", Dm_Address->process.disp_nbr);

    D(printf("END init\n"));
    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: control_fpe
 *
 *  This function processes floating point exceptions.
 *****************************************************************************/

int control_fpe ( )
{
    signal ( SIGFPE, control_fpe );
}
```

```
/*****************************************************************************
 * MODULE NAME: init_disp.c
 *
 *  The initialization routine initializes all X attributes and creates the
 *  top-level drawing and scrolling widgets for a new display (if the
 *  display is already active and is being reinitialized, the above actions
 *  are not performed).  The background and foreground DDF files are read.
 *  The timer is set for the foreground update callback here first.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Richard Romeo - Ford Aerospace Corporation/Houston
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Ronnie Killough - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <X11/Shell.h>
#include <Xm/Xm.h>
#include <Xm/ScrolledW.h>
#include <Xm/DrawingA.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <signal.h>
#include <constants.h>
#include <DDdisp.h>
#include <DDplot.h>
#include <disp.h>
#include <user_inter.h>
#include <wex/EXmsg.h>


extern Widget               Top, Scrl_Win, Draw_Win;
extern Colormap             Main_cmap;
extern struct dm_shmemory   *Dm_Address;        /* DM shared memory address      */
extern struct bg_recs       Bg_Rec;             /* bg record header structure    */
extern int                  errno;              /* system call return value      */
extern int                  Nbr_of_plots;       /* # plots for this display      */


init_disp(disp_num)

    short   disp_num;                           /* display # of new display      */
{
    register int    i;                          /* argument list counter         */

    Arg                 args[10];               /* argument array for X calls    */
    Display             *xdisplay;              /* ptr to X display for display  */
    XGCValues           *gc_val;                /* contains GC initial values    */
    XtCallbackProc      cb_pbi(),               /* callback procedures           */
                        cb_cmd(),
                        cb_expose_display();

    struct disp_info    *display;               /* ptr to display disp info struct */
```

```
    unsigned long       gc_mask;                /* mask for GC values              */

    int     total_nbr_bgrecords,                /* total number of bg records      */
            total_nbr_fgrecords;                /* total number of fg records      */


    D(printf("START init_disp\n"));

/*
 *  If this display is not active, initialize X objects.
 */

    if (Dm_Address->display[disp_num].disp_active == NO) {

        D(printf("Activating display number %d...\n", disp_num));

        Dm_Address->display[disp_num].disp_active = YES;

/*
 *      Create the shell widget which is the parent of the display window.
 */

        i = 0;
        Dm_Address->shell[disp_num] =
            tui_create_app_shell ( "Display Window", Main_cmap, args, i );

        i = 0;
        XtManageChild ( Scrl_Win = XmCreateScrolledWindow ( Dm_Address->shell[disp_num],
                                                            "scroll", args, i ) );

        i = 0;
        XtManageChild ( Draw_Win = XmCreateDrawingArea ( Scrl_Win, "draw", args, i ) );
        XtAddCallback ( Draw_Win, XmNinputCallback,  cb_pbi,            0 );
        XtAddCallback(Draw_Win, XmNexposeCallback, cb_expose_display, disp_num);

        XtRealizeWidget ( Dm_Address->shell[disp_num] );

        Dm_Address->window[disp_num] = XtWindow ( Draw_Win );

/*
 *      Initialize the color map for the window.
 */

        set_cmap ( Dm_Address->shell[disp_num] );

/*
 *      Set values in gc values structure
 */

        gc_val = &Dm_Address->gc_val[disp_num];
        xdisplay = Dm_Address->xdisplay[disp_num];

        gc_val->function = GXcopy;                              /* default */
        gc_val->plane_mask = AllPlanes;                         /* default */
        gc_val->foreground = WhitePixel(xdisplay, DefaultScreen(xdisplay));
        gc_val->background = BlackPixel(xdisplay, DefaultScreen(xdisplay));
        gc_val->line_width = 1;
        gc_val->line_style = LineSolid;                         /* default */
        gc_val->cap_style = CapButt;                            /* default */
        gc_val->join_style = JoinMiter;                         /* default */
        gc_val->fill_style = FillSolid;                         /* default */
        gc_val->fill_rule = EvenOddRule;                        /* default */
        gc_val->arc_mode = ArcChord;
```

```
        /* gc_val->tile                                    /* default */
        /* gc_val->stipple                                 /* default */
        gc_val->ts_x_origin = 0;                           /* default */
        gc_val->ts_y_origin = 0;                           /* default */
        /* gc_val->font                                    /* default */
        gc_val->subwindow_mode = ClipByChildren;           /* default */
        gc_val->graphics_exposures = True;                 /* default */
        gc_val->clip_x_origin = 0;                         /* default */
        gc_val->clip_y_origin = 0;                         /* default */
        gc_val->clip_mask = None;                          /* default */
        gc_val->dash_offset = 0;                           /* default */
        gc_val->dashes = 4;                                /* default */

/*
 *      Set mask for update of non-default values
 */

        gc_mask = GCForeground | GCBackground | GCLineWidth | GCArcMode;

/*
 *      Create gc with set values
 */

        Dm_Address->gc[disp_num] =
            XCreateGC(xdisplay, Dm_Address->window[disp_num], gc_mask, gc_val);

/*
 *      If this display is active, must be selecting new display for
 *      this display number.  Clear old display, free memory
 *      (in clear()) and re-initialize for new display.
 */

    } else {

        clear(disp_num);
        gc_val = &Dm_Address->gc_val[disp_num];
        xdisplay = Dm_Address->xdisplay[disp_num];
    }

/*
 *  Read bg display definition file
 */

    total_nbr_bgrecords = readbg(disp_num);

    if (total_nbr_bgrecords <= 0)
        tui_msg(M_YELLOW, "There are no background records available\n");

    display = &Dm_Address->display[disp_num];

/*
 *  Set the background of the display gc and of the window according to
 *  the color specified in the background file.
 */

    gc_val->background = Bg_Rec.s_color;
    XSetBackground(xdisplay, Dm_Address->gc[disp_num], Bg_Rec.s_color);
    XSetWindowBackground(xdisplay, Dm_Address->window[disp_num],Bg_Rec.s_color);

/*
 *  Set the size of the scrolled window widget to the size of the display
 *  plus a few pixels to account for spacing and borders. MDC - fix to really
 *  take into account the borders and such.
 */
```

```
    i = 0;
    XtSetArg ( args[i], XmNwidth,  display->size_x + 4 ); i++;
    XtSetArg ( args[i], XmNheight, display->size_y + 4 ); i++;
    XtSetValues ( Scrl_Win, args, i );

/*
 *  Set the size of the drawing area widget to the exact size of the image.
 *  This is necessary, as it sets up the relationship between the size of
 *  the drawing area and the scrolled window widget.
 */

    i = 0;
    XtSetArg ( args[i], XmNwidth,  display->size_x ); i++;
    XtSetArg ( args[i], XmNheight, display->size_y ); i++;
    XtSetValues ( Draw_Win, args, i );

/*
 *  Read foreground Display Definition File and draw background
 *  axes and grid lines for all active plots (done in readfg()).
 */

    total_nbr_fgrecords = readfg(disp_num);

    if ((total_nbr_bgrecords <= 0) || (total_nbr_fgrecords <= 0)) {
        tui_msg(M_YELLOW, "There are no records available");
        return(-1);
    }

/*
 *  RLK 9/26/90 Here need to check out the fg/bg records and fonts and create
 *              GCs for the common colors/fonts
 *
 *  Actually, should prob. do this in readfg(), accumulating counts as
 *  the stuff is read, then call a routine to figure out the colors/fonts
 *
 *  XCreateGC, XSetForeground, XSetBackground, XLoadQueryFont, XSetFont, etc.
 */

/* RLK 10/25/90 font stuff
 *  Initialize fonts

    DDfont_init();
 */

/*
 *  Return successful initialization flag to DM and clear new display flag
 */

    Dm_Address->display[disp_num].new_display = NO;
    Dm_Address->display[disp_num].disp_init = YES;

/*
 *  Set up callback timer for foreground update
 */

    set_timer(disp_num);

    D(printf("END init_disp\n"));
    return(0);
}
```

```
/*****************************************************************************
 * MODULE NAME: init_fg.c
 *
 *  This routine displays the Dead status for all the data fields,
 *  pending the first foreground data update and initializes the data
 *  buffers to all z's.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Richard Romeo - Ford Aerospace Corporation/Houston
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Ronnie Killough &
 *  Nancy Martin    - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <X11/Xlib.h>
#include <constants.h>
#include <disp.h>
#include <DDdisp.h>
#include <Dbdata.h>
#include <wex/EXmsg.h>


extern struct dm_shmemory   *Dm_Address;
extern struct tabular_ent   *Tab;
extern struct fg_file_header    *Ffile;
extern struct msid_ent      *Msid;
extern struct limit_ent     *Limit;

extern unsigned char        Old_Data[60000];
extern unsigned char        Graph_Old_Data[60000];

extern float                Font65_height;
extern float                Font80_height;
extern float                Font100_height;

extern Widget               Draw_Win;


init_fg (disp_num)

    short   disp_num;
{
    register struct msid_ent    *msid_info;

    struct tabular_ent          *tab_info;
    struct limit_ent            *lim_info;

    int     i, j, n;

    DbElement   Wdata[MAX_MSIDS];
    Widget      db_data;
    Arg         wargs[20];
    double      init_val = 0;
```

```
    D(printf("START INIT_FG\n"));
/*
 * Set dead status for each msid
 */

    msid_info = Msid;

    n = 0;
    for (i = 0; i < Ffile->Entry_Num; i++) {

        if (msid_info->Stat_Flag != 0 && msid_info->hist_ind == 0) {
            tab_info = Tab + msid_info->Tab_Index - 1;

/*
 *          Nancy is beginning to make the move to the Dbdata widget.
 */

            Wdata[n].Value = (char*)malloc(8);
            memcpy (&Wdata[n].Value, &init_val, 8);
            printf ("Value %d\n", Wdata[n].Value);
            Wdata[n].Attrib = 0;
            Wdata[n].Type = msid_info->Scrn_Type;
            Wdata[n].Width = tab_info->Data_Width;
            Wdata[n].Precision = tab_info->Dig_Right;
            Wdata[n].JustFlag = tab_info->Just_Flag;
            Wdata[n].DispStat = DEAD_DATA;
            Wdata[n].StatFlag = 0;
            Wdata[n].X = tab_info->X_XC;
            Wdata[n].Y = tab_info->Y_XC;
            if ( msid_info->Limit_Ind > 0 ) {
                lim_info = Limit + msid_info->Limit_Ind - 1;
                Wdata[n].MinLimit = lim_info->Low_Limit;
                Wdata[n].MaxLimit = lim_info->Hi_Limit;
                Wdata[n].LowColor = lim_info->Lo_Color;
                Wdata[n].HiColor = lim_info->Hi_Color;
                Wdata[n].CrMinLimit = lim_info->Crit_Low;
                Wdata[n].CrMaxLimit = lim_info->Crit_Hi;
                Wdata[n].CrLColor = lim_info->Cr_Lcolor;
                Wdata[n].CrHColor = lim_info->Cr_Hcolor;
            } else {
                Wdata[n].MinLimit = 0;
                Wdata[n].MaxLimit = 0;
                Wdata[n].LowColor = msid_info->Nom_Color;
                Wdata[n].HiColor = msid_info->Nom_Color;
                Wdata[n].CrMinLimit = 0;
                Wdata[n].CrMaxLimit = 0;
                Wdata[n].CrLColor = msid_info->Nom_Color;
                Wdata[n].CrHColor = msid_info->Nom_Color;
            }
            Wdata[n].NomColor = msid_info->Nom_Color;
            Wdata[n].StaColor = msid_info->Sta_Color;
            Wdata[n].OvrColor = msid_info->Ovr_Color;
            Wdata[n].DeadColor = msid_info->Dead_Color;

/* RLK 9/12/90  More font stuff to fix.

            Wdata[n].DefFont = Pixels(tab_info->Dead_Color);

            if ((tab_info->Font_Num - 1) % 3 == 0)
                fp.font = 1;
            else
                if ((tab_info->Font_Num - 2) % 3 == 0)
                    fp.font = 2;
                else
```

```
                fp.font = 3;

            if (tab_info->Font_Num < 4)
                gsetcharheight (Font65_height);
            else
                if (tab_info->Font_Num < 7)
                    gsetcharheight (Font80_height);
                else
                    gsetcharheight (Font100_height);

            gsettextfontprec (&fp);
*/
            n++;
        }
        msid_info++;
    }

/*
 *  Create a Dbdata widget to display all foreground values in their specified
 *  positions.
 */

    i = 0;
    printf("INITFG num values %d\n", n);
    XtSetArg (wargs[i], XtNvalues, Wdata); i++;
    XtSetArg (wargs[i], XtNnumValues, n); i++;
    XtSetArg (wargs[i], XtNpacking, XeNO_PACKING); i++;
    db_data = XtCreateManagedWidget ("data", XedbdataWidgetClass, Draw_Win,
                                     wargs, i);

/*
 *  Fill old data buffer with z's
 */

    for (i = 0; i < 60000; i++) {
        Old_Data[i] = 'z';
        Graph_Old_Data[i] = 'z';
    }

    D(printf("END INIT_FG\n"));
    return (0);
}
```

```
/*****************************************************************************
 * MODULE NAME: init_label
 *
 *  This function initializes the labels on menu entries which act as toggles.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <X11/Intrinsic.h>
#include <constants.h>
#include <disp.h>
#include <wex/EXmsg.h>


extern Widget                Pb_Alarm, Pb_Pbi, Pb_Log, Pb_Log_A, Pb_Msg, Pb_Pf;
extern struct dm_shmemory    *Dm_Address;

extern short                 Disp_Num,
                             Msg_Popup_Flag,
                             Pbi_Disable;


int init_label ( )
{
    register int    index;

    D(printf("START init_label\n"));
/*
 *  Initialize alarm label.
 */

    index = Dm_Address->display[Disp_Num].pos_id_indx;
    if ( Dm_Address->process.alarm[index] == ON )
        set_label ( Pb_Alarm, "Disable Alarms" );
    else
        set_label ( Pb_Alarm, "Enable Alarms" );

/*
 *  Initialize Pbi's label.
 */

    if ( Pbi_Disable == ENABLED )
        set_label ( Pb_Pbi, "Disable PBI's" );
    else
        set_label ( Pb_Pbi, "Enable PBI's"  );

/*
 *  Initialize logging label.
 */

    if ( Dm_Address->display[Disp_Num].log_enable == YES )
        set_label ( Pb_Log, "Disable Logging" );
    else
        set_label ( Pb_Log, "Enable Logging"  );

/*
 *  Initialize all logging label.
```

```
 */

    if ( Dm_Address->process.log_enable == YES )
        set_label ( Pb_Log_A, "Disable All Logging" );
    else
        set_label ( Pb_Log_A, "Enable All Logging"  );

/*
 *  Initialize message label.
 */

    if ( Msg_Popup_Flag == YES )
        set_label ( Pb_Msg, "Disable Messages" );
    else
        set_label ( Pb_Msg, "Enable Messages"  );

/*
 *  Initialize FREEZE/PAUSE label;
 */

    set_label ( Pb_Pf, "Freeze Display" );

    D(printf("END init_label\n"));
    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: int_ln.c
 *
 *  This function determines if any of the given lines intersect
 *  the bounding lines of the redraw box.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Tod Milam       - Ford Aerospace Corporation/Houston
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Ronnie Killough - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <X11/Xlib.h>
#include <wex/EXmsg.h>
#include <constants.h>


int int_ln(ulx, uly, lrx, lry, points, num_pts)

    short   ulx, uly, lrx, lry;         /* coordinates of the redraw box    */

    XPoint  points[];                   /* endpoints of the lines           */

    short   num_pts;                    /* number of endpoints              */
{

    int     pt_cnt,         /* loop counter                                 */
            redraw_flag,    /* set as soon as redraw necessity determined   */
            x, y;           /* temp x/y coordinates                         */

/*
 *  Check each line to see if it crosses a line of the redraw box
 */

    pt_cnt = 0;
    redraw_flag = NO;

    while (redraw_flag == NO && pt_cnt < (num_pts - 1)) {

        if (points[pt_cnt + 1].x != points[pt_cnt].x
            && ((ulx > points[pt_cnt + 1].x && ulx < points[pt_cnt].x)
                || (points[pt_cnt + 1].x > ulx && points[pt_cnt].x < ulx))) {

/*
 *          Calculate the intersection point of the line and the edge of the box
 */

            y = points[pt_cnt].y + (points[pt_cnt+1].y - points[pt_cnt].y)
                        / (points[pt_cnt+1].x - points[pt_cnt].x)
                        * (ulx - points[pt_cnt].x);

/*
 *          If this point lies on the two lines then set the redraw flag.
 */
```

```
            if (y < uly && y > lry) {

                if (y <= points[pt_cnt].y && y >= points[pt_cnt+1].y)
                    redraw_flag = YES;
                else if (y < points[pt_cnt+1].y && y > points[pt_cnt].y)
                    redraw_flag = YES;
            }

        }

/*
 *      If the redraw flag has not yet been set, check the remaining
 *      edges of the redraw box.
 */

        if (!redraw_flag) {

            if (points[pt_cnt+1].x != points[pt_cnt].x
                && ((lrx > points[pt_cnt+1].x && lrx < points[pt_cnt].x)
                    || (points[pt_cnt+1].x > lrx && points[pt_cnt].x < lrx))) {

/*
 *              Calculate the intersection point of the
 *              line and the edge of the box.
 */

                y = points[pt_cnt].y + (points[pt_cnt+1].y - points[pt_cnt].y)
                        / (points[pt_cnt+1].x - points[pt_cnt].x)
                        * (lrx - points[pt_cnt].x);

/*
 *              If this point lies on the two lines
 *              then set the redraw flag.
 */

                if (y < uly && y > lry) {

                    if (y <= points[pt_cnt].y && y >= points[pt_cnt+1].y)
                        redraw_flag = YES;
                    else if (y < points[pt_cnt+1].y && y > points[pt_cnt].y)
                        redraw_flag = YES;

                }
            }
        }

/*
 *      If the redraw flag has not yet been set, check the remaining
 *      edges of the redraw box.
 */

        if (!redraw_flag) {

            if (points[pt_cnt+1].y != points[pt_cnt].y
                && ((uly > points[pt_cnt+1].y && uly < points[pt_cnt].y)
                    || (points[pt_cnt+1].y > uly && points[pt_cnt].y < uly))) {

/*
 *              Calculate the intersection point of the
 *              line and the edge of the box.
 */

                x = points[pt_cnt].x + (points[pt_cnt+1].x - points[pt_cnt].x)
```

```
                        / (points[pt_cnt+1].y - points[pt_cnt].y)
                        * (uly - points[pt_cnt].y);

/*
 *              If this point is on the two lines then set the redraw flag.
 */

                if (x < ulx && x > lrx) {

                    if (x <= points[pt_cnt].x && x >= points[pt_cnt+1].x)
                        redraw_flag = YES;
                    else if (x < points[pt_cnt+1].x && x > points[pt_cnt].x)
                        redraw_flag = YES;
                }
            }
        }

/*
 *      If the redraw flag has not yet been set, check the remaining
 *      edges of the redraw box.
 */

        if (!redraw_flag) {

            if (points[pt_cnt+1].y != points[pt_cnt].y
                && ((lry > points[pt_cnt+1].y && lry < points[pt_cnt].y)
                    || (points[pt_cnt+1].y > lry && points[pt_cnt].y < lry))) {

/*
 *              Calculate the intersection point of the
 *              line and the edge of the box.
 */

                x = points[pt_cnt].x + (points[pt_cnt+1].x - points[pt_cnt].x)
                        / (points[pt_cnt+1].y - points[pt_cnt].y)
                        * (lry - points[pt_cnt].y);

/*
 *              If this point is on the two lines then set the redraw flag.
 */

                if (x < ulx && x > lrx) {

                    if (x <= points[pt_cnt].x && x >= points[pt_cnt+1].x)
                        redraw_flag = YES;
                    else if (x < points[pt_cnt+1].x && x > points[pt_cnt].x)
                        redraw_flag = YES;

                }
            }
        }

        pt_cnt++;
    }

    return (redraw_flag);
}
```

```
/*****************************************************************************
 * MODULE NAME: lim_grp.c
 *
 *  This function allows the user to turn a limit group on or off.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  C. Davis        - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <constants.h>
#include <pf_key.h>
#include <disp.h>
#include <wex/EXmsg.h>


extern struct pfkey_defs    Current_Com;
extern struct dm_shmemory   *Dm_Address;
extern struct limit_file    *First_lim_ptr,
                            *Last_lim_ptr;

extern short                Disp_Num;
extern char                 Disp_Path[DNAME_LEN];
extern int                  errno;


int lim_grp ( )
{
    struct limit_file           *temp_ptr,
                                *lim_ptr,
                                *current_lim_ptr;

    short                       error = 0,      /* return value                 */
                                match;

    char                        *calloc ( );

    D(printf("START lim_grp\n"));
/*
 *  Send the turn on or off limit group command to the Data Handler.
 *  Call the chk_flg routine to monitor a response from the Data Handler.
 */

    Dm_Address->display[Disp_Num].display_name[0] = 0;
    if ( Current_Com.disp_name[0] != '/' )
        strcpy ( Dm_Address->display[Disp_Num].display_name, Disp_Path );

    strcat ( Dm_Address->display[Disp_Num].display_name, Current_Com.disp_name );
    Dm_Address->display[Disp_Num].action = (Current_Com.func_no == LIM_GRP) ? START : STOP
;
    Dm_Address->display[Disp_Num].grp_lim = YES;
    Dm_Address->display[Disp_Num].dh_grp_limit = NO;
    error = chk_flg ( &Dm_Address->display[Disp_Num].dh_grp_limit, 10, 1 );
```

```
    if ( error ) {
        tui_msg ( M_YELLOW, "Data Handler - Unable to process the limit group command" );

/*
 *  If the command is to STOP the limit, then check the link
 *  list for the file name to delete.  Once found, the file will
 *  be deleted, and the pointers to the other file name in the
 *  list will be adjusted.
 */

    } else {
        if ( Current_Com.func_no == LIM_GRP_OFF ) {
            match = NO;
            temp_ptr = First_lim_ptr;
            if ( temp_ptr != NULL ) {
                while ( match == NO && temp_ptr != NULL ) {
                    if ( strcmp ( Dm_Address->display[Disp_Num].display_name,
                                  temp_ptr->file_name ) == 0 ) {
                        match = YES;
                        lim_ptr = temp_ptr;
                    } else
                        temp_ptr = temp_ptr->next_ptr;
                }
                if ( match == YES ) {
                    if ( lim_ptr->prev_ptr != NULL )
                        lim_ptr->prev_ptr->next_ptr = lim_ptr->next_ptr;
                    else {
                        if ( lim_ptr->next_ptr != NULL )
                            First_lim_ptr = lim_ptr->next_ptr;
                        else
                            First_lim_ptr = NULL;
                    }
                    if ( lim_ptr->next_ptr != NULL )
                        lim_ptr->next_ptr->prev_ptr = lim_ptr->prev_ptr;
                    else
                        Last_lim_ptr = lim_ptr->prev_ptr;

                    free ( lim_ptr );
                }
            }

/*
 *      Else, we want to start the limit, so add it to the link list.
 */

        } else if ( Current_Com.func_no == LIM_GRP ) {
            if ( First_lim_ptr == NULL ) {
                First_lim_ptr = (struct limit_file *)
                                    calloc ( 1, sizeof ( struct limit_file ) );
                if ( First_lim_ptr == NULL ) {
                    tui_msg ( M_YELLOW, "Error %d on calloc of Limit List", errno );
                    return ( -1 );
                } else {
                    current_lim_ptr = First_lim_ptr;
                    current_lim_ptr->prev_ptr = NULL;
                    current_lim_ptr->next_ptr = NULL;
                    Last_lim_ptr = First_lim_ptr;
                }
            } else {
                current_lim_ptr = ( struct limit_file * )
                                        calloc ( 1, sizeof ( struct limit_file ) );
                if ( current_lim_ptr == NULL ) {
                    tui_msg ( M_YELLOW, "Error %d on calloc of Limit List", errno );
                    return ( -1 );
```

```
                } else {
                    Last_lim_ptr->next_ptr = current_lim_ptr;
                    current_lim_ptr->prev_ptr = Last_lim_ptr;
                    current_lim_ptr->next_ptr = NULL;
                    Last_lim_ptr = current_lim_ptr;
                }
            }

/*
 *          Store the file name into the link list
 */

            if ( Current_Com.disp_name[0] != '/' )
                strcpy ( current_lim_ptr->file_name, Disp_Path );
            strcat ( current_lim_ptr->file_name, Current_Com.disp_name );
        }
    }
    D(printf("END lim_grp\n"));
    return ( error );
}
```

```
/*****************************************************************************
 * MODULE NAME: lim_ln.c
 *
 *  This function draws a nominal or limit line for a plot.  Note, coefficient
 *  limit lines have not been tested.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Richard Romeo - Ford Aerospace Corporation/Houston
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Ronnie Killough - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <string.h>
#include <math.h>
#include <wex/EXmsg.h>
#include <constants.h>
#include <disp.h>
#include <DDdisp.h>
#include <DDplot.h>
#include <errno.h>


extern struct dm_shmemory   *Dm_Address;    /* ptr to DM shared memory      */


void lim_ln(disp_num, plot_ptr, line_ptr)

    short   disp_num;
    struct plot_ptrs *plot_ptr;
    struct lim_lines *line_ptr;
{

    double  sqrt();                 /* square root function for logs        */

    XPoint      points[101];        /* vertices of limit line               */
    Display     *xdisplay;          /* ptr to X display in DM shared memory */
    Window      xwindow;            /* XID of display window                */
    GC          gc;                 /* XID of GC in DM shared memory        */
    XGCValues   *gc_val;            /* ptr to GC values in DM shared memory */

    struct axis_info    *x_ptr,     /* ptr to x axis records                */
                        *y_ptr;     /* ptr to y axis records                */

    double  xvalue, yvalue,         /* used for sqrt function calculation   */
            coeffs[6],              /* contains double format of coeffs     */
            axis_xmax, axis_xmin,   /* local high/low axis scale values     */
            axis_ymax, axis_ymin,
            axis_ylow, axis_xlow,   /* lower of low/hi scale values         */
            factor_x, factor_y;     /* coordinate transformation factors    */

    float   xpoints[101],           /* world coordinate points              */
            ypoints[101],
            xpoint, ypoint,         /* used in world coord pt calculations  */
```

```
            sqrtval,                /* temp holder for result of sqrt       */
            xscaler, yscaler;       /* ratio of display scale to axis scale */

    unsigned long   gc_mask;        /* mask for GC change values            */

    int             count;          /* index into point arrays              */

    short   point_num,              /* convenience local for # point pairs  */
            polyn_num,              /* convenience local for # coefficients */
            xmultiply = 1,          /* used to temporarily change value to  */
            ymultiply = 1;          /* either positive or negative for      */
                                    /* processing lo values which are       */
                                    /* higher than the hi value             */

/*
 *  Set up local X variables and transformation factors
 */

    xdisplay = Dm_Address->xdisplay[disp_num];
    xwindow = XtWindow(plot_ptr->draw_win);
    gc_val = &Dm_Address->gc_val[disp_num];
    gc = Dm_Address->gc[disp_num];

    factor_x = plot_ptr->plot_pos->factor_x;
    factor_y = plot_ptr->plot_pos->factor_y;

/*
 *  Set up misc local variables
 */

    point_num = line_ptr->point_num;
    polyn_num = line_ptr->polyn_num;

/*
 *  Get the axis information
 */

    x_ptr = plot_ptr->axis + (line_ptr->xaxis_num - 1);
    y_ptr = plot_ptr->axis +
                (line_ptr->yaxis_num + plot_ptr->header->xaxes_num - 1);

    axis_xmax = x_ptr->high_value;
    axis_xmin = x_ptr->low_value;
    axis_ymax = y_ptr->high_value;
    axis_ymin = y_ptr->low_value;

/*
 *  Set the line parameters
 */

    if (gc_mask = set_gc(xdisplay, gc, gc_val, line_ptr->line_color, 1, 1.0,
                         NO_CHANGE, NO_CHANGE, NO_CHANGE, NO_CHANGE))
        XChangeGC(xdisplay, gc, gc_mask, gc_val);


/*
 *  If line type is Point pairs, vertices of line are given
 *  as x/y world coordinates.
 */

    if (line_ptr->line_def == 'P') {

/*
 *      Copy points from plot structure into local points array
```

```
    */

        if (x_ptr->scal_type == 'T')

            for (count = 0; count < point_num; count++)
                xpoints[count] =
                    (float) p_atimei((line_ptr->plot_pts_ptr + count)->point_x);

        else if (x_ptr->scal_type == 'N')

            for (count = 0; count < point_num; count++)
                sscanf((line_ptr->plot_pts_ptr + count)->point_x,
                                                "%f", &xpoints[count]);

        if (y_ptr->scal_type == 'T')

            for (count = 0; count < point_num; count++)
                ypoints[count] =
                    (float) p_atimei((line_ptr->plot_pts_ptr + count)->point_y);

        else if (y_ptr->scal_type == 'N')

            for (count = 0; count < point_num; count++)
                sscanf((line_ptr->plot_pts_ptr + count)->point_y,
                                                    "%f", &ypoints[count]);

    /*
     *      Convert the x and y axis scale to a 100 x 100 window
     */

        if (x_ptr->axis_type == POLAR && y_ptr->axis_type == POLAR) {
            xscaler = 50.0 / axis_xmax;

    /*
     *          Y scale value is 360, the radius
     *          of this is 2PI => 2PI/2PI = 1.
     */

            yscaler = 1.0;

        } else {    /* LOGARITHMIC or CARTESIAN */

            xscaler = 100.0 / (axis_xmax - axis_xmin);
            yscaler = 100.0 / (axis_ymax - axis_ymin);
        }

    /*
     *      Calculate the x axis coordinates
     */

        if (x_ptr->axis_type == CARTESIAN) {
            for (count = 0; count < point_num; count++)
                points[count].x = (short) ((xpoints[count] - axis_xmin)
                                            * xscaler * factor_x);

        } else if (x_ptr->axis_type == POLAR && y_ptr->axis_type == POLAR) {

            for (count = 0; count < point_num; count++) {
                xpoint = xpoints[count];
                ypoint = ypoints[count] * (PI / 180.0);
                points[count].x = (short) ((xpoint * xscaler)
                    * (float)cos((double) ypoint * yscaler) + 50.0 * factor_x);
                points[count].y = (short) ((100.0 - (xpoint * xscaler)
                    * (float)sin((double) ypoint * yscaler) + 50.0) * factor_y);
```

```
        }

    } else if (x_ptr->axis_type == LOGARITHMIC) {

/*
 *          If low value is greater than the high value, set multiply to -1
 *          so that while processing log values we don't take the sqrt of a
 *          negative number.
 */

            if (axis_xmin > axis_xmax)
                xmultiply = -1;
            else
                xmultiply = 1;

            for (count = 0; count < point_num; count++)
                xpoints[count] = xpoints[count] * xmultiply;

            axis_xmin = axis_xmin * xmultiply;
            axis_xmax = axis_xmax * xmultiply;

            for (count = 0; count < point_num; count++) {

                if (xpoints[count] == 0.0)
                    xvalue = 0.0;
                else
                    xvalue = (double) xpoints[count] - axis_xmin;

                points[count].x = (short) ((((float) sqrt((double) xvalue))
                                        / x_ptr->logval * 100.0) * factor_x);
            }
        }


/*
 *      Calculate the y axis coordinates
 */

        if (y_ptr->axis_type == LOGARITHMIC) {

            if (axis_ymin > axis_ymax)
                ymultiply = -1;
            else
                ymultiply = 1;

            for (count = 0; count < point_num; count++)
                ypoints[count] = ypoints[count] * ymultiply;

            axis_ymin = axis_ymin * ymultiply;
            axis_ymax = axis_ymax * ymultiply;

            for (count = 0; count < point_num; count++) {

                if (ypoints[count] == 0.0)
                    yvalue = 0.0;
                else
                    yvalue = (double) ypoints[count] - axis_ymin;

                points[count].y = (short)
                            (100.0 - (((float) sqrt((double) yvalue))
                                    / y_ptr->logval * 100.0) * factor_y);
            }

        } else if (y_ptr->axis_type == CARTESIAN) {
```

```
            for (count = 0; count < point_num; count++)
                points[count].y = (short) ((100.0 - (ypoints[count] - axis_ymin)
                                        * yscaler) * factor_y);
        }

/*
 *      Draw the nominal/limit line
 */

        if (point_num > 0)
            XDrawLines(xdisplay, xwindow, gc,
                               points, point_num, CoordModeOrigin);

/*
 *  If line type is coEfficient, calculate the vertices of the
 *  nominal/limit line and draw the line.
 */

    } else if (line_ptr->line_def == 'E') {

        yscaler = fabs(100.0 / (axis_ymax - axis_ymin));
        xscaler = fabs((axis_xmax - axis_xmin) / 100.0);

/*
 *      Convert the coefficients from string to double
 */

        for (count = 0; count < polyn_num; count++)
            sscanf(line_ptr->coeff[count], "%lf", &coeffs[count]);

/*
 *      Calculate the points along the line varying x from 0 to 100
 */

        if (x_ptr->axis_type == CARTESIAN)

            for (count = 0; count <= 100; count++)
                xpoints[count] = (float) count;

        else if (x_ptr->axis_type == LOGARITHMIC) {

            for (count = 0; count <= 100; count++) {
                xpoints[count] = (short) count;

                xpoints[count] = ((float) sqrt((double) xpoints[count]))
                                        / 10.0 * 100.0;
            }

        }

/*
 *      Calculate y axis points for coefficent line
 */

        if (y_ptr->axis_type == CARTESIAN) {

            if (axis_ymin > axis_ymax)
                axis_ylow = axis_ymax;
            else
                axis_ylow = axis_ymin;

            if (axis_xmin > axis_xmax)
                axis_xlow = axis_xmax;
```

```
        else
            axis_xlow = axis_xmin;

        if (axis_xmin > axis_xmax) {

            if (axis_ymin > axis_ymax) {

                for (count = 0; count <= 100; count++) {

                    ypoints[100 - count] = (axis_ymin - axis_ymax
                                - add_pt(xscaler * xpoints[count]
                                + (float) axis_xlow, coeffs, polyn_num))
                                * yscaler;

                }

            } else {

                for (count = 0; count <= 100; count++) {
                    ypoints[100 - count] =
                                (add_pt(xscaler * xpoints[count]
                                + (float) axis_xlow, coeffs, polyn_num)
                                - (float) axis_ylow) * yscaler;
                }

            }

        } else {

            if (axis_ymin > axis_ymax) {

                for (count = 0; count <= 100; count++) {
                    ypoints[count] = (axis_ymin - axis_ymax -
                            add_pt(xscaler * xpoints[count]
                                + (float) axis_xlow, coeffs, polyn_num))
                                * yscaler;
                }

            } else {

                for (count = 0; count <= 100; count++) {
                    ypoints[count] =
                            (add_pt(xscaler * xpoints[count]
                                + (float) axis_xlow, coeffs, polyn_num)
                            - (float) axis_ylow) * yscaler;
                }

            }

        }

    } else if (y_ptr->axis_type == LOGARITHMIC) {

        if (axis_ymin > axis_ymax)
            axis_ylow = axis_ymax;
        else
            axis_ylow = axis_ymin;

        if (axis_xmin > axis_xmax)
            axis_xlow = axis_xmax;
        else
            axis_xlow = axis_xmin;

        if (axis_xmin > axis_xmax) {
```

```
    if (axis_ymin > axis_ymax) {

        for (count = 0; count <= 100; count++) {

            ypoints[100 - count] = (axis_ymin - axis_ymax
                        - add_pt(xscaler * xpoints[count]
                        + (float) axis_xlow, coeffs, polyn_num))
                        * yscaler;

            if ((ypoints[100 - count] - axis_ylow) < 0.0)
                ypoints[100 - count] = -1.0;
            else {
                sqrtval = (float)
                    sqrt((double) ypoints[100-count] - axis_ylow);
                ypoints[100 - count] = sqrtval / 10.0 * 100.0;
            }
        }

    } else {

        for (count = 0; count <= 100; count++) {

            ypoints[100 - count] =
                (add_pt(xscaler * xpoints[count]
                            + (float) axis_xlow, coeffs, polyn_num)
                        - (float) axis_ylow)
                * yscaler;

            if ((ypoints[100 - count] - axis_ylow) < 0.0)
                ypoints[100 - count] = -1.0;
            else {
                sqrtval = (float)
                    sqrt((double) ypoints[100-count] - axis_ylow);
                ypoints[100 - count] = sqrtval / 10.0 * 100.0;
            }

        }

    }

} else {

    if (axis_ymin > axis_ymax) {

        for (count = 0; count <= 100; count++) {

            ypoints[count] = (axis_ymin - axis_ymax
                        - add_pt(xscaler * xpoints[count]
                        + (float) axis_xlow, coeffs, polyn_num))
                        * yscaler;

            if ((ypoints[count] - axis_ylow) < 0.0)
                ypoints[count] = -1.0;
            else {
                sqrtval = (float) sqrt((double) ypoints[count]
                                        - axis_ylow);
                ypoints[count] = sqrtval / 10.0 * 100.0;
            }

        }

    } else {
```

```
                    for (count = 0; count <= 100; count++) {
                        ypoints[count] = (add_pt(xscaler * xpoints[count]
                                        + (float) axis_xlow, coeffs, polyn_num)
                                        - (float) axis_ylow) * yscaler;

                        if ((ypoints[count] - axis_ylow) < 0.0)
                            ypoints[count] = -1.0;
                        else {
                            sqrtval = (float) sqrt((double) ypoints[count]
                                                        - axis_ylow);
                            ypoints[count] = sqrtval / 10.0 * 100.0;
                        }

                    }

                }

            }
        }

/*
 *      Transform the points to pixel coordinates
 */

        for (count=0; count <= 100; count++) {
            points[count].x = (short) (xpoints[count] * factor_x);
            points[count].y = (short) ((100.0 - ypoints[count]) * factor_y);
        }

/*
 *      Draw the line.  There are 101 points since the limit line is
 *      formed from a function varying x from 0..100.
 */

        if (polyn_num > 0)
            XDrawLines(xdisplay, xwindow, gc, points, 101, CoordModeOrigin);

    }

    return;
}
```

```
/*****************************************************************************
 * MODULE NAME: limit_val.c
 *
 *  This function verifies a limit value in scientific notation.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  K. Noonan - Ford Aerospace Corporation/Houston
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <const.h>
#include <wex/EXmsg.h>


int limit_val ( limit )

    char            limit[15];

{
    short           i,
                    j,
                    decimal;

/*
 *  If limit is null, consider it valid
 */

    if ( limit[0] == 0 )
        return ( YES );

    i = 0;

/*
 *  First character may be "+" or "-"
 */

    if ( ( limit[i] == '+' ) || ( limit[i] == '-' ) ) {
        i++;

/*
 *      If first character is a sign, the next character must be a digit
 */

        if ( ( limit[i] >= '0' ) && ( limit[i] <= '9' ) )
            i++;
        else
            return ( NO );/* exit with invalid code */
    }

/*
 *  Search for digits or decimal point until possible "E" is reached
 */

    while ( ( ( limit[i] >= '0' ) && ( limit[i] <= '9' ) ) || ( ( limit[i] == '.' ) ) )
```

```
        i++;

/*
 *  Now make sure there is a decimal point before the "E"
 */

    decimal = 0;
    for ( j = 0; j < i; j++ ) {
        if ( limit[j] == '.' )
            decimal++;/* count the number of decimal points */

    }

/*
 *  Make sure there is only one decimal point
 */

    if ( decimal != 1 )
        return ( NO );/* exit with invalid code */

    if ( ( limit[i] == 'E' ) || ( limit[i] == 'e' ) )
        limit[i] = 'E';/* force "e" to upper case */
    else
        return ( NO );/* if not an "E" call it invalid */

/*
 *  A digit must precede the "E" to be valid
 */

    if ( ! ( limit[i - 1] >= '0' ) && ( limit[i - 1] <= '9' ) )
        return ( NO );/* exit with invalid code */

/*
 *  A sign followed by two digits must follow the "E" to be valid
 */

    i++;
    if ( ( limit[i] == '+' ) || ( limit[i] == '-' ) )
        i++;
    else
        return ( NO );/* if not a sign, call it invalid */

    if ( ( limit[i] >= '0' ) && ( limit[i] <= '9' ) &&
         ( limit[i + 1] >= '0' ) && ( limit[i + 1] <= '9' ) )
        i++;
    else
        return ( NO );

/*
 *  Make sure next character is a null if limit is less than 14 characters
 */


    i++;
    if ( i < 14 ) {
        if ( limit[i] == 0 )
            return ( YES );
        else
            return ( NO );
    } else
        return ( YES );
}
```

```
/*****************************************************************************
 * MODULE NAME: list_files.c
 *
 *  This function builds a list of limit or plot files and allows the user to
 *  turn a plot/limit file on or off.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  D. Rice          - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <X11/Intrinsic.h>
#include <constants.h>
#include <disp.h>
#include <pf_key.h>
#include <stdio.h>
#include <wex/EXmsg.h>


extern Widget               Top;
extern struct dm_shmemory   *Dm_Address;
extern struct file_info     *Disp_Info;     /* ptr to file information          */
extern struct pfkey_defs    Current_Com;    /* Current command structure        */

extern int                  Num_plot,
                            Num_limit;

extern char                 **List_plot,
                            **List_limit;



int list_files ( limit_list, hist_flag )

    int             limit_list,
                    hist_flag;
{
    struct file_info            *d_info_ptr;    /* ptr to file information          */

    short                       error,          /* return function value            */
                                i;              /* index counter                    */

    int                         flag,
                                num_disps;

    char                        *s, *s1,
                                **ptr,
                                filename[50],
                                *malloc();

    D(printf("START list_files\n"));
/*
 *  Call read_files to read the directory of limit or plot files.
 */
```

```
    if (  limit_list && List_limit == NULL  ||
         !limit_list && List_plot  == NULL ) {
        num_disps = read_files ( limit_list );

        if ( num_disps == ERROR )
            return ( ERROR );
        else if ( num_disps == 0 ) {
            if ( limit_list == YES )
                tui_msg ( M_WHITE, "No Limit Files Found" );
            else
                tui_msg ( M_WHITE, "No Plot Files Found" );
            return ( 0 );
        }

/*
 *      Format the names into a list of character strings.
 */

        d_info_ptr = Disp_Info;
        ptr = (char **)malloc ( num_disps * sizeof ( char * ) );
        for ( i = 0; i < num_disps; i++ ) {
            *(ptr+i) = malloc ( 30 );
            strcpy ( *(ptr+i), d_info_ptr->name     );
            strcat ( *(ptr+i), d_info_ptr->act_flag );
            d_info_ptr++;
        }
        if ( limit_list ) {
            Num_limit  = num_disps;
            List_limit = ptr;
        } else {
            List_plot  = ptr;
            Num_plot   = num_disps;
        }
        free ( (char *)Disp_Info );

/*
 *  Otherwise, list is already set up, so set (ptr) to the right list.
 */

    } else {
        ptr       = ( limit_list ) ? List_limit : List_plot;
        num_disps = ( limit_list ) ? Num_limit  : Num_plot;
    }

/*
 *  If the history table parameter is set, then the only required action is to set up
 *  the list of filenames. In this case, return now.
 */

    if ( hist_flag )
        return ( 0 );

/*
 *  Present the list of names to the user and wait for a response.
 */

    s  = ( limit_list ) ? "List Limit Files" : "List Plot Files";
    s1 = ( limit_list ) ? "Limit Files"      : "Plot Files";

    flag = tui_get_list ( Top, ptr, num_disps, filename, s, s1, 1, -1, NULL, 0 );

/*
 *  If the user canceled the pop up, return.
```

```
*/

    if ( flag == 0 )
        return ( 0 );

/*
 *  Remove any trailing blanks from the filename.
 */

    strncpy ( Current_Com.disp_name, filename, 8 );
    if ( s = index ( Current_Com.disp_name, ' ' ) )
        *s = '\0';
    else
        Current_Com.disp_name[8] = '\0';

/*
 *  Scan the list for a match. If none found, generate a warning and return.
 */

    for ( i = 0; i < num_disps; i++ ) {
        if ( strncmp ( *ptr, filename, DNAME_LEN ) == 0 )
            break;
        ptr++;
    }

/*
 *  Set action to ON or OFF based on the selection of the user.
 */

    if ( flag == 1 ) {
        if ( limit_list )
            Current_Com.func_no = LIM_GRP;
        else
            Current_Com.func_no = PLOT;
    } else {
        if ( limit_list )
            Current_Com.func_no = LIM_GRP_OFF;
        else
            Current_Com.func_no = PLOT_OFF;
    }

/*
 *  Call appropriate function to initialize the limit group or the plot.
 */

    Current_Com.prompt_flag = NO;
    if ( limit_list )
        error = lim_grp  ( );
    else
        error = get_plot ( );

    if ( error != -1 ) {
        s = *ptr;
        s += 11;
        if ( *s == 'I' )
            strcpy ( s, "ACTIVE"   );
        else
            strcpy ( s, "INACTIVE" );
    }

/*
 *  Normal return.
 */
```

```
    D(printf("END list_files\n"));
    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: main.c
 *
 *  This function is the main controller for the Display Manager.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  K. Noonan       - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <constants.h>
#include <wex/EXmsg.h>


extern Widget   Top;
extern short    Dm_Halt;


int main ( argc, argv )

    int             argc;

    char           **argv;
{
    XEvent          event;

    int             error = 0;

    D(printf("START main\n"));
/*
 *  Call the initialization routine to initialize shared memory, data files, and all
 *  other initialization not concerned with graphics.
 */

    error = init ( );

/*
 *  Initialize the X Toolkit and the bulk of the user interface.
 */

    if ( error == 0 )
        error = ui_init ( argc, argv );

/*
 *  If no error occured during initialization, then loop forever processing toolkit
 *  events.
 */

    if ( error == 0 ) {
        Dm_Halt = 0;
        while ( Dm_Halt == 0 ) {
            XtNextEvent      ( &event );
```

```
            XtDispatchEvent ( &event );
        }
    }

/*
 *  Perform all necessary cleanup.
 */

    cleanup ( );

    D(printf("END main\n"));
    exit ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: new_disp.c
 *
 *  This function brings up the DTE emulator task or a new
 *  display.  If a new display other than the DTE display is selected, then
 *  the current window is resized to the size of the window specified in the
 *  display definition file.  A new displayer task is started, if one hasn't
 *  already been started for this display manager.  The PF key file for this
 *  display is read in also if one exists.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  K. Noonan       - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <signal.h>
#include <fcntl.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include <constants.h>
#include <disp.h>
#include <DDdisp.h>
#include <pf_key.h>
#include <wex/EXmsg.h>
#include <wex/EXexec.h>
#include <wex/EXWEX.h>
#include <wex/EXerror.h>


extern Widget                Top,
                             Scrl_Win,
                             Draw_Win,
                             Pb_Pf;
extern struct dm_shmemory    *Dm_Address;
extern struct data_info      *Dh_Address;
extern struct pfkey_defs     Current_Com;
extern struct msid_ent       *Msid;
extern struct fg_file_header     *Ffile;


extern short                 Disp_Num,       /* display number of this task          */
                             Pbi_Num,        /* Number of Pbi's in the display       */
                             Good_Strm,      /* set to YES if data type is valid     */
                             Flt_Selected,   /* Yes, if flight and data type input   */
                             Pixels[128];    /* Display Builder to Colormap index    */

extern int                   Pbi_Env_Id,     /* Pbi enviroment id for this display   */
                             Msid_num_lim,
                             Msid_num_ddd,
                             Num_ht,
                             Screen_color,
                             errno;          /* error return value                   */
```

```
extern char             Disp_Path[DNAME_LEN],   /* display path              */
                        Plot_Path[DNAME_LEN],
                        **List_ht,
                        **Msid_list_ddd,
                        **Msid_list_lim;


int new_disp ( )
{
    register int        i;

    struct disp_info    *display;           /* ptr to display information    */

    struct shm_decom    *decom_buffer;

    double              width,              /* width  of display             */
                        height;             /* height of display             */

    int                 error = 0,          /* return value from a function call */
                        access,             /* access restriction code       */
                        pid;                /* DTE process Id                */

    FILE                *fopen ( ),
                        *fp;                /* pointers to the display bg file */

    char                disp_fn[DNAME_LEN+4];/* display name                 */

    D(printf("START new_disp\n"));
/*
 *  Search the display information table in the Display Manager Shared Memory for the
 *  first unused slot.  The entry number will become the new display's number.
 */

    Disp_Num = 0;

/*
 *  RLK 9/19/90 Commented out temporarily until all code can handle multiple displays.
 *              At that time, make lower-case, non-global disp_num, need to setup
 *              concept of "current display".

    while (!found && Disp_Num < MAX_DISP) {

        if ( Dm_Address->display[Disp_Num].disp_active == NO )
            found = YES;
        else
            Disp_Num++;

    }

    if ( !found ) {
        tui_msg ( M_YELLOW, "Number of maximum displays exceeded in this workstation" );
        return ( -1 );
    }
*/

/*
 *  Get the process Id of the Display Manager associated with this display
 */

    if ( ( Dm_Address->dm_pid[Disp_Num] = getpid ( ) ) == -1 )
        tui_msg ( M_YELLOW, "Error %d in getpid call", errno );

/*
 *  If a flight has been selected and the event trigger file has not
```

```
 *  been read by the Data Handler, signal the Data Handler to read
 *  the event trigger file and wait for an acknowledgement.
 */

    if (Flt_Selected == YES && Dm_Address->process.dh_ack_evnt == NO &&
            Dm_Address->process.dh_evnt != YES) {
        Dm_Address->process.dh_num = Disp_Num;
        Dm_Address->process.dh_evnt = YES;
        error = chk_flg ( &Dm_Address->process.dh_ack_evnt, 5, 1 );

        if (error) {
            tui_msg(M_YELLOW,"Data Handler not able to process initialization files");
            Dm_Address->process.dh_ack_evnt = YES;
            Dm_Address->process.dh_evnt = NO;
        }
    }

/*
 * If the DTE emulator was selected to be started, then call EXexec to start
 * the DTE task.  If an error occurs, then advise.
 */

    if ( ( strcmp ( Current_Com.disp_name, "DTE DISPLAY" ) == 0 ) ||
        ( strcmp ( Current_Com.disp_name, "/WEX/Exec/FCdte" ) == 0 ) ||
        ( strcmp ( Current_Com.disp_name, "FCdte" ) == 0 ) ) {

        if ( ( pid = fork ( ) ) == 0 ) {
            execl ( "/WEX/Exec/FCdte", "FCdte", 0 );
            exit ( );
        }
        if ( pid == -1 ) {
            tui_msg ( M_YELLOW, "Unable to start DTE Emulator task" );
            return ( -1 );
        } else {
            return ( 1 );
        }
    }

/*
 *  Generate the path name of the display. If the display name in the current
 *  command structure already has a path name ( if it contains a "/" ), then
 *  copy the display name into the display information in shared memory and
 *  into a local variable.  If no path name, copy the display path name from
 *  globals and add the display name. Store the name into shared memory.  For
 *  the display name to be used in reading the background file, add the ".bg"
 *  extension.
 */

    display = &Dm_Address->display[Disp_Num];

    if ( Current_Com.disp_name[0] != '/' ) {
        strcpy ( disp_fn, Disp_Path );
        strcat ( disp_fn, Current_Com.disp_name );
        strcpy ( display->display_name, disp_fn );
    } else {
        strcpy ( display->display_name, Current_Com.disp_name );
        strcpy ( disp_fn, Current_Com.disp_name );
    }

    strncat ( disp_fn, ".bg\0", 4 );

/*
 *  Read the background table for the display size and the access restriction
 *  flag.  Then close the file.
```

```
 */

    if ( ( fp = fopen ( disp_fn, "r" ) ) == NULL ) {
        tui_msg ( M_YELLOW, "Error %d on open of <%s>", errno, disp_fn );
        return ( -1 );
    }

    fscanf ( fp, "%*53c" );
    fscanf ( fp, "%lf", &width );
    fscanf ( fp, "%lf", &height );
    fscanf ( fp, "%d", &Screen_color );
    Screen_color = Pixels[Screen_color];
    fscanf ( fp, "%*15c" );
    fscanf ( fp, "%d", &access );
    fclose ( fp );

/*
 *   Check the access restriction code to see if the display is either a
 *   Medical or Payload restricted display.  If the display is access
 *   restricted and the position Id does not match the access restriction, then
 *   exit out of this routine.
 */

    if ( chk_res ( access, display->pos_id ) )
        return ( -1 );

/*
 *  Set flags for the data and display handler.
 */

    display->dh_disp_init = NO;
    display->dh_new_disp  = YES;
    display->disp_pause   = NO;
    display->new_display  = YES;

/*
 *  Initialize the new display
 */

    init_disp ( Disp_Num );

/*
 *  Set the menu item which allows the display to be paused/restarted to PAUSE, as
 *  the new display will be active.
 */

    set_label ( Pb_Pf, "Freeze Display" );

/*
 *  Check the initialization flag to insure the data handler has properly initialized.
 */

    error = chk_flg ( &display->dh_disp_init, 100, 1 );
    if ( error ) {
        tui_msg ( M_YELLOW, "Data Handler not initialized to new display" );
        display->clear = YES;
        clear ( Disp_Num );
        return ( -1 );
    }

/*
 *  Read the function keys.
 */
```

```
    read_pf ( NO, display->display_name );
    display->active_display = YES;

/*
 *  Build a list of the MSID's available for updating limits. Note that only MSID's with
 *  a type of 'D', 'E', or 'F' are valid for limit sensing.
 */

    if ( ( Msid_list_lim = (char **)malloc ( Ffile->Entry_Num * sizeof ( char * ) ) )
            == NULL ) {
        tui_msg ( M_YELLOW, "Could not allocate memory for list" );
        return ( -1 );
    }

    decom_buffer = (struct shm_decom *)((char *)Dh_Address + Dh_Address->decom_buf);
    Msid_num_lim = 0;
    for ( i = 0; i < Ffile->Entry_Num; i++ )
        if ( (decom_buffer+i)->attribute == 'D' || (decom_buffer+i)->attribute == 'E' ||
             (decom_buffer+i)->attribute == 'B' ) {
            *(Msid_list_lim+Msid_num_lim) = (Msid+i)->MSID;
            Msid_num_lim++;
        }

/*
 *  Build a list of the MSID's available for updating DDD status. Note that I am assuming
 *  that the only DDD MSIDs are ones with either ddd flag set.
 */

    if ( ( Msid_list_ddd = (char **)malloc ( Ffile->Entry_Num * sizeof ( char * ) ) )
            == NULL ) {
        tui_msg ( M_YELLOW, "Could not allocate memory for list" );
        return ( -1 );
    }

    Msid_num_ddd = 0;
    for ( i = 0; i < Ffile->Entry_Num; i++ )
        if ( (Msid+i)->ddd0_latch || (Msid+i)->ddd1_latch  ) {
            *(Msid_list_ddd+Msid_num_ddd) = (Msid+i)->MSID;
            Msid_num_ddd++;
        }

/*
 *  Set the number of limit files for history tables to zero.
 */

    if ( List_ht )
        free ( List_ht );
    Num_ht = 0;

/*
 *  Read the foreground file to see if any PBI's are present
 *  and call a routine to set up the newest pbi environment.
 */

    if ( read_pbi ( ) )
        return ( -1 );

    if ( Pbi_Num > 0 ) {
        Pbi_Env_Id = pbi_setup ( Dm_Address->display[Disp_Num].flight_id,
                                 Dm_Address->display[Disp_Num].strm_type );
        if ( Pbi_Env_Id < 1 ) {
            Pbi_Num = 0;
            tui_msg ( M_YELLOW, "Error %d on PBI Environment Init", errno );
        }
```

```
    }

    D(printf("END new_disp\n"));
    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: org_file.c
 *
 *  This function checks for the existance of a plot data file (.pdt).
 *  If one exists, it is opened and the header and decom information is
 *  read.  All data points are then plotted to initialize the plot.
 *
 *  Because all displays on a single workstation will use the same set of
 *  plot data files, when a display is started containing a plot, it is
 *  possible that another display on the same workstation is also plotting
 *  the same data.  If so, the plot shown in the new display must be origined.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Tod Milam       - Ford Aerospace Corporation/Houston
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Ronnie Killough - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <X11/Xlib.h>
#include <fcntl.h>
#include <constants.h>
#include <disp.h>
#include <DDdisp.h>
#include <DDplot.h>
#include <wex/EXmsg.h>


extern struct dm_shmemory   *Dm_Address;    /* ptr to DM shared memory      */
extern short    End_of_file;                /* plot data file EOF flag      */
extern int      errno;                      /* system error return value    */


org_file(disp_num, plot_ptr)

    short   disp_num;                       /* effective display number     */

    struct plot_ptrs *plot_ptr;             /* ptr to effective plot record */

{
    char    *malloc();                  /* get malloc as a pointer          */

    struct shm_decom    *decom_ptr;     /* ptr thru plot decom structure    */
    struct plot_hdr     *plot_hdr_ptr;  /* ptr thru plot header             */

    int     j, m,                       /* loop count variables             */
            next_offset,                /* # bytes for offset cal.          */
            plot_fp;                    /* local plot file pointer          */

    short   match;
    char    plot_name[DNAME_LEN + 5];   /* plot data file name              */


    D(printf("START org_file\n"));
```

```
    j = 0;
    match = NO;

/*
 *  Copy plot name from memory and store into local variable.
 */

    strcpy(plot_name, plot_ptr->plot_name);

/*
 *  Check to be sure plot is not already active
 */

    while ((match == NO) && (j < MAX_PLOTS)) {

        if ((strcmp(Dm_Address->plots.act_plots[j], plot_name)) == 0)
            match = YES;

        j++;
    }


/*
 *  If plot is not active, open the plot data file
 *  and read the header and decom information.
 */

    if (match == NO) {

/*
 *      Open plot data file and read decom buffer
 */

        strcpy(plot_name, plot_ptr->plot_data_file);
        strcat(plot_name, ".pdt");
        plot_ptr->plot_fp = open(plot_name, O_RDONLY);

        if (plot_ptr->plot_fp != INVALID) {
            plot_fp = plot_ptr->plot_fp;

/*
 *          Skip over decom buffer header and read
 *          decom buffer to memory.
 */

            lseek(plot_fp, 80, 0);
            decom_ptr = plot_ptr->plt_decom;
            plot_hdr_ptr = plot_ptr->header;
            plot_ptr->buf_size = 0;
            next_offset = 0;

            for (m = 0; m < plot_hdr_ptr->msid_num; m++) {
                read(plot_fp, &decom_ptr->size, sizeof(int));
                read(plot_fp, &decom_ptr->length, sizeof(int));
                read(plot_fp, &decom_ptr->num_samps, sizeof(short));
                read(plot_fp, &decom_ptr->attribute, sizeof(char));
                read(plot_fp, &decom_ptr->error, sizeof(char));
                lseek(plot_fp, 12, 1);

#ifdef FAC
                if (decom_ptr->error != NULL) {
                    decom_ptr->num_samps = 1;
                    decom_ptr->length = 4;
                }
```

```
#endif

/*
 *              Calculate sample size, offset, and buffer size
 *              then store into memory.
 */

                decom_ptr->sample_size = decom_ptr->size / decom_ptr->num_samps;
                plot_ptr->buf_size = plot_ptr->buf_size + 2
                                        + decom_ptr->size;

                decom_ptr->offset = next_offset;
                next_offset = decom_ptr->size + decom_ptr->offset + 2;
                decom_ptr++;
            }

/*
 *          Allocate space for data buffer, and check for error.
 */

            plot_ptr->plot_data = malloc(plot_ptr->buf_size);

            if (plot_ptr->plot_data == NULL) {
                tui_msg(M_YELLOW, "Error %d on creating data buffer space", errno);
                return (-1);
            }

/*
 *          Initialize seconds elapsed time flag (used for time plot only)
 */

            plot_ptr->seconds_elapsed = 0;

/*
 *          Initialize end_of_file flag and call proc_plt() to plot
 *          data points.  Continue calls to proc_plt() until EOF, signalling
 *          all data points plotted.  Check for EOF necessary since
 *          proc_plt() and subordinate plot_msid() may exit prematurely due
 *          to an out-of-scale data point.
 */

            End_of_file = NO;

            while (End_of_file == NO)
                proc_plt(disp_num, plot_ptr);

/*
 *              Set previous active plot flag in memory and free memory
 *              of plot data buffer once data file has reach the end of
 *              the file.
 */

            plot_ptr->prev_act_flg = YES;
            free(plot_ptr->plot_data);

        }       /* end chk on data file open */
    }           /* end on match is NO */
    return (0);
}
```

```
/*****************************************************************************
 * MODULE NAME: p_atimei.c
 *
 *  This function converts a time string to the corresponding number of sec-
 *  onds.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Tod Milam - Ford Aerospace Corporation/Houston
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <math.h>
#include <string.h>
#include <constants.h>
#include <wex/EXmsg.h>


int p_atimei ( t_string )

    char        t_string[15];
{
    int         length,
                secs,
                seconds,
                minutes,
                hours,
                days,
                i, j,
                count,
                atoi ( ),
                strlen ( );

    char        temp[5];

    short       is_secs;

    double      pow ( );

    temp[1] = '\n';

    D(printf("START p_atimei\n"));
/*
 *  Check to see if the input string is all seconds.
 */

    is_secs = YES;
    length = strlen ( t_string );
    if ( t_string[length - 3] == ':' )
        is_secs = NO;
    if ( is_secs )
        seconds = atoi ( t_string );

/*
```

```
 *  Figure out the total seconds from days, hours, minutes, and seconds.
 */

    else {

/*
 *      Initialize the variables.
 */

        secs = days = hours = minutes = count = 0;

/*
 *      Figure out how many seconds are in the time.
 */

        i = length - 1;
        count = 0;
        while ( i >= 0 && t_string[i] != ':' ) {
            i--;
            count++;
        }
        if ( count > 0 ) {
            for ( j = 0; j < count; j++ )
                temp[j] = t_string[i + j + 1];
            temp[count] = '\0';
            secs = atoi ( temp );
        }

/*
 *      Figure out how many minutes are in the time.
 */

        count = 0;
        i--;
        while ( i >= 0 && t_string[i] != ':' ) {
            i--;
            count++;
        }
        if ( count > 0 ) {
            for ( j = 0; j < count; j++ )
                temp[j] = t_string[i + j + 1];
            temp[count] = '\0';
            minutes = atoi ( temp );
        }

/*
 *      Figure out how many hours are in the time.
 */

        count = 0;
        i--;
        while ( i >= 0 && t_string[i] != ':' ) {
            i--;
            count++;
        }
        if ( count > 0 ) {
            for ( j = 0; j < count; j++ )
                temp[j] = t_string[i + j + 1];
            temp[count] = '\0';
            hours = atoi ( temp );
        }

/*
 *      Figure out how many days are in the time.
```

```
 */

        count = 0;
        i--;
        while ( i >= 0 && t_string[i] != ':' ) {
            i--;
            count++;
        }
        if ( count > 0 ) {
            for ( j = 0; j < count; j++ )
                temp[j] = t_string[i + j + 1];
            temp[count] = '\0';
            days = atoi ( temp );
        }

/*
 *  Calculate the total number of seconds using days, hours, minutes and seconds.
 */

        hours   += days * 24;
        minutes += hours * 60;
        secs    += minutes * 60;
        seconds = secs;
    }

    D(printf("END p_atimei\n"));
    return ( seconds );
}
```

```
/*****************************************************************************
 * MODULE NAME: p_dataval.c
 *
 *  This function retrieves the appropriate union data type of the
 *  current plot value based on the parameter attribute for plot_msid().
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Richard Romeo - Ford Aerospace Corporation/Houston
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Ronnie Killough - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <X11/Xlib.h>
#include <wex/EXmsg.h>
#include <sys/types.h>
#include <constants.h>
#include <disp.h>
#include <DDdisp.h>


int p_dataval(decom_ptr)

    struct shm_decom *decom_ptr;
{
    extern union p_data Data;   /* local union structure for p_data        */

    switch (decom_ptr->attribute) {

        case ('E'):     /* single precision real */
            Data.ddata = Data.sfdata[0];
            break;

        case (9):       /* binary coded decimal tacan range */
        case (12):      /* binary coded decimal analog variable */
        case (14):      /* binary coded hexadecimal analog var. */
        case (16):      /* bit weighted analog variable  */
            Data.ddata = Data.uldata[0];
            break;

        case ('F'):     /* integer - signed */
        case (2):       /* integer - signed */
        case (3):       /* integer - no compliment */
        case (4):       /* integer - no compliment - overflow bit */

            if (decom_ptr->length == 16)
                Data.ddata = Data.ssdata[0];
            else
                Data.ddata = Data.sldata[0];
            break;

        case ('B'):     /* discrete */
        case (24):      /* discrete */
            Data.ddata = Data.sldata[0];
            break;
```

```
        case ('P'):     /* discrete Parent */
        case ('L'):     /* natural - unsigned */
        case (5):       /* natural - unsigned */
        case (6):       /* discrete Parent */

            if (decom_ptr->length <= 32)
                Data.ddata = Data.uldata[0];
            break;

        case (1):       /* real */

            if (decom_ptr->length == 16)
                Data.ddata = Data.ssdata[0];
            else if (decom_ptr->length == 32);
                Data.ddata = Data.sldata[0];
            break;

        case (7):       /* binary coded decimal - format x */
        case (8):       /* binary coded decimal - format y */
        case (10):      /* binary coded decimal GMT - days/hrs */
            Data.ddata = Data.usdata[0];
            break;

        case (19):      /* spacelab floating point */

            if (decom_ptr->length <= 32)
                Data.ddata = Data.uldata[0];
            break;

        default:
            break;
    }
}
```

```
/*****************************************************************************
 * MODULE NAME: DMp_itimea.c
 *
 *  This function converts a time value expressed in seconds to an ascii
 *  string containing hours, minutes, and seconds.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Tod Milam - Ford Aerospace Corporation/Houston
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <wex/EXmsg.h>


void p_itimea ( time_int, time_char )

    int             time_int;

    char            time_char[15];
{
    int             days    = 0,
                    hours   = 0,
                    minutes = 0,
                    seconds = 0,
                    total   = 0;

    D(printf("START p_itimea\n"));
/*
 *  Break out the seconds into parts.
 */

    total = time_int;

    days = ( int ) ( total / 86400 );
    total -= 86400 * days;

    hours = ( int ) ( total / 3600 );
    total -= 3600 * hours;

    minutes = ( int ) ( total / 60 );
    total -= 60 * minutes;

    seconds = total;

    hours += days * 24;

/*
 *  Put the time into the string.
 */

    if ( hours > 0 ) {
        sprintf ( time_char, "%3.3d:%2.2d:%2.2d\0", hours,
            minutes, seconds );
```

```
    } else
        sprintf ( time_char, "%2.2d:%2.2d\0", minutes, seconds );

    D(printf("END p_itimea\n"));
    return;
}
```

```
/*****************************************************************************
 * MODULE NAME: parse_cmd.c
 *
 *  This routine reads a command string definition and assigns a command
 *  number based on the content of a command line string.  In the case of
 *  and invalid string the command is set to INVALID.
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  A. Sprinkle     - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <constants.h>
#include <pf_key.h>
#include <wex/EXmsg.h>


#define TRUE    1
#define FALSE   0


int parse_cmd ( cmd_struct, cmd_string, cmd_string_length, pbi_or_pfkey, version )

    struct pfkey_defs   *cmd_struct;

    char                *cmd_string;

    int                 cmd_string_length,
                        pbi_or_pfkey,
                        version;
{
    static char     blank_char = ' ';

    static int      change_limits_args[4][2] = {
        {0, 0},
        {5, 8},
        {7, 12},
        {9, 16}
    };

    struct {
        char *string;
    } arg[20];                                  /* structure to handle an array of strings */

    char            *local_cmd_string,          /* local copy of the input command string  */
                     real_src[4];               /* real src name if PPM or EVN             */

    int              arg_num = 0,               /* number of arguments in the command      */
                     arg_pos[20],               /* array of starting positions for strings */
                     argument_length[20],       /* array of argument lengths               */
                     blank = OFF,               /* T/F for blank being the prev character  */
                     comment = OFF,             /* T/F for comment on                      */
                     critical = 0,              /* position for critical flag in the string */
                     external_command = 0,      /* flag for an external command            */
```

```
                    i,                          /* loop counter                                  */
                    int_version,
                    lower_limit_arg,
                    min_oc_offset = 6,
                    operational = 0,            /* pos. for operational flag in the string */
                    second_oc,
                    position = 0,               /* character position in the argument       */
                    string_position = 0,        /* pos. in the input string                 */
                    upd_rate,                   /* temp storage place for the update rate   */
                    upper_limit_arg,
                    valid,                      /* flag for various validity tests          */
                    len;                        /* length of a string                       */


    D(printf("START parse_cmd\n"));
    for ( i = 0; i < 20; i++ )
        arg[i].string = ( char * ) calloc ( 1, 120 );

    argument_length[arg_num] = 0;

    local_cmd_string = ( char * ) calloc ( 1, cmd_string_length );
    strncpy ( local_cmd_string, cmd_string, cmd_string_length );

/*
 *  Start of executable
 */

    if ( cmd_string_length > COMMAND_LINE ) {
        cmd_struct->valid_flag = INVALID;
        return ( INVALID );
    }
    arg_pos[0] = 0;
    cmd_struct->defined = YES;
    cmd_struct->valid_flag = VALID;

    for ( i = 0; i < cmd_string_length; i++ ) {

/*
 *      Extract comments
 */

        if ( ( i < cmd_string_length - 2 ) &&
             ( local_cmd_string[i] == '-' ) &&
             ( local_cmd_string[i + 1] == '-' ) ) {

            if ( comment == OFF )
                comment = ON;
            else
                comment = OFF;

        }
/*
 *      Extract the arguments
 */

        if ( comment == OFF ) {

            if ( local_cmd_string[i] == blank_char ) {

/*
 *              Place a null at the end of the argument
 */

                if ( ( blank == OFF ) && ( !external_command ) ) {
```

```
                    local_cmd_string[string_position] = NULL;
                    string_position++;
                    blank = ON;

                } else if ( ( external_command ) && ( blank == OFF ) ) {

                    arg[arg_num].string[position] = local_cmd_string[string_position];
                    argument_length[arg_num]++;
                    string_position++;
                    position++;

                }
            } else {

/*
 *              If blank is on, start a new argument
 */

                if ( blank == ON ) {

                    arg_num++;
                    arg_pos[arg_num] = string_position;
                    argument_length[arg_num] = 0;
                    position = 0;

                    if ( ( ( arg_num == 1 ) &&
                         ( local_cmd_string[0] == 'u' ) &&
                         ( local_cmd_string[1] == 'n' ) ) ||
                        ( ( arg_num == 1 ) &&
                         ( local_cmd_string[0] == 'm' ) &&
                         ( local_cmd_string[1] == 's' ) ) ) {

                        external_command = TRUE;

                    }
                    blank = OFF;
                }
/*
 *              Add a character to the present argument
 */

                arg[arg_num].string[position] = local_cmd_string[string_position];

                argument_length[arg_num]++;
                string_position++;
                position++;

            }

        }
    }

/*
 *   If the command is an external command
 */

    if ( external_command ) {

        cmd_struct->mesg_ptr = ( char * ) calloc ( 1, argument_length[arg_num] );
        strncpy ( cmd_struct->mesg_ptr, arg[arg_num].string, argument_length[arg_num] );

    }
/*
```

```
 *   If the command requests for a prompt
 */

    if ( ( ( cmd_string[arg_pos[arg_num]] == 'P' ) ||
         ( cmd_string[arg_pos[arg_num]] == 'p' ) ) &&
        ( argument_length[arg_num] == 1 ) ) {

        cmd_struct->prompt_flag = YES;
        arg_num--;

    }
/*
 *   Switch on the first letter of the command string
 */

    cmd_struct->valid_flag = VALID;
    switch ( arg[0].string[0] ) {

/*
 *  A c can be change group, change limits or clear display
 */

    case 'c':

        switch ( arg[0].string[1] ) {

        case 'g':

            if ( arg_num == 2 ) {

                if ( strcmp ( arg[2].string, "on" ) == 0 ) {
                    cmd_struct->action = ON;
                    cmd_struct->func_no = LIM_GRP;

                } else if ( strcmp ( arg[2].string, "off" ) == 0 ) {
                    cmd_struct->action = OFF;
                    cmd_struct->func_no = LIM_GRP_OFF;

                } else {

                    cmd_struct->defined = YES;
                    cmd_struct->valid_flag = INVALID;

                }

                valid = val_fn ( arg[1].string, YES );
                if ( valid )
                    strcpy ( cmd_struct->disp_name, arg[1].string );
                else
                    cmd_struct->valid_flag = INVALID;

            } else
                cmd_struct->valid_flag = INVALID;

            break;

       case 'h':

            if ( arg_num == 0 )
                cmd_struct->func_no = LIM_MENU;
            else {

                cmd_struct->limit_change.ol_alrm = NO;
                cmd_struct->limit_change.ol_adv = NO;
```

```
cmd_struct->limit_change.oh_alrm = NO;
cmd_struct->limit_change.oh_adv = NO;

cmd_struct->limit_change.cl_alrm = NO;
cmd_struct->limit_change.cl_adv = NO;
cmd_struct->limit_change.ch_alrm = NO;
cmd_struct->limit_change.ch_adv = NO;

if ( version > 3 )
    int_version = 3;
else
    int_version = version;

if ( strlen ( arg[1].string ) <= MSID_LENGTH )
    strcpy ( cmd_struct->limit_change.msid, arg[1].string );
else
    cmd_struct->valid_flag = INVALID;

if ( ( arg_num == change_limits_args[int_version][0] ) ||
    ( arg_num == change_limits_args[int_version][1] ) ) {

    valid = val_src ( arg[2].string, real_src );
    if ( valid == NO ) {
        tui_msg ( M_YELLOW, "Invalid data source %s", arg[2].string );
        cmd_struct->valid_flag = INVALID;
    } else {
        cmd_struct->func_no = CHG_LIM;
        if ( ( strcmp ( real_src, "PPM" ) == 0 ) ||
            ( strcmp ( real_src, "EVN" ) == 0 ) ) {
            strcpy ( cmd_struct->limit_change.src, real_src );
            strcpy ( cmd_struct->limit_change.option, arg[2].string );
        } else {
            strcpy ( cmd_struct->limit_change.src, arg[2].string );
            cmd_struct->limit_change.option[0] = 0;
        }

        if ( arg[3].string[0] == 'O' )
            operational = 3;
        else if ( arg[3].string[0] == 'C' )
            critical = 3;
        else
            cmd_struct->valid_flag = INVALID;

        if ( arg_num == change_limits_args[int_version][1] ) {

            second_oc = min_oc_offset + int_version * 2 - 2;

            if ( arg[second_oc].string[0] == 'O' )
                operational = second_oc;
            else if ( arg[second_oc].string[0] == 'C' )
                critical = second_oc;
            else
                cmd_struct->valid_flag = INVALID;

        }

/*
 *              Set up operational lower limits
 */

        if ( operational > 0 ) {

            lower_limit_arg = operational + 1;
            upper_limit_arg = operational + 1 + int_version;
```

```
                if ( limit_val ( arg[lower_limit_arg].string ) ) {
                    sscanf ( arg[lower_limit_arg].string, "%lf",
                        &( cmd_struct->limit_change.ops_ll ) );
                    cmd_struct->valid_flag = VALID;
                }

/*
 *
 */              Set up operational upper limits


                if ( limit_val ( arg[upper_limit_arg].string ) ) {
                    sscanf ( arg[upper_limit_arg].string, "%lf",
                        &( cmd_struct->limit_change.ops_ul ) );
                    cmd_struct->valid_flag = VALID;
                }

/*
 *
 */              Set up advisory and alarm flags


                if ( int_version >= 2 ) {

                    if ( ( arg[lower_limit_arg + 1].string[0] == 'Y' ) ||
                         ( arg[lower_limit_arg + 1].string[0] == 'y' ) )
                        cmd_struct->limit_change.ol_alrm = 1;
                    else
                        cmd_struct->limit_change.ol_alrm = 0;

                    if ( ( arg[upper_limit_arg + 1].string[0] == 'Y' ) ||
                         ( arg[upper_limit_arg + 1].string[0] == 'y' ) )
                        cmd_struct->limit_change.oh_alrm = 1;
                    else
                        cmd_struct->limit_change.oh_alrm = 0;

                    if ( int_version >= 3 ) {

                        if ( ( arg[lower_limit_arg + 2].string[0] == 'Y' ) ||
                             ( arg[lower_limit_arg + 2].string[0] == 'y' ) )
                            cmd_struct->limit_change.ol_adv = 1;
                        else
                            cmd_struct->limit_change.ol_adv = 0;

                        if ( ( arg[upper_limit_arg + 2].string[0] == 'Y' ) ||
                             ( arg[upper_limit_arg + 2].string[0] == 'y' ) )
                            cmd_struct->limit_change.oh_adv = 1;
                        else
                            cmd_struct->limit_change.oh_adv = 0;

                    }
                }
            }

/*
 *
 */          Assign critical lower limits


            if ( critical > 0 ) {

                lower_limit_arg = critical + 1;
                upper_limit_arg = critical + 1 + int_version;

                if ( limit_val ( arg[lower_limit_arg].string ) ) {
                    sscanf ( arg[lower_limit_arg].string, "%lf",
                        &( cmd_struct->limit_change.crit_ll ) );
                    cmd_struct->valid_flag = VALID;
                }

/*
```

```
     *                    Assign critical upper limits
     */

                          if ( limit_val ( arg[upper_limit_arg].string ) ) {
                              sscanf ( arg[upper_limit_arg].string, "%lf",
                                  & ( cmd_struct->limit_change.crit_ul ) );
                              cmd_struct->valid_flag = VALID;
                          }
    /*
     *                    Set up advisory and alarm flags
     */

                          if ( int_version >= 2 ) {

                              if ( ( arg[lower_limit_arg + 1].string[0] == 'Y' ) ||
                                  ( arg[lower_limit_arg + 1].string[0] == 'y' ) )
                                  cmd_struct->limit_change.cl_alrm = 1;
                              else
                                  cmd_struct->limit_change.cl_alrm = 0;

                              if ( ( arg[upper_limit_arg + 1].string[0] == 'Y' ) ||
                                  ( arg[upper_limit_arg + 1].string[0] == 'y' ) )
                                  cmd_struct->limit_change.ch_alrm = 1;
                              else
                                  cmd_struct->limit_change.ch_alrm = 0;

                              if ( int_version >= 3 ) {

                                  if ( ( arg[lower_limit_arg + 2].string[0] == 'Y' ) ||
                                      ( arg[lower_limit_arg + 2].string[0] == 'y' ) )
                                      cmd_struct->limit_change.cl_adv = 1;
                                  else
                                      cmd_struct->limit_change.cl_adv = 0;

                                  if ( ( arg[upper_limit_arg + 2].string[0] == 'Y' ) ||
                                      ( arg[upper_limit_arg + 2].string[0] == 'y' ) )
                                      cmd_struct->limit_change.ch_adv = 1;
                                  else
                                      cmd_struct->limit_change.ch_adv = 0;

                              }
                          }
                      }   /* end chk for valid source */

    /*
     *                Set up limit change flags
     */

                      if ( cmd_struct->valid_flag == VALID ) {

                          if ( ( operational > 0 ) && ( critical > 0 ) )
                              cmd_struct->limit_change.flag = 2;
                          else if ( critical > 0 )
                              cmd_struct->limit_change.flag = 1;
                          else if ( operational > 0 )
                              cmd_struct->limit_change.flag = 0;
                          else
                              cmd_struct->valid_flag = INVALID;

                      }
                  } else
                      cmd_struct->valid_flag = INVALID;
```

```
            }
            break;

        case 'l':

            cmd_struct->func_no = CLEAR_DISPLAY;
            break;

        default:

            cmd_struct->defined = NO;
            cmd_struct->valid_flag = INVALID;
            break;

        }
        break;

/*
 *  A d can only be disable logging
 */

    case 'd':

        switch ( arg[0].string[1] ) {

        case 'l':

            if ( arg_num == 0 )
                cmd_struct->func_no = LOGDISABLE_DISPLAY;
            else if ( ( arg_num == 1 ) && ( strncmp ( arg[1].string, "all", 3 ) == 0 ) )
                cmd_struct->func_no = LOGDISABLE_ALL;
            else
                cmd_struct->func_no = INVALID;
            break;

        default:

            cmd_struct->func_no = INVALID;
            cmd_struct->defined = NO;
            break;

        }
        break;

/*
 *  An e can be exit or enable logging
 */

    case 'e':

        switch ( arg[0].string[1] ) {

        case 'l':

            if ( arg_num == 0 )
                cmd_struct->func_no = LOGENABLE_DISPLAY;
            else if ( ( arg_num == 1 ) && ( strncmp ( arg[1].string, "all", 3 ) == 0 ) )
                cmd_struct->func_no = LOGENABLE_ALL;
            else
                cmd_struct->func_no = INVALID;
            break;

        case 'x':
```

```
            cmd_struct->func_no = HALT_DISPLAY;
            break;

        default:

            cmd_struct->func_no = INVALID;
            cmd_struct->defined = NO;
            break;

        }
        break;

/*
 *  A g can only be GDR get next command
 */

    case 'g':

        switch ( arg[0].string[1] ) {

        case 'n':

            if ( arg_num == 1 ) {

                valid = val_ppl ( arg[1].string );
                if ( valid ) {
                    cmd_struct->func_no = GDR_GETNEXT;
                    strcpy ( cmd_struct->disp_name, arg[1].string );
                } else
                    cmd_struct->valid_flag = INVALID;

            } else
                cmd_struct->func_no = INVALID;
            break;

        default:

            cmd_struct->func_no = INVALID;
            cmd_struct->defined = NO;
            break;

        }
        break;

/*
 *  An h can be help or history tabs
 */

    case 'h':

        switch ( arg[0].string[1] ) {

        case 'e':

            cmd_struct->func_no = MAIN_HELP;
            break;

        case 't':

            if ( arg_num == 2 ) {

                valid = val_fn ( arg[1].string, YES );
                if ( valid ) {
```

```
                    strcpy ( cmd_struct->disp_name, arg[1].string );
                    valid = val_fn ( arg[2].string, YES );
                    if ( valid ) {
                        cmd_struct->func_no = HIST_TAB;
                        strcpy ( cmd_struct->ovr_name, arg[2].string );
                    } else
                        cmd_struct->func_no = INVALID;

                } else
                    cmd_struct->func_no = INVALID;

            } else
                cmd_struct->func_no = INVALID;

            break;

        default:

            cmd_struct->func_no = INVALID;
            cmd_struct->defined = NO;
            break;

        }
        break;

/*
 *  An m can be pull up the main menu or send a message
 */

    case 'm':

        switch ( arg[0].string[1] ) {

        case 'm':

            cmd_struct->func_no = DRAW_MAIN;
            break;

        case 's':

            if ( arg_num >= 1 && pbi_or_pfkey == PBI )
                cmd_struct->func_no = EXMSG_SEND;
            else
                cmd_struct->func_no = INVALID;
            break;

        default:

            cmd_struct->func_no = INVALID;
            cmd_struct->defined = NO;
            break;

        }
        break;

/*
 *  An o can only be overlay a plot
 */

    case 'o':

        if ( arg[0].string[1] == 'v' ) {

            cmd_struct->func_no = PLOT_OVRLAY;
```

```
            valid = val_fn ( arg[1].string, YES );
            if ( valid )
                strcpy ( cmd_struct->disp_name, arg[1].string );
            else
                cmd_struct->valid_flag = INVALID;

            valid = val_fn ( arg[2].string, YES );
            if ( valid )
                strcpy ( cmd_struct->ovr_name, arg[2].string );
            else
                cmd_struct->valid_flag = INVALID;


            if ( cmd_struct->valid_flag ) {

                strcpy ( cmd_struct->disp_name, arg[1].string );
                strcpy ( cmd_struct->ovr_name, arg[2].string );


            }
        } else {

            cmd_struct->func_no = INVALID;
            cmd_struct->defined = NO;

        }
        break;

/*
 *  A p can be switch pos id alarm, disable PBIs, enable PBIs or
 *  start/stop a plot
 */

    case 'p':

        switch ( arg[0].string[1] ) {

        case 'a':

            if ( strcmp ( arg[1].string, "on" ) == 0 ) {
                cmd_struct->action = ON;
                cmd_struct->func_no = POS_ALARM;

            } else if ( strcmp ( arg[1].string, "off" ) == 0 ) {
                cmd_struct->action = OFF;
                cmd_struct->func_no = POS_ALARM_OFF;

            } else {
                cmd_struct->defined = YES;
                cmd_struct->valid_flag = INVALID;

            }
            break;

        case 'd':

            cmd_struct->func_no = PBI_DISABLE;
            break;

        case 'e':

            cmd_struct->func_no = PBI_ENABLE;
            break;
```

```
        case 'l':

            valid = val_fn ( arg[1].string, YES );
            if ( !valid ) {

                cmd_struct->defined = YES;
                cmd_struct->valid_flag = INVALID;

            } else {

                strncpy ( cmd_struct->disp_name, arg[1].string, argument_length[1] );
                if ( strcmp ( arg[2].string, "start" ) == 0 ) {

                    cmd_struct->func_no = PLOT;
                    cmd_struct->action = ON;

                } else if ( strcmp ( arg[2].string, "stop" ) == 0 ) {

                    cmd_struct->func_no = PLOT_OFF;
                    cmd_struct->action = OFF;

                } else {

                    cmd_struct->defined = YES;
                    cmd_struct->valid_flag = INVALID;

                }

            }
            break;

        default:

            cmd_struct->func_no = INVALID;
            cmd_struct->defined = NO;
            break;

        }
        break;

/*
 *  An s can be screen dump, select display or show pf keys
 */

    case 's':

        switch ( arg[0].string[1] ) {

        case 'c':

            cmd_struct->func_no = SCRN_DUMP;
            break;

        case 'e':

            if ( arg_num == 0 )
                cmd_struct->func_no = START_DISPLAY;
            else if ( arg_num == 1 ) {

                valid = val_fn ( arg[1].string, YES );

                if ( !valid )
                    cmd_struct->valid_flag = INVALID;
```

```
            else {

                strcpy ( cmd_struct->disp_name, arg[1].string );
                cmd_struct->func_no = START_PDISPLAY;

            }

        }
        break;

    case 'h':

        cmd_struct->func_no = DRAW_PF;
        break;

    default:

        cmd_struct->func_no = INVALID;
        cmd_struct->defined = NO;
        break;

    }
    break;

/*
 *  A u can be unlatch DDDs, a unix command or change the update rate
 */

    case 'u':

        switch ( arg[0].string[1] ) {

        case 'd':

            cmd_struct->func_no = DDD_UNLATCH;
            if ( arg_num == 2 ) {
                if ( valmsid ( arg[1].string ) ) {
                    strcpy ( cmd_struct->limit_change.msid, arg[1].string );
                    if ( val_src ( arg[2].string, real_src ) ) {
                        strcpy ( cmd_struct->limit_change.src, arg[2].string );
                        cmd_struct->action = UNLATCH_MSID;
                    } else
                        cmd_struct->func_no = INVALID;
                } else {
                    cmd_struct->func_no = INVALID;
                    tui_msg ( M_YELLOW, "Msid %s is an invalid msid name" );
                }


            } else if ( arg_num > 0 )
                cmd_struct->func_no = INVALID;
            else {
                cmd_struct->func_no = DDD_UNL_ALL;
                cmd_struct->action = ALL;
            }

            break;

        case 'n':

            if ( strlen ( arg[1].string ) > 0 && pbi_or_pfkey == PBI )
                cmd_struct->func_no = UNIX_COMMAND;
            else
                cmd_struct->func_no = INVALID;
```

```
            break;

        case 'r':

            sscanf ( arg[1].string, "%d", &upd_rate );
            if ( upd_rate >= 1 && upd_rate <= 99999 ) {
                cmd_struct->func_no = UPD_RATE;
                cmd_struct->rate = upd_rate * 1000;
            } else
                cmd_struct->valid_flag = INVALID;
            break;

        default:

            cmd_struct->func_no = INVALID;
            cmd_struct->defined = NO;
            break;

        }
        break;

/*
 *  A z can be set zoom factor, zoom display or zoom reset
 */

    case 'z':

        switch ( arg[0].string[1] ) {

        case 'f':

            if ( arg_num == 1 ) {

                valid = YES;
                len = strlen ( arg[arg_num].string );
                for ( i = 0; i < len; i++ ) {
                    if ( ( arg[arg_num].string[i] == '+' ) || ( arg[arg_num].string[i] ==
'-' ) )
                        valid = NO;
                }
                if ( valid ) {
                    valid = dec_val ( arg[arg_num].string );
                    if ( valid ) {
                        sscanf ( arg[arg_num].string, "%f", & ( cmd_struct->factor ) );
                        if ( cmd_struct->factor < .01 ) {
                            tui_msg ( M_YELLOW, "Zoom factor less than .01 - default to .0
1" );
                            cmd_struct->factor = .01;
                        } else if ( cmd_struct->factor > 9.9 ) {
                            tui_msg ( M_YELLOW, "Zoom factor greater than 9.9 - default to
 9.9" );
                            cmd_struct->factor = 9.9;
                        }
                    } else
                        tui_msg ( M_YELLOW, "Invalid zoom factor - should be between .01 a
nd 9.9" );
                } else
                    tui_msg ( M_YELLOW, "Invalid zoom factor - should be between .01 and 9
.9" );

                if ( valid )
                    cmd_struct->func_no = ZOOM_FAC;
                else
                    cmd_struct->func_no = INVALID;
```

```
            } else {

                cmd_struct->func_no = INVALID;
                cmd_struct->defined = NO;

            }
            break;

        case 'm':

            cmd_struct->func_no = ZOOM_DIS;
            break;

        case 'r':

            cmd_struct->func_no = ZOOM_RES;
            break;

        default:

            cmd_struct->func_no = INVALID;
            cmd_struct->defined = NO;
            break;

        }
        break;

/*
 *  In all other cases the function is invalid
 */

    default:

        cmd_struct->func_no = INVALID;
        cmd_struct->defined = NO;
        break;

    }

/*Debug printout ***************
    tui_msg ( M_GREEN,"func_no = %d defined = %d valid_flag = %d",
          cmd_struct->func_no,cmd_struct->defined,cmd_struct->valid_flag );
 */

    for ( i = 0; i < 20; i++ )
        free ( arg[i].string );

    free ( local_cmd_string );

    D(printf("END parse_cmd\n"));
    if ( ( cmd_struct->func_no == INVALID ) || ( cmd_struct->defined == NO ) ) {

        cmd_struct->valid_flag = INVALID;
        return ( INVALID );

    } else
        return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: pbi_cmd.c
 *
 *  This routine indicates the type of PBI command to process by
 *  finding the pbi pushed, either processing, sending a PBI msid, or setting a
 *  PBI token to accomplish the command.  Any display modifications are then
 *  sent to the displayer for modification via shared memory and a process
 *  signal.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  A. Sprinkle      - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <constants.h>
#include <pf_key.h>
#include <disp.h>
#include <wex/FCpbi.h>
#include <wex/EXmsg.h>


extern struct pbi_def   *Pbi_Def;   /* Pbi Hot index for Pbi processing */

extern short            Pbi_Hot_Ndx;/* Pbi Hot index for Pbi processing */


int pbi_cmd ( )
{
    int             modified_count; /* Number of Pbi's Modified        */

    struct pbi_def *pbi_def_ptr;    /* pointer to PBI definitions      */

    D(printf("START pbi_cmd\n"));
/*
 *  Process the host command by calling the routine for host processing
 */

    pbi_def_ptr = Pbi_Def;

    if ( strncmp ( pbi_def_ptr[Pbi_Hot_Ndx - 1].pbi_dest, PBI_HOST_DEST,
            pbi_def_ptr[Pbi_Hot_Ndx - 1].pbi_dest_len ) == 0 ) {
        if ( pbi_def_ptr[Pbi_Hot_Ndx - 1].pbi_disable == ENABLED )
            pbi_host ( );

    }

/*
 *  If the command is not a HOST targeted PBI then process in the local
 *  R ( Display Manager ) pbi process.
 */

    else {
```

```
        if ( strncmp ( pbi_def_ptr[Pbi_Hot_Ndx - 1].pbi_dest, PBI_LOCAL_DEST,
                pbi_def_ptr[Pbi_Hot_Ndx - 1].pbi_dest_len ) == 0 ) {
            modified_count = pbi_local ( );

            if ( modified_count == INVALID )
                tui_msg ( M_YELLOW, "PBI ERROR In local pbi selection" );
        } else {
            modified_count = INVALID;
            tui_msg ( M_YELLOW, "Invalid PBI Destination Specified %3s",
                  pbi_def_ptr[Pbi_Hot_Ndx - 1].pbi_dest );
            return ( 0 );
        }
    }
    D(printf("END pbi_cmd\n"));
    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: pbi_config
 *
 *  This handles changes to pbi by turning on/off backlighting, redrawing
 *  labels, and new backlighting depending on the state of the button.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Scott Zrubek    - Ford Aerospace Corporation/Houston
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Ronnie Killough - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <wex/EXmsg.h>
#include <constants.h>
#include <DDdisp.h>
#include <disp.h>
#include <DDfg_graph.h>


extern struct data_info *Dh_Address;     /* Displayer shared memory      */
extern struct ddd_ent   *Ddd;            /* ptr to Ddd entry table       */
extern struct fg_recs   Fg_rec;          /* fg graphics records          */
extern struct pbi_ent   *Pbi;            /* ptr to pbi records           */
extern struct msid_ent  *Msid;           /* ptr to msid entry table      */


int pbi_config(disp_num, redraw_rect, pbi_changed, number_of_changes)

    short   disp_num;                    /* effective display number     */

    struct pbi_redraw_rect  redraw_rect;/* redraw rectangle              */

    struct pbi_changes  *pbi_changed;    /* list of pbi changes          */

    int number_of_changes;               /* number of pbi changes        */

{
    register struct shm_decom *decom_buffer;
    register char             *ldata_buffer;

    struct fgr_record         *loc_graph_ptr;
    struct fg_line_rec        *fg_line_ptr;
    struct fg_rectangle_rec   *fg_rect_ptr;
    struct fg_polygon_rec     *fg_poly_ptr;
    struct fg_curve_rec       *fg_cur_ptr;
    struct fg_circle_rec      *fg_cir_ptr;
    struct fg_arc_rec         *fg_arc_ptr;
    struct fg_ellipse_rec     *fg_ell_ptr;
    struct fg_clkmtr_rec      *fg_clk_ptr;
    struct fg_bar_rec         *fg_bar_ptr;

    long                      new_color;

    int                       i,j;
```

```
                        grph_index;
                        pbi_index;
                        reverse_video = OFF;
                        change_flag = OFF;

    short               first_pass = 0;

    D(printf("START pbi_config\n"));
/*
 *  Process all of the OFF PBI entries
 */

    loc_graph_ptr = Fg_rec.graph_rec;

    for (i = 0; i < number_of_changes; i++) {
        pbi_index = pbi_changed[i].pbi_chg_ndx;
        grph_index = Pbi[pbi_index].grph_indx - 1;

        if (grph_index >= 0) {
            change_flag = ON;

            if (! pbi_changed[i].pbi_active_flag) {

/*
 *              If ddd or backlighting is enabled for the pbi,
 *              set new color
 */

                if (Pbi[pbi_index].ddd_indx > 0)
                    new_color = Pbi[pbi_index].grph_color;

                reverse_video = OFF;

/*
 *              Clear the offs for the pbis being set to off
 */

                switch (loc_graph_ptr[grph_index].graph_typ) {

                    case LINE:

                        fg_line_ptr = (struct fg_line_rec *)
                                      loc_graph_ptr[grph_index].graph_ptr;

                        for (j = 0; j < fg_line_ptr->label_num; j++)
                            fg_line_ptr->line_lbl_ptr->label_ind[j] = NULL;

                        fg_line_ptr->rev_video = OFF;
                        fg_line_ptr->ddd_ind = NULL;

                        break;

                    case RECTANGLE:

                        fg_rect_ptr = (struct fg_rectangle_rec *)
                                      loc_graph_ptr[grph_index].graph_ptr;

                        for (j = 0; j < fg_rect_ptr->label_num; j++)
                            fg_rect_ptr->rect_lbl_ptr->label_ind[j] = NULL;

                        fg_rect_ptr->ddd_ind = NULL;
                        fg_rect_ptr->rev_video = NO;

                        break;
```

```
case POLYGON:

    fg_poly_ptr = (struct fg_polygon_rec *)
                  loc_graph_ptr[grph_index].graph_ptr;

    for (j = 0; j < fg_poly_ptr->label_num; j++)
        fg_poly_ptr->poly_lbl_ptr->label_ind[j] = NULL;

    fg_poly_ptr->ddd_ind = NULL;
    fg_poly_ptr->rev_video = NO;

    break;

case CURVE:

    fg_cur_ptr = (struct fg_curve_rec *)
                  loc_graph_ptr[grph_index].graph_ptr;

    for (j = 0; j < fg_cur_ptr->label_num; j++)
        fg_cur_ptr->cur_lbl_ptr->label_ind[j] = NULL;

    fg_cur_ptr->ddd_ind = NULL;
    fg_cur_ptr->rev_video = NO;

    break;

case CIRCLE:

    fg_cir_ptr = (struct fg_circle_rec *)
                  loc_graph_ptr[grph_index].graph_ptr;

    for (j = 0; j < fg_cir_ptr->label_num; j++)
        fg_cir_ptr->cir_lbl_ptr->label_ind[j] = NULL;

    fg_cir_ptr->ddd_ind = NULL;
    fg_cir_ptr->rev_video = NO;

    break;

case ARC:

    fg_arc_ptr = (struct fg_arc_rec *)
                  loc_graph_ptr[grph_index].graph_ptr;

    for (j = 0; j < fg_arc_ptr->label_num; j++)
        fg_arc_ptr->arc_lbl_ptr->label_ind[j] = NULL;

    fg_arc_ptr->ddd_ind = NULL;
    fg_arc_ptr->cur_color = new_color;

    break;

case ELLIPSE:

    fg_ell_ptr = (struct fg_ellipse_rec *)
                  loc_graph_ptr[grph_index].graph_ptr;

    for (j = 0; j < fg_ell_ptr->label_num; j++)
        fg_ell_ptr->ell_lbl_ptr->label_ind[j] = NULL;

    fg_ell_ptr->ddd_ind = NULL;
    fg_ell_ptr->rev_video = NO;
```

```
                        break;

                    case CLOCK_METER:

                        fg_clk_ptr = (struct fg_clkmtr_rec *)
                                        loc_graph_ptr[grph_index].graph_ptr;

                        for (j = 0; j < fg_clk_ptr->label_num; j++)
                            fg_clk_ptr->clk_lbl_ptr->label_ind[j] = NULL;

                        fg_clk_ptr->rev_video = NO;

                        break;

                    case BAR_CHART:

                        fg_bar_ptr = (struct fg_bar_rec *)
                                        loc_graph_ptr[grph_index].graph_ptr;

                        for (j = 0; i < fg_bar_ptr->label_num; j++)
                            fg_bar_ptr->bar_lbl_ptr->label_ind[j] = NULL;

                        fg_bar_ptr->rev_video = NO;

                        break;

                    default:

                        break;

                }
            }   /* end of check for inactive pbi */
        }   /* end if grph_indx > 0 */
    }   /* end for <# of pbi changes> */

/*
 *  Process all of the ON PBI entries
 */

    for (i = 0; i < number_of_changes; i++) {
        pbi_index = pbi_changed[i].pbi_chg_ndx;
        grph_index = Pbi[pbi_index].grph_indx - 1;

        if (grph_index >= 0) {

/*
 *          If the pbi is now active
 */

            change_flag = ON;
            if (pbi_changed[i].pbi_active_flag) {

/*
 *              If ddd or backlighting is enabled for the pbi
 */

                if (Pbi[pbi_index].ddd_indx > 0) {

/*
 *                  Check to see if the Data Handler
 *                  is updating the decom buffer.
 */

                    if (Dh_Address->need_decom == YES) {
```

```
#ifdef SUN
                    usleep ( 100000 );
#else
                    astpause(0, 100);    /* NON - PORTABLE CODE */
#endif
                    return (0);
                }

                Dh_Address->decom_in_use[disp_num] = YES;

/*
 *
 *              Set up the local pointers. The decom buffer will
 *              be used to access information from the updated buffer.
 */

                decom_buffer = (struct shm_decom *) ((char *) Dh_Address +
                                    Dh_Address->decom_buf);
                ldata_buffer = (char *) ((char *) Dh_Address +
                             Dh_Address->buffer[Dh_Address->buf_ready]);

                new_color = ddd(decom_buffer, ldata_buffer,
                                Pbi[pbi_index].ddd_indx, first_pass);

                reverse_video = OFF;

            } else {

                new_color = Pbi[pbi_index].grph_color;

                if (pbi_changed[i].pbi_feedback_flag)
                    reverse_video = ON;
                else
                    reverse_video = OFF;
            }

/*
 *
 *          Assign reverse video to the proper element
 *          of the foreground records structure.
 */

            switch (loc_graph_ptr[grph_index].graph_typ) {

            case LINE:

                fg_line_ptr = (struct fg_line_rec *)
                                    loc_graph_ptr[grph_index].graph_ptr;

                fg_line_ptr->pbi_ind = pbi_index + 1;
                fg_line_ptr->label_num = Pbi[pbi_index].num_labels;

                if (fg_line_ptr->label_num > 0
                        && fg_line_ptr->line_lbl_ptr == NULL)
                    fg_line_ptr->line_lbl_ptr = (struct label_index *)
                                calloc(1, sizeof(struct label_index));

                for (j = 0; j < fg_line_ptr->label_num; j++)
                    fg_line_ptr->line_lbl_ptr->label_ind[j] =
                                    Pbi[pbi_index].label_ptr[j].index;

                fg_line_ptr->rev_video = reverse_video;
                fg_line_ptr->ddd_ind = Pbi[pbi_index].ddd_indx;

                if (Pbi[pbi_index].ddd_indx > 0) {
                    fg_line_ptr->rev_video = NO;
```

```
            reverse_video = NO;
        } else {
            fg_line_ptr->cur_color = new_color;
            fg_line_ptr->rev_video = reverse_video;
        }

        break;

    case RECTANGLE:

        fg_rect_ptr = (struct fg_rectangle_rec *)
                            loc_graph_ptr[grph_index].graph_ptr;

        fg_rect_ptr->pbi_ind = pbi_index + 1;
        fg_rect_ptr->label_num = Pbi[pbi_index].num_labels;

        if (fg_rect_ptr->label_num > 0
                && fg_rect_ptr->rect_lbl_ptr == NULL)
            fg_rect_ptr->rect_lbl_ptr = (struct label_index *)
                    calloc(1, sizeof(struct label_index));

        for (j = 0; j < fg_rect_ptr->label_num; j++)
            fg_rect_ptr->rect_lbl_ptr->label_ind[j] =
                        Pbi[pbi_index].label_ptr[j].index;

        fg_rect_ptr->ddd_ind = Pbi[pbi_index].ddd_indx;

        if (Pbi[pbi_index].ddd_indx > 0) {
            fg_rect_ptr->rev_video = NO;
            reverse_video = NO;
        } else {
            fg_rect_ptr->cur_color = new_color;
            fg_rect_ptr->rev_video = reverse_video;
        }

        break;

    case POLYGON:

        fg_poly_ptr = (struct fg_polygon_rec *)
                            loc_graph_ptr[grph_index].graph_ptr;

        fg_poly_ptr->pbi_ind = pbi_index + 1;
        fg_poly_ptr->label_num = Pbi[pbi_index].num_labels;

        for (j = 0; j < fg_poly_ptr->label_num; j++)
            fg_poly_ptr->poly_lbl_ptr->label_ind[j] =
                    Pbi[pbi_index].label_ptr[j].index;

        fg_poly_ptr->rev_video = reverse_video;

        fg_poly_ptr->ddd_ind = Pbi[pbi_index].ddd_indx;

        if (Pbi[pbi_index].ddd_indx > 0) {
            fg_poly_ptr->rev_video = NO;
            reverse_video = NO;
        } else {
            fg_poly_ptr->cur_color = new_color;
            fg_poly_ptr->rev_video = reverse_video;
        }

        break;

    case CURVE:
```

```
        fg_cur_ptr = (struct fg_curve_rec *)
                            loc_graph_ptr[grph_index].graph_ptr;

        fg_cur_ptr->pbi_ind = pbi_index + 1;
        fg_cur_ptr->label_num = Pbi[pbi_index].num_labels;

        for (j = 0; j < fg_cur_ptr->label_num; j++)
            fg_cur_ptr->cur_lbl_ptr->label_ind[j] =
                    Pbi[pbi_index].label_ptr[j].index;

        fg_cur_ptr->rev_video = reverse_video;

        fg_cur_ptr->ddd_ind = Pbi[pbi_index].ddd_indx;

        if (Pbi[pbi_index].ddd_indx > 0) {
            fg_cur_ptr->rev_video = NO;
            reverse_video = NO;
        } else {
            fg_cur_ptr->cur_color = new_color;
            fg_cur_ptr->rev_video = reverse_video;
        }

        break;


    case CIRCLE:

        fg_cir_ptr = (struct fg_circle_rec *)
                            loc_graph_ptr[grph_index].graph_ptr;

        fg_cir_ptr->pbi_ind = pbi_index + 1;
        fg_cir_ptr->label_num = Pbi[pbi_index].num_labels;

        for (j = 0; j < fg_cir_ptr->label_num; j++)
            fg_cir_ptr->cir_lbl_ptr->label_ind[j] =
                    Pbi[pbi_index].label_ptr[j].index;

        fg_cir_ptr->rev_video = reverse_video;
        fg_cir_ptr->ddd_ind = Pbi[pbi_index].ddd_indx;

        if (Pbi[pbi_index].ddd_indx > 0) {
            fg_cir_ptr->rev_video = NO;
            reverse_video = NO;
        } else {
            fg_cir_ptr->cur_color = new_color;
            fg_cir_ptr->rev_video = reverse_video;
        }
        break;

    case ARC:

        fg_arc_ptr = (struct fg_arc_rec *)
                            loc_graph_ptr[grph_index].graph_ptr;

        fg_arc_ptr->pbi_ind = pbi_index + 1;
        fg_arc_ptr->label_num = Pbi[pbi_index].num_labels;

        if (fg_arc_ptr->label_num > 0
                && fg_arc_ptr->arc_lbl_ptr == NULL)
            fg_arc_ptr->arc_lbl_ptr = (struct label_index *)
                        calloc(1, sizeof(struct label_index));

        for (j = 0; j < fg_arc_ptr->label_num; j++)
```

```
            fg_arc_ptr->arc_lbl_ptr->label_ind[j] =
                    Pbi[pbi_index].label_ptr[j].index;

        fg_arc_ptr->ddd_ind = Pbi[pbi_index].ddd_indx;

        fg_arc_ptr->rev_video = reverse_video;

        if (Pbi[pbi_index].ddd_indx > 0) {
            fg_arc_ptr->cur_color = new_color;
            fg_arc_ptr->rev_video = NO;
            reverse_video = NO;
        } else {
            fg_arc_ptr->cur_color = new_color;
            fg_arc_ptr->rev_video = reverse_video;
        }

        break;

    case ELLIPSE:

        fg_ell_ptr = (struct fg_ellipse_rec *)
                            loc_graph_ptr[grph_index].graph_ptr;

        fg_ell_ptr->pbi_ind = pbi_index + 1;
        fg_ell_ptr->label_num = Pbi[pbi_index].num_labels;

        for (j = 0; j < fg_ell_ptr->label_num; j++)
            fg_ell_ptr->ell_lbl_ptr->label_ind[j] =
                    Pbi[pbi_index].label_ptr[j].index;

        fg_ell_ptr->rev_video = reverse_video;

        fg_ell_ptr->ddd_ind = Pbi[pbi_index].ddd_indx;

        if (Pbi[pbi_index].ddd_indx > 0) {
            fg_ell_ptr->rev_video = NO;
            reverse_video = NO;
        } else {
            fg_ell_ptr->cur_color = new_color;
            fg_ell_ptr->rev_video = reverse_video;
        }

        break;

    case CLOCK_METER:

        fg_clk_ptr = (struct fg_clkmtr_rec *)
                            loc_graph_ptr[grph_index].graph_ptr;

        fg_clk_ptr->label_num = Pbi[pbi_index].num_labels;

        for (j = 0; j < fg_clk_ptr->label_num; j++)
            fg_clk_ptr->clk_lbl_ptr->label_ind[j] =
                    Pbi[pbi_index].label_ptr[j].index;

        fg_clk_ptr->rev_video = reverse_video;

        break;

    case BAR_CHART:

        fg_bar_ptr = (struct fg_bar_rec *)
                            loc_graph_ptr[grph_index].graph_ptr;
```

```
                    fg_bar_ptr->label_num = Pbi[pbi_index].num_labels;

                    for (j = 0; i < fg_bar_ptr->label_num; j++)
                        fg_bar_ptr->bar_lbl_ptr->label_ind[j] =
                                Pbi[pbi_index].label_ptr[j].index;

                    fg_bar_ptr->rev_video = reverse_video;

                    break;

                default:

                    break;

                }
            }
        }
    }

    if (change_flag == ON)
        redraw(disp_num);

/*
    if (change_flag == ON) {
        redraw(redraw_rect.ulx, redraw_rect.lry,
             redraw_rect.lrx, redraw_rect.uly, NO);
        redraw(redraw_rect.ulx, redraw_rect.lry,
             redraw_rect.lrx, redraw_rect.uly, YES);
    }
*/

    D(printf("END pbi_config\n"));
    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: pbi_free.c
 *
 *  This routine draws a red box around the pbi on which the
 *  cursor was located when the mouse button was pressed.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  S. Zrubek       - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <wex/FCpbi.h>
#include <constants.h>
#include <disp.h>
#include <wex/EXmsg.h>


extern struct pbi_def *Pbi_Def;      /* Pbi display definition table pointer */
extern PBI_ENTRY *Pbi_Ptr;           /* Pbi Emulation interface table ptr    */
extern PBI_TABLE *Pbi_Table;         /* Number of Pbi entries currently used */

extern short    Pbi_Hot_Ndx,         /* Number of Pbi entries currently used */
                Pbi_Num;             /* Number of Pbi entries currently used */

extern int      errno,
                Pbi_Env_Id;          /* Pbi Environment id for this display  */


int pbi_free ( )
{
    struct pbi_def *pbi_def_ptr;     /* Pbi definition pointer for pbi defs  */

    int             i,               /* integer loop control index           */
                    success;         /*  */

    D(printf("START pbi_Free\n"));
/*
 *  Call WSA software to delete the PBI environment associated with this display
 */

    Pbi_Num = 0;
    Pbi_Hot_Ndx = 0;

#if FAC == NO

    if ( Pbi_Env_Id > 0 )
        /*success = FCpbidel ( Pbi_Env_Id )*/;

    if ( success == INVALID )
        tui_msg ( M_YELLOW, "%d %s", errno, EXerrmsg ( errno ) );

#endif
```

```
/*
 *  Free PBI TABLE, PBI GROUP, and PBI ENTRY environment associated with
 *  this display
 */

    if ( Pbi_Table != NULL ) {
        if ( Pbi_Table->group_entry != NULL )
            free ( Pbi_Table->group_entry );
        free ( Pbi_Table );
    }
    if ( Pbi_Ptr != NULL )
        free ( Pbi_Ptr );

/*
 *  Free all messages, destinations, and dependent MSIDs associated with the
 *  Pbi Definition Table entries before free the Manager display definition.
 */

    if ( Pbi_Def != NULL ) {
        pbi_def_ptr = Pbi_Def;

        for ( i = 0; i < Pbi_Num; i++ ) {
            if ( pbi_def_ptr->pbi_mesg_len > 0 )
                free ( pbi_def_ptr->pbi_message );

            if ( pbi_def_ptr->pbi_dest_len > 0 )
                free ( pbi_def_ptr->pbi_dest );

            if ( pbi_def_ptr->pbi_dep_msid_cnt > 0 )
                free ( pbi_def_ptr->pbi_dep_msids );

            pbi_def_ptr++;
        }

        free ( Pbi_Def );

    }
    D(printf("END pbi_Free\n"));
}
```

```
/*****************************************************************************
 * MODULE NAME: pbi_host.c
 *
 *  This function executes a host PBI.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  A. Sprinkle      - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <constants.h>
#include <disp.h>
#include <wex/FCpbi.h>
#include <wex/EXmsg.h>


extern struct pbi_def   *Pbi_Def;          /* Pbi structure for Pbi processing    */
extern PBI_ENTRY        *Pbi_Ptr;          /* pbi entry WSA table pointer         */
extern PBI_TABLE        *Pbi_Table;        /* Pbi header table pointer for groups */

extern short            Pbi_Hot_Ndx;       /* Pbi Hot index for Pbi processing    */

extern int              Pbi_Env_Id,        /* Pbi environment Id for this display */
                        Pbi_Toggle_Dir,    /* Forward/Reverse toggle direction    */
                        errno;


int pbi_host ( )
{
    PBI_ENTRY      *entry_ptr,             /* pointer to pbi entry found          */
                   *pbi;                   /* pointer to PBI Table selected       */

    GROUP_ENTRY    *pbi_grp;               /* pointer to PBI Group selected       */

    struct pbi_def *pbi_def_ptr;           /* pointer to PBI definitions          */

    int             field_select_reset = FALSE,
                    group_cnt,             /* number of groups in the pbi table   */
                    i,
                    loop_counter = 0,      /* counter to increment pbi_def_ptr    */
                    match_found = FALSE,   /* flag for searching for PBI groups   */
                    modified = 0,          /* Number of Pbi's Modified            */
                    start_index;           /* offset for pbi_updt                 */

    pbi = Pbi_Ptr;
    pbi_grp = Pbi_Table->group_entry;
    group_cnt = Pbi_Table->group_count;
    pbi_def_ptr = Pbi_Def;

    D(printf("START pbi_host\n"));
/*
 *  Switch on the type of the pbi selected
 */
```

```
    switch ( pbi[Pbi_Hot_Ndx - 1].pbi_type ) {

/*
 *  If the selected PBI is a dependent disable
 */

    case DS:

/*
 *      If there is no backlighting for this PBI
 */

        if ( pbi_def_ptr->pbi_bklght != NO_BCKLGHT ) {
            pbi[Pbi_Hot_Ndx - 1].feedback_ind = ON;
            pbi[Pbi_Hot_Ndx - 1].modify_flag = ON;
            modified++;
        }

/*
 *      Find the group of displays PBI's that matches the group of the selected PBI
 */

        for ( i = 0; i < group_cnt && !match_found; i++ ) {
            if ( pbi[Pbi_Hot_Ndx - 1].group_num == pbi_grp->group_num ) {
                match_found = TRUE;
                entry_ptr = pbi_grp->pbi_ptr;
            } else {
                loop_counter = loop_counter + pbi_grp->entry_count;
                pbi_grp++;
            }
        }

/*
 *      Increment pbi_def_ptr to match the pbi in entry_ptr
 */

        for ( i = 0; i < loop_counter; i++ ) {
            pbi_def_ptr++;
        }

/*
 *      Process all PBI's in this group
 */

        for ( i = 0; i < pbi_grp->entry_count; i++ ) {

/*
 *          If the entry is active and it is a dependent disable
 */

            if ( entry_ptr->active_flag == TRUE && entry_ptr->pbi_type == DP ) {
                pbi_def_ptr->pbi_disable = DISABLED;
            }

/*
 *          If the entry is active and if the selected PBI has backlight enabled
 */

            if ( entry_ptr->active_flag == TRUE &&
                 pbi_def_ptr->pbi_bklght != NO_BCKLGHT ) {

/*
```

```
 *              If the entry is a dependent enable
 */

                if ( entry_ptr->pbi_type == EN ) {
                    entry_ptr->feedback_ind = OFF;
                    entry_ptr->modify_flag = ON;
                    modified++;
                }

/*
 *              If the entry is a dependent disable
 */

                if ( entry_ptr->pbi_type == DS ) {
                    entry_ptr->feedback_ind = ON;
                    entry_ptr->modify_flag = ON;
                    modified++;

                }
            }

/*
 *          If the entry is active and it is a dependent execute or it is a dependent clea
r
 */

            if ( entry_ptr->active_flag == TRUE &&
                 ( entry_ptr->pbi_type == DE ||
                  entry_ptr->pbi_type == DC ) ) {

                pbi_def_ptr->pbi_disable = DISABLED;
            }
            pbi_def_ptr++;
            entry_ptr++;

        }

/*
 *      If a pbi entry has been modified during all of this
 */

        if ( modified > 0 ) {
            start_index = ( pbi_grp->pbi_ptr ) - Pbi_Ptr;
            pbi_updt ( modified, start_index, pbi_grp->entry_count );
        }
        break;

/*
 *  If the selected PBI is a dependent enable
 */

    case EN:

/*
 *      If there is no backlighting for this PBI
 */

        if ( pbi_def_ptr->pbi_bklght != NO_BCKLGHT ) {

            pbi[Pbi_Hot_Ndx - 1].feedback_ind = ON;
            pbi[Pbi_Hot_Ndx - 1].modify_flag = ON;
            modified++;
        }
```

```
/*
 *      Find the group of displays PBI's that matches the group of the selected PBI
 */

        for ( i = 0; i < group_cnt && !match_found; i++ ) {

            if ( pbi[Pbi_Hot_Ndx - 1].group_num == pbi_grp->group_num ) {
                match_found = TRUE;
                entry_ptr = pbi_grp->pbi_ptr;
            } else {
                loop_counter = loop_counter + pbi_grp->entry_count;
                pbi_grp++;
            }

        }

/*
 *      Increment pbi_def_ptr to match the pbi in entry_ptr
 */

        for ( i = 0; i < loop_counter; i++ ) {
            pbi_def_ptr++;
        }

/*
 *      Process all PBI's in this group
 */

        for ( i = 0; i < pbi_grp->entry_count; i++ ) {

/*
 *          If the entry is active and it is a dependent disable
 */

            if ( entry_ptr->active_flag && entry_ptr->pbi_type == DP ) {
                pbi_def_ptr->pbi_disable = ENABLED;
            }

/*
 *          If the entry is active and it is a dependent clear or it is a dependent
 *          execute
 */

            if ( entry_ptr->active_flag == TRUE &&
                 ( entry_ptr->pbi_type == DE || entry_ptr->pbi_type == DC ) ) {

                pbi_def_ptr->pbi_disable = ENABLED;

            }

/*
 *          If the entry is active and the selected PBI has backlighting enabled
 */

            if ( entry_ptr->active_flag &&
                 pbi_def_ptr->pbi_bklght != NO_BCKLGHT ) {

/*
 *              If the entry is a dependent enable
 */

                if ( entry_ptr->pbi_type == EN ) {
                    entry_ptr->feedback_ind = ON;
                    entry_ptr->modify_flag = ON;
```

```
                    modified++;
                }

/*
 *              If the entry is a dependent disable
 */

                if ( entry_ptr->pbi_type == DS ) {
                    entry_ptr->feedback_ind = OFF;
                    entry_ptr->modify_flag = ON;
                    modified++;
                }
            }

/*
 *          If a pbi entry has been modified during all of this
 */

            if ( modified > 0 ) {
                start_index = ( pbi_grp->pbi_ptr ) - Pbi_Ptr;
                pbi_updt ( modified, start_index, pbi_grp->entry_count );
            }
            pbi_def_ptr++;
            entry_ptr++;

        }
        break;

/*
 *  If the pbi selected is any other type
 */

    default:

        if ( pbi[Pbi_Hot_Ndx - 1].pbi_type == FS ) {
            field_select_reset = TRUE;
        } else if ( pbi[Pbi_Hot_Ndx - 1].pbi_type == FP ) {
            Pbi_Toggle_Dir = TOGGLE_FORWARD;
        } else if ( pbi[Pbi_Hot_Ndx - 1].pbi_type == MULTIDEF + FP ) {
            Pbi_Toggle_Dir = TOGGLE_REVERSE;
        } else if ( pbi[Pbi_Hot_Ndx - 1].pbi_type == RP ) {
            Pbi_Toggle_Dir = TOGGLE_REVERSE;
        } else if ( pbi[Pbi_Hot_Ndx - 1].pbi_type == MULTIDEF + RP ) {
            Pbi_Toggle_Dir = TOGGLE_FORWARD;
        }
        errno = 0;

#if FAC == NO
        modified = FCpbisel ( Pbi_Env_Id, & ( pbi[Pbi_Hot_Ndx - 1] ) );
        if ( errno > 0 )
            tui_msg ( M_YELLOW, "%d %s", errno, EXerrmsg ( errno ) );
#endif
        break;
    }

/*
 *  If we reset the field select
 */

    if ( field_select_reset ) {
        entry_ptr = pbi_grp->pbi_ptr;

/*
 *      Process all entries in this group
```

```
 */

        for ( i = 0; i < pbi_grp->entry_count; i++ ) {
            if ( entry_ptr->active_flag && pbi_def_ptr->pbi_disable == DISABLED ) {
                pbi_def_ptr->pbi_disable = ENABLED;
            }
            if ( entry_ptr->active_flag && pbi_def_ptr->pbi_bklght == INIT_BCKLGHT ) {
                entry_ptr->feedback_ind = ON;
                entry_ptr->modify_flag = ON;
            }
            entry_ptr++;
            pbi_def_ptr++;

        }

/*
 *      If a pbi entry has been modified during all of this
 */

        if ( modified > 0 ) {
            start_index = ( pbi_grp->pbi_ptr ) - Pbi_Ptr;
            pbi_updt ( modified, start_index, pbi_grp->entry_count );

        }
    }
    D(printf("END pbi_host\n"));
}
```

```
/*****************************************************************************
 * MODULE NAME: pbi_hot.c
 *
 *  This function determines if a PBI was selected by comparing mouse coor-
 *  dinates with the bounding boxes of each PBI.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  A. Sprinkle     - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <X11/Xlib.h>
#include <constants.h>
#include <disp.h>
#include <wex/FCpbi.h>
#include <wex/EXmsg.h>


extern struct dm_shmemory   *Dm_Address;
extern struct pbi_def       *Pbi_Def;
extern PBI_ENTRY            *Pbi_Ptr;

extern short                Disp_Num,
                            Pbi_Num;

extern int                  Screen_color;


int pbi_hot ( x, y )

    int                     x,              /* X Coordinate of mouse button selection.  */

                            y;              /* Y Coordinate of mouse button selection.  */

{
    register int            i;              /* loop control variable for indexing       */

    struct pbi_def          *pbi_def_ptr;   /* Pbi Pointer to pbi display definition    */
    PBI_ENTRY               *pbi;           /* Pbi Pointer to pbi display definition    */

    Display                 *display;       /* X Windows display variable.              */


    D(printf("START pbi_hot\n"));
/*
 *  Return 0 if there are no PBI's.
 */

    if ( Pbi_Num == 0 )
        return ( 0 );

/*
 *  Search the PBI table to determine if the x/y coordinates are in a PBI. If so, flash
```

```
 *  a rectangle around the PBI and return current PBI index.
 */

    display     = Dm_Address->xdisplay[Disp_Num];

    pbi_def_ptr = Pbi_Def;
    pbi         = Pbi_Ptr;

    for ( i = 0; ( i < Pbi_Num ); i++ ) {
        if ( pbi[i].active_flag == PBI_ACTIVE &&
             ( x >= pbi_def_ptr[i].pbi_ul_x_p    ) &&
             ( x <= pbi_def_ptr[i].pbi_lr_x_p    ) &&
             ( y >= pbi_def_ptr[i].pbi_ul_y_p    ) &&
             ( y <= pbi_def_ptr[i].pbi_lr_y_p    )    ) {

            XSetForeground ( display, Dm_Address->gc[Disp_Num], 31  );
            XDrawRectangle ( display, Dm_Address->window[Disp_Num],
                             Dm_Address->gc[Disp_Num],
                             pbi_def_ptr[i].pbi_ul_x_p, pbi_def_ptr[i].pbi_ul_y_p,
                             pbi_def_ptr[i].pbi_lr_x_p - pbi_def_ptr[i].pbi_ul_x_p,
                             pbi_def_ptr[i].pbi_lr_y_p - pbi_def_ptr[i].pbi_ul_y_p );
            XSync ( display );
            sleep ( 1 );

            XSetForeground ( display, Dm_Address->gc[Disp_Num], Screen_color  );
            XDrawRectangle ( display, Dm_Address->window[Disp_Num],
                             Dm_Address->gc[Disp_Num],
                             pbi_def_ptr[i].pbi_ul_x_p, pbi_def_ptr[i].pbi_ul_y_p,
                             pbi_def_ptr[i].pbi_lr_x_p - pbi_def_ptr[i].pbi_ul_x_p,
                             pbi_def_ptr[i].pbi_lr_y_p - pbi_def_ptr[i].pbi_ul_y_p );

            return ( i + 1 );
        }

    }

    D(printf("END pbi_hot\n"));
    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: pbi_local.c
 *
 *  This routine processes the Local Dependent enable, disable, execute and
 *  clear, and any local standard, dependent pbi, dependent group pbi, or no
 *  action pbis that have a local destination. The command token number and
 *  number modified are returned. The pbis are processed according to pbi type.
 *  A display manager command token function number is set according to
 *  the input instruction, and the display manager then executes the function
 *  selection.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  A. Sprinkle      - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <constants.h>
#include <pf_key.h>
#include <disp.h>
#include <wex/FCpbi.h>
#include <wex/EXmsg.h>


extern PBI_ENTRY            *Pbi_Ptr;       /* pbi entry WSA table pointer         */
extern PBI_TABLE            *Pbi_Table;     /* Pbi header table pointer for groups */
extern struct pfkey_defs    Current_Com;
extern struct pbi_def       *Pbi_Def;       /* Pbi structure for Pbi processing    */

extern short                Pbi_Hot_Ndx,    /* Pbi Hot index for Pbi processing    */
                            Pbi_Disable;

extern int                  Pbi_Toggle_Dir; /* Forawrd/Reverse toggle direction    */


int pbi_local ( )
{
    struct pbi_def *pbi_def_ptr;            /* pointer to PBI definitions          */
    struct pfkey_defs *cmd_ptr;             /* pointer to command structure        */

    PBI_ENTRY       *entry_ptr,             /* pointer to pbi entry found          */
                    *entry_to_check,        /* pointer to pbi entry found          */
                    *first_entry,           /* pointer to pbi entry found          */
                    *last_entry,            /* pointer to pbi entry found          */
                    *pbi;                   /* pointer to PBI Table selected       */

    GROUP_ENTRY     *pbi_grp;               /* pointer to PBI Group selected       */

    int             entry_count = 0,        /* number of entries in the group      */
                    group_cnt,              /* number of groups in the pbi table   */
                    i,
                    match_found = FALSE,    /* flag for searching for PBI groups   */
                    modified = 0,           /* Number of Pbi's Modified            */
                    next_found = FALSE;     /* flag for if the next entry is found */
```

```
    int             loop_counter = 0,       /* counter for number of pbis seen      */
                    pbi_array_index;        /* index into array of pbis             */

    D(printf("START pbi_local\n"));

    pbi = Pbi_Ptr;
    pbi_grp = Pbi_Table->group_entry;
    group_cnt = Pbi_Table->group_count;
    pbi_def_ptr = Pbi_Def;

    if ( pbi[Pbi_Hot_Ndx - 1].pbi_type >= MULTIDEF &&
        pbi[Pbi_Hot_Ndx - 1].active_flag ) {

/*
 *      Find the group of displays PBI's that matches the group of the selected PBI
 */

        for ( i = 0; i < group_cnt && !match_found; i++ ) {
            if ( pbi[Pbi_Hot_Ndx - 1].group_num == pbi_grp->group_num ) {
                match_found = TRUE;
                entry_ptr = pbi_grp->pbi_ptr;
                entry_count = pbi_grp->entry_count;
            } else {
                loop_counter = loop_counter + pbi_grp->entry_count;
                pbi_grp++;
            }
        }

/*
 *      Find a new entry table entry
 */

        entry_to_check = & ( pbi[Pbi_Hot_Ndx - 1] );
        first_entry = entry_ptr;
        last_entry = & ( entry_ptr[entry_count - 1] );
        pbi_array_index = Pbi_Hot_Ndx - 1;

/*
 *      Find the next button in the MULTI DEF
 */

        next_found = FALSE;
        while ( !next_found ) {

            if ( entry_to_check == first_entry && Pbi_Toggle_Dir == TOGGLE_REVERSE ) {
                entry_to_check = last_entry;
                pbi_array_index = loop_counter + entry_count - 1;
            } else if ( ( entry_to_check == last_entry &&
                    Pbi_Toggle_Dir == TOGGLE_FORWARD ) ||
                  ( entry_to_check == NULL ) ) {
                entry_to_check = first_entry;
                pbi_array_index = loop_counter;
            } else {
                entry_to_check = entry_to_check + Pbi_Toggle_Dir;
                pbi_array_index = pbi_array_index + Pbi_Toggle_Dir;
            }

            if ( entry_to_check->lock_num == pbi[Pbi_Hot_Ndx - 1].lock_num ) {

                if ( ( entry_to_check->pbi_type == pbi[Pbi_Hot_Ndx - 1].pbi_type ) ||
                    ( entry_to_check->pbi_type == MULTIDEF + FP &&
                     pbi[Pbi_Hot_Ndx - 1].pbi_type == MULTIDEF + RP ) ||
                    ( entry_to_check->pbi_type == MULTIDEF + RP &&
```

```
                    pbi[Pbi_Hot_Ndx - 1].pbi_type == MULTIDEF + FP ) ) {

                    pbi[Pbi_Hot_Ndx - 1].active_flag = OFF;
                    pbi[Pbi_Hot_Ndx - 1].modify_flag = ON;
                    modified++;

                    entry_ptr = entry_to_check;
                    entry_ptr->feedback_ind = OFF;
                    entry_ptr->active_flag = ON;
                    entry_ptr->modify_flag = ON;
                    modified++;
                    next_found = TRUE;

                }
            }
        }

/*
 *      Turn off feedback on all non-MULTIDEF PBIs that have the same lock number
 *      as the pbi selected ( if it is greater than 0 )
 */

        if ( ( pbi[Pbi_Hot_Ndx - 1].pbi_type < MULTIDEF ) &&
             ( pbi[Pbi_Hot_Ndx - 1].lock_num > 0 ) ) {

            entry_ptr = first_entry;

            for ( i = 0; i < entry_count; i++ ) {

                if ( ( entry_ptr->lock_num == pbi[Pbi_Hot_Ndx - 1].lock_num ) &&
                     ( entry_ptr != & ( pbi[Pbi_Hot_Ndx - 1] ) ) ) {

                    entry_ptr->modify_flag = ON;
                    entry_ptr->feedback_ind = OFF;
                    modified++;

                }
                entry_ptr++;

            }

        }
    } else {
        entry_ptr = & ( pbi[Pbi_Hot_Ndx - 1] );
        pbi_array_index = Pbi_Hot_Ndx - 1;
    }

/*
 *  Return invalid for invalid PBIs
 */

    if ( pbi_def_ptr[pbi_array_index].pbi_cmd_ptr != NULL ) {

        cmd_ptr = pbi_def_ptr[pbi_array_index].pbi_cmd_ptr;
        if ( pbi_def_ptr[pbi_array_index].pbi_cmd_ptr->func_no == INVALID ||
             pbi_def_ptr[pbi_array_index].pbi_cmd_ptr->defined == NO ||
             pbi_def_ptr[pbi_array_index].pbi_cmd_ptr->valid_flag == INVALID ) {
             return ( -1 );
        }
    }

/*
 *  Switch on the type of the selected PBI
 */
```

```
    match_found = FALSE;
    switch ( pbi[pbi_array_index].pbi_type ) {

/*
 *  Process standard PBIs
 */

    case ( LOCAL_PBI + SP ) :
    case ( MULTIDEF + LOCAL_PBI + SP ) :

        cmd_ptr = pbi_def_ptr[pbi_array_index].pbi_cmd_ptr;
        if ( pbi_def_ptr[pbi_array_index].pbi_bklght != NO_BCKLGHT ) {

            if ( entry_ptr->feedback_ind == OFF ) {

                entry_ptr->feedback_ind = ON;
                entry_ptr->modify_flag = ON;
                modified++;
            }
        }
        if ( modified > 0 )
            pbi_updt ( modified, pbi_grp->pbi_ptr, pbi_grp->entry_count );

        if ( ( entry_ptr->active_flag ) &&
            ( !pbi_def_ptr[pbi_array_index].pbi_disable ) &&
            ( cmd_ptr->defined && cmd_ptr->valid_flag == VALID ) ) {
            Current_Com = *cmd_ptr;
            Pbi_Disable = DISABLED;
            command ( FALSE );
            Pbi_Disable = ENABLED;
            Current_Com.func_no = INVALID;
        }
        break;

/*
 *  Process dependent PBIs
 */

    case ( LOCAL_PBI + DP ) :

        cmd_ptr = pbi_def_ptr[pbi_array_index].pbi_cmd_ptr;

        if ( pbi_def_ptr[Pbi_Hot_Ndx - 1].pbi_disable == ENABLED ) {

            if ( ( entry_ptr->feedback_ind == OFF ) &&
                ( cmd_ptr->defined && cmd_ptr->valid_flag == VALID ) )
                entry_ptr->feedback_ind = ON;
            else
                entry_ptr->feedback_ind = OFF;

            entry_ptr->modify_flag = ON;
            modified++;
            pbi_updt ( modified, entry_ptr, pbi_grp->entry_count );

        }
        break;

/*
 *  Process dependent group PBIs
 */

    case ( LOCAL_PBI + DG ) :
    case ( MULTIDEF + LOCAL_PBI + DG ) :
```

```
        if ( pbi_def_ptr[pbi_array_index].pbi_bklght != NO_BCKLGHT ) {

            if ( entry_ptr->feedback_ind == OFF ) {

                entry_ptr->feedback_ind = ON;
                entry_ptr->modify_flag = ON;
                modified++;

            }
        }

/*
 *      Call pbi_updt for the modified PBIs
 */

        if ( modified > 0 )
            pbi_updt ( modified, pbi_grp->pbi_ptr, pbi_grp->entry_count );

/*
 *      Process all of the commands for the selected PBI
 */

        cmd_ptr = pbi_def_ptr[pbi_array_index].pbi_cmd_ptr;

        for ( i = 0; i < pbi_def_ptr[pbi_array_index].pbi_cmd_cnt; i++ ) {

            Current_Com = *cmd_ptr;

            if ( cmd_ptr->func_no != CLEAR_DISPLAY &&
                 cmd_ptr->func_no != HALT_DISPLAY &&
                 cmd_ptr->func_no != START_DISPLAY &&
                 cmd_ptr->func_no != START_PDISPLAY ) {

                Pbi_Disable = DISABLED;
                command ( FALSE );
                Pbi_Disable = ENABLED;
                Current_Com.func_no = INVALID;
                cmd_ptr++;

            } else {

                return ( cmd_ptr->func_no );

            }

        }

        break;

/*
 *  Process local dependent clear PBIs
 */

    case ( LOCAL_PBI + DC ) :

        pbi_grp = Pbi_Table->group_entry;

/*
 *      Find the group of the selected PBI
 */

        for ( i = 0; i < group_cnt && !match_found; i++ ) {
```

```
            if ( pbi[pbi_array_index].group_num == pbi_grp->group_num ) {
                match_found = TRUE;
                entry_ptr = pbi_grp->pbi_ptr;
            } else {
                loop_counter = loop_counter + pbi_grp->entry_count;
                pbi_grp++;
            }

        }

/*
 *      Process all of the dependent PBIs
 */

        for ( i = 0; i < pbi_grp->entry_count; i++ ) {

            cmd_ptr = pbi_def_ptr[loop_counter + i].pbi_cmd_ptr;

            if ( ( entry_ptr->pbi_type == ( LOCAL_PBI + DP ) ) &&
                ( cmd_ptr->defined && cmd_ptr->valid_flag == VALID ) ) {

                if ( ( pbi_def_ptr[loop_counter + i].pbi_bklght != NO_BCKLGHT ) &&
                    ( entry_ptr->feedback_ind == ON ) ) {

                    entry_ptr->feedback_ind = OFF;
                    entry_ptr->modify_flag = ON;
                    modified++;

                }
            }
            if ( ( entry_ptr->pbi_type == ( LOCAL_PBI + DC ) ) &&
                ( pbi_def_ptr[loop_counter + i].pbi_bklght != NO_BCKLGHT ) ) {

                entry_ptr->feedback_ind = OFF;
                entry_ptr->modify_flag = ON;
                modified++;

            }
            entry_ptr++;

        }

        if ( modified > 0 )
            pbi_updt ( modified, pbi_grp->pbi_ptr, pbi_grp->entry_count );

        break;

/*
 *  Process local dependent execute PBIs
 */

    case ( LOCAL_PBI + DE ) :

/*
 *      Reset the feedback and backlighting
 */

        if ( pbi_def_ptr[pbi_array_index].pbi_bklght != NO_BCKLGHT ) {

            entry_ptr->feedback_ind = OFF;
            entry_ptr->modify_flag = ON;
            modified++;

        }
```

```
        pbi_grp = Pbi_Table->group_entry;

    /*
     *      Find the group of the selected PBI
     */

        for ( i = 0; i < group_cnt && !match_found; i++ ) {

            if ( pbi[pbi_array_index].group_num == pbi_grp->group_num ) {
                match_found = TRUE;
                entry_ptr = pbi_grp->pbi_ptr;
            } else {
                loop_counter = loop_counter + pbi_grp->entry_count;
                pbi_grp++;
            }

        }

    /*
     *      Call pbi_updt to handle the changed PBIs
     */

        if ( modified > 0 )
            pbi_updt ( modified, pbi_grp->pbi_ptr, pbi_grp->entry_count );

    /*
     *      Process the PBIs in the group of the PBI selected
     */

        for ( i = 0; i < pbi_grp->entry_count; i++ ) {

            cmd_ptr = pbi_def_ptr[loop_counter + i].pbi_cmd_ptr;


            if ( ( entry_ptr->pbi_type == ( LOCAL_PBI + DP ) ) &&
                 ( entry_ptr->active_flag ) && ( entry_ptr->feedback_ind == ON ) &&
                 ( cmd_ptr->defined && cmd_ptr->valid_flag == VALID ) ) {

                Current_Com = *cmd_ptr;

                if ( ( cmd_ptr->func_no != CLEAR_DISPLAY ) &&
                     ( cmd_ptr->func_no != HALT_DISPLAY ) &&
                     ( cmd_ptr->func_no != START_DISPLAY ) &&
                     ( cmd_ptr->func_no != START_PDISPLAY ) ) {

                    Pbi_Disable = DISABLED;
                    command ( FALSE );
                    Pbi_Disable = ENABLED;
                    Current_Com.func_no = INVALID;

                } else
                    return ( cmd_ptr->func_no );

            }
            entry_ptr++;
        }
        break;

    /*
     *  Process local dependent enable PBIs
     */

        case ( LOCAL_PBI + EN ) :
```

```
        pbi_grp = Pbi_Table->group_entry;

/*
 *      Find the group of the selected PBI
 */

        for ( i = 0; i < group_cnt && !match_found; i++ ) {

            if ( pbi[pbi_array_index].group_num == pbi_grp->group_num ) {
                match_found = TRUE;
                entry_ptr = pbi_grp->pbi_ptr;
            } else {
                loop_counter = loop_counter + pbi_grp->entry_count;
                pbi_grp++;
            }

        }

/*
 *      Process the PBIs in the group of the selected PBI
 */

        for ( i = 0; i < pbi_grp->entry_count; i++ ) {

            cmd_ptr = pbi_def_ptr[loop_counter + i].pbi_cmd_ptr;

            if ( entry_ptr->active_flag ) {

                if ( ( entry_ptr->pbi_type == ( LOCAL_PBI + DP ) ) &&
                     ( cmd_ptr->defined && cmd_ptr->valid_flag == VALID ) ) {
                    pbi_def_ptr[loop_counter + i].pbi_disable = ENABLED;
                }
                if ( entry_ptr->pbi_type == ( LOCAL_PBI + EN ) &&
                    pbi_def_ptr[loop_counter + i].pbi_bklght != NO_BCKLGHT ) {

                    entry_ptr->feedback_ind = ON;
                    entry_ptr->modify_flag = ON;
                    modified++;

                }
                if ( entry_ptr->pbi_type == ( LOCAL_PBI + DS ) &&
                    pbi_def_ptr[loop_counter + i].pbi_bklght != NO_BCKLGHT ) {

                    entry_ptr->feedback_ind = OFF;
                    entry_ptr->modify_flag = ON;
                    modified++;

                }
                if ( entry_ptr->pbi_type == ( LOCAL_PBI + DE ) ||
                    entry_ptr->pbi_type == ( LOCAL_PBI + DC ) ) {
                    pbi_def_ptr[loop_counter + i].pbi_disable = ENABLED;
                }
            }
            entry_ptr++;
        }

/*
 *      Call pbi_updt for the modified PBIs
 */

        if ( modified > 0 )
            pbi_updt ( modified, pbi_grp->pbi_ptr, pbi_grp->entry_count );

        break;
```

```
/*
 *  Handle local dependent disable
 */

    case ( LOCAL_PBI + DS ) :

        pbi_grp = Pbi_Table->group_entry;

/*
 *      Find the group of the selected PBI
 */

        for ( i = 0; i < group_cnt && !match_found; i++ ) {

            if ( pbi[pbi_array_index].group_num == pbi_grp->group_num ) {
                match_found = TRUE;
                entry_ptr = pbi_grp->pbi_ptr;
            } else {
                loop_counter = loop_counter + pbi_grp->entry_count;
                pbi_grp++;
            }
        }

/*
 *      Process all of the PBIs in the group of the selected PBI
 */

        for ( i = 0; i < pbi_grp->entry_count; i++ ) {

            cmd_ptr = pbi_def_ptr[loop_counter + i].pbi_cmd_ptr;

            if ( entry_ptr->active_flag ) {

                if ( ( entry_ptr->pbi_type == ( LOCAL_PBI + DP ) ) &&
                    ( cmd_ptr->defined && cmd_ptr->valid_flag == VALID ) ) {

                    pbi_def_ptr[loop_counter + i].pbi_disable = DISABLED;

                }
                if ( entry_ptr->pbi_type == ( LOCAL_PBI + EN ) &&
                    pbi_def_ptr[loop_counter + i].pbi_bklght != NO_BCKLGHT ) {

                    entry_ptr->feedback_ind = OFF;
                    entry_ptr->modify_flag = ON;
                    modified++;

                }
                if ( entry_ptr->pbi_type == ( LOCAL_PBI + DS ) &&
                    pbi_def_ptr[loop_counter + i].pbi_bklght != NO_BCKLGHT ) {

                    entry_ptr->feedback_ind = ON;
                    entry_ptr->modify_flag = ON;
                    modified++;

                }
                if ( entry_ptr->pbi_type == ( LOCAL_PBI + DE ) ||
                    entry_ptr->pbi_type == ( LOCAL_PBI + DC ) ) {

                    pbi_def_ptr[loop_counter + i].pbi_disable = DISABLED;

                }
            }
            entry_ptr++;
```

```
        }

/*
 *      Call pbi_updt for the modified PBIs
 */

        if ( modified > 0 )
            pbi_updt ( modified, pbi_grp->pbi_ptr, pbi_grp->entry_count );

        break;
    }
    D(printf("END pbi_local\n"));
    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: pbi_setup.c
 *
 *  This routine initializes the PBI table entries necessary for both
 *  interfacing with the WSA PBI emulation process and for processing local
 *  PBIs.  A PBI header is set up for each pbi group to process the PBIs by
 *  group.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  A. Sprinkle      - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <constants.h>
#include <pf_key.h>
#include <disp.h>
#include <wex/FCpbi.h>
#include <wex/EXmsg.h>


extern PBI_ENTRY        *Pbi_Ptr;           /* Pbi pointer to Pbi entry table          */
extern GROUP_ENTRY      *Pbi_Group;         /* Pbi pointer to Pbi header table         */
extern PBI_TABLE        *Pbi_Table;         /* Pbi pointer to Pbi header table         */
extern struct pbi_def   *Pbi_Def;           /* Pbi pointer to Pbi header table         */

extern int              Pbi_Env_Id,         /* Pbi environment id for this display     */
                        errno;
extern short            Pbi_Num;            /* Display number of the Display Manager   */


int pbi_setup ( flt, datatype )

    char            *flt;                   /* input flight id used for processing     */

    char            *datatype;              /* input data type used for processing     */

{
    GROUP_ENTRY         *pbi_grp;           /* Pbi pointer to Pbi header table         */
    PBI_ENTRY           *pbi;               /* internal pointer for type PBI_TABLE     */
    struct pbi_def      *pbi_def_ptr;       /* internal pointer for type pbi_def ( DDF )*/
    struct pfkey_defs   *cmd_ptr;           /* pointer to a pbi command structure      */


    char                *full_buffer,       /* temp storage for the complete command   */
                        *message;           /* storage for misc. char variables        */

    short               pbi_prev_grp;       /* previous group found while searching    */

    int                 already_invalid,    /* flag for invalid local group commands   */
                        i,                  /* loop index control variable             */
                        j,                  /* loop index control variable             */
                        index = 0,          /* for calc. digits in a number            */
                        length_of_cmd = 0,  /* length of a given PBI command           */
```

```
                        number_of_cmds = 0, /* number of commands for a given PBI        */
                        modification = 0;   /* number of PBIs modified                   */

    D(printf("START pbi_setup\n"));
/*
 *  Set up Pbi Tables for interfacing with PBI emulation software
 *  Set up the Group Pbi table pointed to by the header
 */

    Pbi_Table->group_entry = ( GROUP_ENTRY * ) calloc ( Pbi_Table->group_count,
                            sizeof ( GROUP_ENTRY ) );

    if ( Pbi_Table->group_entry == NULL ) {
        free ( Pbi_Table );
        free ( Pbi_Ptr );
        free ( Pbi_Def );
        tui_msg ( M_YELLOW, "Error in group allocation of PBI Group Table" );
        return ( -1 );
    }

/*
 *  Set up the Group Pbi table pointers to the corresponding entry
 *  table entries
 */

    pbi = Pbi_Ptr;

    pbi_prev_grp = pbi->group_num;
    pbi_grp = Pbi_Table->group_entry;
    pbi_grp->pbi_ptr = pbi;

    for ( i = 0; i < Pbi_Num; i++ ) {
        if ( pbi->group_num == pbi_prev_grp )
             ( pbi_grp->entry_count ) ++;
        else {
            pbi_grp++;
            pbi_grp->pbi_ptr = pbi;
            pbi_grp->entry_count++;
            pbi_grp->group_num = pbi->group_num;
            pbi_prev_grp = pbi->group_num;
        }
        pbi++;
    }

/*
 *  Parse the message string for the portion that is the command
 */

    modification = 0;

    pbi = Pbi_Ptr;
    pbi_def_ptr = Pbi_Def;

/*
 *  Loop for all of the PBI's
 */

    for ( i = 0; i < Pbi_Num; i++ ) {

/*
 *      If the PBI is initial backlit and it is active.
 */

        if ( pbi_def_ptr->pbi_bklght == INIT_BCKLGHT &&
```

```
            pbi->active_flag ) {

            pbi->feedback_ind = ON;
            pbi->modify_flag = ON;
            modification++;

        } else if ( pbi->active_flag ) {
            pbi->modify_flag = ON;
            modification++;
        }
/*
 *      If the PBI destination is local and it is a dependent PBI or a standard PBI.
 */

        if ( pbi->pbi_type == LOCAL_PBI + DP || pbi->pbi_type == LOCAL_PBI + SP ||
            pbi->pbi_type == MULTIDEF + LOCAL_PBI + DP ||
            pbi->pbi_type == MULTIDEF + LOCAL_PBI + SP ) {

/*
 *          Allocate the command structure
 */

            pbi_def_ptr->pbi_cmd_ptr =
                (struct pfkey_defs *)malloc ( sizeof ( struct pfkey_defs ) );
            cmd_ptr = pbi_def_ptr->pbi_cmd_ptr;

            parse_cmd ( cmd_ptr, pbi_def_ptr->pbi_message, pbi_def_ptr->pbi_mesg_len,
                        PBI, 0 );
        }
/*
 *      If the PBI is a local dependent group
 */

        if ( pbi->pbi_type == LOCAL_PBI + DG || pbi->pbi_type == MULTIDEF+LOCAL_PBI+DG ) {

/*
 *          Strip off the number of commands
 */

            message = ( char * ) calloc ( 1, 120 );
            strncpy ( message, pbi_def_ptr->pbi_message, strlen(pbi_def_ptr->pbi_message)
);
            sscanf ( message, "%d", &number_of_cmds );
            pbi_def_ptr->pbi_cmd_cnt = number_of_cmds;
            index = 2;
            if ( number_of_cmds > 9 )
                index++;

            strncpy ( message, &( pbi_def_ptr->pbi_message[index] ),
                      pbi_def_ptr->pbi_mesg_len - index );

/*
 *          Allocate the number of command structures necessary
 */

            pbi_def_ptr->pbi_cmd_ptr =
                (struct pfkey_defs *)calloc( number_of_cmds, sizeof ( struct pfkey_defs )
);

            cmd_ptr = pbi_def_ptr->pbi_cmd_ptr;

/*
 *          Loop for the number of commands
 */
```

```
            already_invalid = FALSE;
            for ( j = 0; j < number_of_cmds; j++ ) {

/*
 *              Strip off the length of this command
 */

                sscanf ( message, "%d", &length_of_cmd );
                index = index + 2;
                if ( length_of_cmd > 9 )
                    index++;

/*
 *              Strip off the actual command
 */

                strncpy ( message, & ( pbi_def_ptr->pbi_message[index] ),
                    pbi_def_ptr->pbi_mesg_len - index );

                full_buffer = ( char * ) calloc ( 1, length_of_cmd );
                strncpy ( full_buffer, message, length_of_cmd );
                index = index + length_of_cmd + 1;
                strncpy ( message, & ( pbi_def_ptr->pbi_message[index] ),
                    pbi_def_ptr->pbi_mesg_len - index );
                if ( pbi_def_ptr->pbi_mesg_len > index )
                    message[pbi_def_ptr->pbi_mesg_len - index] = '\0';

                parse_cmd ( cmd_ptr, full_buffer, length_of_cmd, PBI, 0 );

/*
 *              If any command in the group is invalid set a flag
 */

                if ( cmd_ptr->func_no == INVALID ||
                    cmd_ptr->defined == NO ||
                    cmd_ptr->valid_flag == INVALID )
                    already_invalid = TRUE;

                free ( full_buffer );

                cmd_ptr++;
            }
            free ( message );

/*
 *          If any command in the group is invalid set them all to invalid
 */

            cmd_ptr = pbi_def_ptr->pbi_cmd_ptr;
            if ( already_invalid ) {

                for ( j = 0; j < number_of_cmds; j++ ) {

                    cmd_ptr->func_no = INVALID;
                    cmd_ptr->defined = NO;
                    cmd_ptr->valid_flag = INVALID;

                }
                cmd_ptr++;

            }
        }
        pbi++;
```

```
        pbi_def_ptr++;
    }

/*
 *  Call WSA software to initialize the Pbi environment for interfacing with
 *  the host, and communicating pbi emulation responses ( ie button modifications
 *  in the PBI entry table
 */

    errno = 0;

#if FAC == YES
    Pbi_Env_Id = 1;
#else
    Pbi_Env_Id = FCpbinit ( flt, datatype, PBI_HST_RSP_OVRD, Pbi_Table );
#endif

    if ( errno > 0 )
        tui_msg ( M_YELLOW, "%d %s", errno, EXerrmsg ( errno ) );
    else
        pbi_updt ( modification, Pbi_Ptr, Pbi_Num );

    D(printf("END pbi_setup\n"));
    return ( Pbi_Env_Id );
}
```

```
/*****************************************************************************
 * MODULE NAME: pbi_updt.c
 *
 *  This routine passes any button modifications to the Displayer
 *  to be processed by the Displayer process.  The information is passed through
 *  shared memory and a signal is sent to the Displayer to identify a possible
 *  visible button changes.  The process delays until the Displayer has set back
 *  the pbi shared modification area for other pbi use.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  A. Sprinkle      - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <constants.h>
#include <disp.h>
#include <wex/FCpbi.h>
#include <wex/EXmsg.h>


extern struct dm_shmemory   *Dm_Address;
extern struct pbi_def       *Pbi_Def;
extern PBI_ENTRY *Pbi_Ptr;                  /* Beginning of the Pbi entry Table        */

extern short        Pbi_Num,                /* Beginning of the Pbi entry Table        */
                    Disp_Num;               /* Display Manager number                  */


int pbi_updt ( modified, pbi_start, pbi_entries )

    int             modified;               /* Number of modified entries to search for */

    PBI_ENTRY      *pbi_start;              /* Offset to start searching the Pbi table  */

    int             pbi_entries;            /* Total number of entries to search       */

{
    PBI_ENTRY           *pbi,               /* PBI Entry pointer to WSA entry table    */
                        *pbi_mark;

    struct pbi_def      *pbi_def_ptr;       /* PBI Entry pointer to Display Def Tbl    */

    int                 change_count = 0,   /* count of number of pbis changed         */
                        entry_found = 0,    /* flag for finding a changed pbi          */
                        i,                  /* index for loop control variable         */
                        start_loop = 0;     /* loop index                              */

    D(printf("START pbi_updt\n"));
/*
 *  Search for modifications to update the DISPLAYER with.
 */

    pbi = Pbi_Ptr;
```

```
    pbi_mark = Pbi_Ptr;
    pbi_def_ptr = Pbi_Def;

    if ( modified > 0 ) {

/*
 *      Find the first changed PBI
 */

        for ( i = 0; i < Pbi_Num; i++ ) {
            if ( pbi_mark == pbi_start )
                start_loop = i;

            pbi_mark++;

        }

/*
 *      Initialize the redraw rectangle to minimum values. Note that these values
 *      assume that the Y value is corrected.
 */

        Dm_Address->pbi_redraw.ulx = 100.0;
        Dm_Address->pbi_redraw.uly = 100.0;
        Dm_Address->pbi_redraw.lrx = 0.0;
        Dm_Address->pbi_redraw.lry = 0.0;

/*
 *      Process all changed PBIs
 */

        for ( i = start_loop; ( i < (pbi_entries+start_loop) ) && ( modified > 0 ); i++ )
{

/*
 *          Copy all necessary information to shared memory
 */

            if ( pbi[i].modify_flag > 0 ) {

                entry_found = TRUE;
                Dm_Address->pbi_shmemory.pbi_change[change_count].pbi_chg_ndx = i;
                Dm_Address->pbi_shmemory.pbi_change[change_count].pbi_active_flag =
                        pbi[i].active_flag;
                Dm_Address->pbi_shmemory.pbi_change[change_count].pbi_feedback_flag =
                        pbi[i].feedback_ind;

                if ( pbi_def_ptr[i].pbi_ul_x < Dm_Address->pbi_redraw.ulx )
                    Dm_Address->pbi_redraw.ulx = pbi_def_ptr[i].pbi_ul_x;

                if ( pbi_def_ptr[i].pbi_ul_y < Dm_Address->pbi_redraw.uly )
                    Dm_Address->pbi_redraw.uly = pbi_def_ptr[i].pbi_ul_y;

                if ( pbi_def_ptr[i].pbi_lr_x > Dm_Address->pbi_redraw.lrx )
                    Dm_Address->pbi_redraw.lrx = pbi_def_ptr[i].pbi_lr_x;

                if ( pbi_def_ptr[i].pbi_lr_y > Dm_Address->pbi_redraw.lry )
                    Dm_Address->pbi_redraw.lry = pbi_def_ptr[i].pbi_lr_y;

                pbi[i].modify_flag = 0;
                change_count++;
                modified--;

            }
```

```
        }

/*
 *      Set shared memory flags and call display function to update PBI.
 */

        if ( entry_found ) {
            Dm_Address->pbi_shmemory.number_of_changes = change_count;
            Dm_Address->pbi_shmemory.disp_num          = Disp_Num;

            DDpbi_updt ( Disp_Num );
        }
    }
    D(printf("END pbi_updt\n"));
    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: pk_chk.c
 *
 *  This routine draws the menu for the user to verify the selection of the PF
 *  key. A menu is drawn to show the definition of the PF key and to prompt
 *  the user for a 'y' or 'n' response.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  K. Noonan       - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <X11/Intrinsic.h>
#include <constants.h>
#include <disp.h>
#include <pf_key.h>
#include <wex/EXmsg.h>


extern Widget               Top;            /* The top level widget          */

extern struct dm_shmemory   *Dm_Address;    /* Display manager shm           */

extern short                Disp_Num;       /* display manager task number   */

extern char                 *Func_Desc[];   /* Array of labels for the functions */


int pf_chk ( New_Com )

    struct pfkey_defs *New_Com;             /* PF key command information    */
{
    char                        str [80],
                                str1[80];

    D(printf("START pf_chk\n"));
/*
 *  Process each of the commands which require display of a corresponding piece
 *  of information (filename, value, etc.).
 */

    switch ( New_Com->func_no ) {

    case START_PDISPLAY:
        get_fn ( New_Com->disp_name, str1 );
        sprintf ( str, "Start Display (%s)? ", str1 );
        break;

    case CHG_LIM:
        sprintf ( str, "Change Limits For MSID (%s)?", New_Com->limit_change.msid );
        break;

    case UPD_RATE:
        sprintf ( str, "Update display Rate Ro (%d) Seconds?", New_Com->rate/1000 );
```

```
        break;

    case LIM_GRP:
        get_fn ( New_Com->disp_name, str1 );
        if ( New_Com->action == ON )
            sprintf ( str, "Turn On Limit Group (%s)?",  str1 );
        else
            sprintf ( str, "Turn Off Limit Group (%s)?", str1 );
        break;

    case PLOT:
        get_fn ( New_Com->disp_name, str1 );
        if ( New_Com->action == ON )
            sprintf ( str, "Start Plot (%s)?", str1 );
        else
            sprintf ( str, "Stop Plot (%s)?",  str1 );
        break;

    case PLOT_OVRLAY:
        get_fn ( New_Com->disp_name, str1 );
        sprintf ( str, "Overlay Plot (%s)?", str1 );
        break;

    case HIST_TAB:
        get_fn ( New_Com->disp_name, str1 );
        sprintf ( str, "Update History Field (%s)?", str1 );
        break;

    case ZOOM_FAC:
        sprintf ( str, "Set Zoom Factor To (%3.1f)?", New_Com->factor );
        break;

    case GDR_GETNEXT:
        get_fn ( New_Com->disp_name, str1 );
        sprintf ( str, "Retrieve GDR Change For (%s)?", str1 );
        break;

    case DDD_UNLATCH:
        sprintf ( str, "Unlatch DDD MSID (%s)?", New_Com->limit_change.msid );
        break;

/*
 *  The default is to simply print the description of the command without any
 *  data values.
 */

    default:
        sprintf ( str, "%s?", Func_Desc[New_Com->func_no] );
        break;
    }

/*
 *  Call function to display and manage a popup. The result of this function is
 *  returned to the calling function. MDC - add a new help entry for this type
 *  of popup.
 */

    D(printf("END pf_chk\n"));
    return ( tui_display_question ( Top, "Verify Function Key", str, -1, NULL, 0 ) );
}
```

```
/*****************************************************************************
 * MODULE NAME: plot_msid
 *
 *  This routine calculates the x,y pixel coordinates and plots all
 *  new values for the given plot until the end of the plot data file
 *  is reached, or an out-of-range value is encountered.
 *
 * DEVELOPMENT NOTES:
 *
 *  o   Log and Polar axes are not functional...only Cartesian axes
 *      are functional.
 *
 *
 * SPECIAL NOTES:
 *
 *  o   This function is not indented to show the structure of the
 *      function as a whole, rather each major section begins at the
 *      left margin.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Richard Romeo   - Ford Aerospace Corporation/Houston
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Ronnie Killough - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <X11/Xlib.h>
#include <wex/EXmsg.h>
#include <constants.h>
#include <disp.h>
#include <DDdisp.h>
#include <DDplot.h>


extern struct dm_shmemory *Dm_Address;  /* ptr to DM shared memory          */
extern struct bg_recs   Bg_Rec;         /* ptr thru all background records  */

extern union    p_data Data;            /* union structure for plot data    */

extern short    End_of_file;            /* global flag to aid in plot redraw */


plot_msid(disp_num, plot_ptr)

    short   disp_num;                   /* display number containing plot   */

    struct plot_ptrs    *plot_ptr;      /* ptr to effective plot            */

{
    register char   *data_buffer;       /* local ptr to data buffer         */

    static short    static_flag;        /* plot with static color           */
    static short    miss_flag;          /* plot with missing color          */

    Display     *xdisplay;              /* ptr to X display in DM sh mem    */
```

```
    Window      xwindow;                /* XID of display window in DM shm  */
    XPoint      point;                  /* new plot point                   */
    GC          gc;                     /* XID of GC in DM shared memory    */
    XGCValues   *gc_val;                /* ptr to GC values in DM sh memory */

    struct shm_decom    *decom_entry;   /* ptr to current decom entry       */
    struct plot_tmplt   *tmplt_ptr;     /* ptr to plot template structure   */
    struct msid_info    *msid_info;     /* ptr thru msid structure          */
    struct msid_info    *msid_pair;     /* ptr thru msid structure          */
    struct axis_info    *x_ptr;         /* ptr thru x axis records          */
    struct axis_info    *y_ptr;         /* ptr thru y axis records          */

    double  xvalue, yvalue,             /* temp in comp. of x/y log values  */
            save_data,                  /* 1st polar data value calculated  */
            scale_ratio,                /* used for y angle polar processing*/
            *new_scale,                 /* new scale val for main msid axis */
            *pr_new_scale,              /* new scale val for pair msid axis */
            low_value,                  /* main msid axis low scale value   */
            high_value,                 /* main msid axis high scale value  */
            x_low_value,                /* new low val for x log processing */
            y_low_value,                /* new low val for y log processing */
            temp_low_value,             /* if low scale value is greater    */
                                        /* than the high scale value, this  */
                                        /* value contains the high value    */
            temp_high_value,            /* and this value contains the low  */
                                        /* scale value.                     */
            pr_low_value,               /* pair msid axis low scale value   */
            pr_high_value;              /* pair msid axis high scale value  */

    float   char_height,                /* height of char. to be plotted    */
            xpoint, ypoint,             /* intermediate calculation temp    */
            factor_x, factor_y,         /* coord transformation factors     */
            line_width,                 /* current data plot line width     */
            x_wc_pt, y_wc_pt;           /* temporary world coordinate points*/

    unsigned long   gc_mask;            /* mask for GC change values        */

    long    status;                     /* status of plot data value        */

    int     axis_type, pr_axis_type,    /* main/pair axis type              */
            update = NO,                /* set if update to screen occurs   */
            k,                          /* loop count variable              */
            bytes_read,                 /* # bytes read from plot data file */
            bytes_to_read;              /* # bytes requested f/ plt data fl */

    short   loc_stat_flag,              /* local static flag                */
            plot_color,                 /* current data plot color          */
            line_type,                  /* current data plot line type      */
            x_axis_flag = NO,           /* flag set if main msid axis is x  */
            plot_flag,                  /* negated if current value will    */
                                        /* not be plotted.                  */
            done,                       /* set when finished processing     */
            xmultiply = 1,              /* set to 1 if low < high scale     */
            ymultiply = 1,              /* set to -1 if low > high scale    */
            *auto_scale,                /* rescale low scale or high scale  */
            *pr_auto_scale,             /* rescale low scale or high scale  */
            auto_flag, pr_auto_flag;    /* main/pair auto scale flag        */

    char    *strt_of_data;              /* strt of msid data in data buffer */

    D(printf("START plot_msid\n"));

/*
 *  Set up initial pointers for active plot.
```

```
 */

    tmplt_ptr = plot_ptr->plot_pos;
    bytes_to_read = plot_ptr->buf_size;
    data_buffer = plot_ptr->plot_data;

/*
 *  Set up local graphics context variables
 *  and world coord transformation factors
 */

    xdisplay = Dm_Address->xdisplay[disp_num];
    xwindow = XtWindow(plot_ptr->draw_win);
    gc = Dm_Address->gc[disp_num];
    gc_val = &Dm_Address->gc_val[disp_num];

    factor_x = tmplt_ptr->factor_x;
    factor_y = tmplt_ptr->factor_y;

/*****************************************************************************
 *  Loop through the plot data log file until end-of-file
 *  or until the done flag is set.
 *****************************************************************************/

    done = NO;

    while (done == NO) {
        bytes_read = read(plot_ptr->plot_fp, data_buffer, bytes_to_read);

        if (bytes_read < bytes_to_read) {
            done = YES;
            End_of_file = YES;
            D(printf("END plot_msid *** End_of_file\n"));
            return(update);
        }

/*
 *      Loop through all actual msids.
 */

        for (k=0; k<plot_ptr->header->actual_msids; k++) {

/*
 *          Set up the local msid ptrs to
 *          access the msid information.
 */

            msid_info = plot_ptr->msids + k;
            msid_pair = msid_info->pair_ptr;

/*
 *          Check to be sure haven't already
 *          processed the msid pair.
 */

            if (msid_pair->msid_indx > k) {


/*****************************************************************************
 *  Setup local variables for msid pairs attributes and axis-dependent
 *  attributes needed for plotting the point.
 *****************************************************************************/

/*
```

```
    *  Main msid: x axis   Pair msid: y axis
    */

       if (msid_info->xory_axis == 'X') {
           x_ptr = plot_ptr->axis + msid_info->axis_num - 1;
           y_ptr = plot_ptr->axis + plot_ptr->header->xaxes_num
                                  + msid_pair->axis_num - 1;
           axis_type = x_ptr->axis_type;
           high_value = x_ptr->high_value;
           low_value = x_ptr->low_value;
           auto_scale = &(x_ptr->auto_scale);
           new_scale = &(x_ptr->new_scale);

           if (x_ptr->auto_flag == 'Y')
               auto_flag = YES;
           else
               auto_flag = NO;

           pr_axis_type = x_ptr->axis_type;
           pr_high_value = y_ptr->high_value;
           pr_low_value = y_ptr->low_value;
           pr_auto_scale = &(y_ptr->auto_scale);
           pr_new_scale = &(y_ptr->new_scale);

           if (y_ptr->auto_flag == 'Y')
               pr_auto_flag = YES;
           else
               pr_auto_flag = NO;

    /*
     *  Main msid: y axis   Pair msid: x axis
     */

       } else {

           x_ptr = plot_ptr->axis + msid_pair->axis_num - 1;
           y_ptr = plot_ptr->axis + plot_ptr->header->xaxes_num
                          + msid_info->axis_num - 1;
           axis_type = y_ptr->axis_type;
           high_value = y_ptr->high_value;
           low_value = y_ptr->low_value;
           auto_scale = &(y_ptr->auto_scale);
           new_scale = &(y_ptr->new_scale);

           if (y_ptr->auto_flag == 'Y')
               auto_flag = YES;
           else
               auto_flag = NO;

           pr_axis_type = y_ptr->axis_type;
           pr_high_value = x_ptr->high_value;
           pr_low_value = x_ptr->low_value;
           pr_auto_scale = &(x_ptr->auto_scale);
           pr_new_scale = &(x_ptr->new_scale);

           if (x_ptr->auto_flag == 'Y')
               pr_auto_flag = YES;
           else
               pr_auto_flag = NO;

       }

    /*
     *  Axis dependent value initialization
```

```
 */

    if (x_ptr->low_value > x_ptr->high_value)
        xmultiply = -1;
    else
        xmultiply = 1;

    x_low_value = x_ptr->low_value * xmultiply;

    if (y_ptr->low_value > y_ptr->high_value)
        ymultiply = -1;
    else
        ymultiply = 1;

    y_low_value = y_ptr->low_value * ymultiply;

    if (low_value > high_value) {
        temp_low_value = high_value;
        temp_high_value = low_value;
    } else {
        temp_low_value = low_value;
        temp_high_value = high_value;
    }


/*****************************************************************************
 *  Continue only if both axes are active.  Initialize plot flag to yes.
 *****************************************************************************/

    if (x_ptr->axis_active == YES && y_ptr->axis_active == YES) {

        plot_flag = YES;


/*****************************************************************************
 *  Extract data from the data buffer using the decom information and
 *  set the status flags.
 *****************************************************************************/

/*
 *  Update decom entry ptr to point to decom entry for current msid
 */

    decom_entry = plot_ptr->plt_decom + k;

/*
 *  Check to be sure there is no error in the decom entry
 */

    if (decom_entry->error == NULL) {

/*
 *  Extract data pt from data buf
 */

    strt_of_data = data_buffer + decom_entry->offset + 2;
    status = extract(strt_of_data, decom_entry);
    p_dataval(decom_entry);

/*
 *  Check status of data and set approp. flags
 */

    if (status & MISSING_DATA) {
```

```
        plot_flag = NO;
        miss_flag = YES;
    }

    if (status & DEAD_DATA)
        plot_flag = NO;

    if (status & STATIC_DATA) {
        loc_stat_flag = YES;

        /* set static flag on. When the data starts back
         * up, the line between the last pt before the
         * data went static and the first pt when data
         * comes back will be plotted in static color */

        static_flag = YES;
    } else
        loc_stat_flag = NO;


/*****************************************************************************
 *  If status check didn't negate plot flag, check if plot value is less than
 *  lowest plot-able value or higher than highest plot-able value.  If it is,
 *  either set flags to signal a rescale of the plot and exit (if auto-scale
 *  is enabled for this axis) or adjust the plot value to the low or high
 *  value axis and continue.
 *****************************************************************************/

    if (plot_flag != NO) {

/*
 *      If extracted data value is less than plot low value then
 *      setup variables to signal a rescale (if auto-scale is enabled)
 *      or adjust plot value to plot along low value axis.
 */

        if (Data.ddata < temp_low_value) {

            if (auto_flag == YES && axis_type != POLAR) {
                done = YES;
                plot_flag = NO;

                if (low_value > high_value)
                    *auto_scale = NEW_HIGH_SCALE;
                else
                    *auto_scale = NEW_LOW_SCALE;

                if (Data.ddata > 0)
                    *new_scale = Data.ddata * .7;
                else
                    *new_scale = Data.ddata + (Data.ddata * .3);

            } else
                Data.ddata = temp_low_value;

/*
 *      If extract data value is greater than plot high value then
 *      setup variables to signal a rescale (if auto-scale is enabled)
 *      or adjust plot value to plot along high value axis.
 */

        } else if (Data.ddata > temp_high_value) {

            if (auto_flag == YES) {
```

```
                done = YES;
                plot_flag = NO;

                if (low_value > high_value)
                    *auto_scale = NEW_LOW_SCALE;
                else
                    *auto_scale = NEW_HIGH_SCALE;

                if (Data.ddata > 0)
                    *new_scale = Data.ddata + (Data.ddata * .3);
                else
                    *new_scale = Data.ddata * .7;

            } else
                Data.ddata = temp_high_value;
        }


/*****************************************************************************
 *  Compute the X pixel coordinate plot point for main msid on the
 *  appropriate axis.
 *****************************************************************************/

/*
 *  Main msid is on X axis...compute x coordinate
 */

    if (msid_info->xory_axis == 'X' && plot_flag == YES) {

/*
 *      If processing logarithmic type axis, then use
 *      the logarithmic calculation for the x point
 */

        if (axis_type == LOGARITHMIC) {

            xvalue = (double) (Data.ddata * xmultiply) - x_low_value;
            xpoint = ((float) sqrt ((double) xvalue)) / x_ptr->logval * 100;
            point.x = (short) ((xpoint * x_ptr->scale_ratio) * factor_x);

/*
 *      If processing polar type axis, save the data value for later
 *      calculations. We will calculate both points when the second value
 *      is received--we need both values to calculate each point.
 */

        } else if (axis_type == POLAR)

                save_data = Data.ddata;

/*
 *      Cartesian axis
 */

        else
            point.x = (short) ((Data.ddata - x_ptr->low_value)
                        * x_ptr->scale_ratio * (double) factor_x);

        x_axis_flag = YES;

/*
 *  Main msid is on Y axis...compute y coordinate
 */
```

```
    } else {

/*
 *      If processing logarithmic type axis, then use
 *      the logarithmic calculation for the x point
 */

        if (axis_type == LOGARITHMIC) {

            yvalue = (double) (Data.ddata * ymultiply) - y_low_value;
            ypoint = ((float) sqrt ((double) yvalue)) / y_ptr->logval * 100;
            point.y = (short) ((100.0 - (ypoint * y_ptr->scale_ratio))
                               * factor_y);

/*
 *      If processing polar type axis, save the data value for later
 *      calculations. We will calculate both points when the second value
 *      is received--we need both values to calculate each point.
 */

        } else if (axis_type == POLAR)

            save_data = Data.ddata;

/*
 *      Cartesian axis
 */

        else
            point.y = (short) ((100.0 - ((Data.ddata - y_ptr->low_value)
                               * y_ptr->scale_ratio)) * factor_y);
    }


/*****************************************************************************
 *  If main msid is begin plotted against local time, calculate the coordinate
 *  on the time axis and check for time expiring on the time axis.
 *****************************************************************************/

    if (strcmp(msid_pair->msid_name, LOCAL_TIME) == 0) {

/*
 *      Main value was an x axis msid, so need y axis coordinate
 */

        if (x_axis_flag == YES) {

/*
 *          Set local low and high plot value variables
*/

            if (y_ptr->low_value > y_ptr->high_value) {
                temp_low_value = y_ptr->high_value;
                temp_high_value = y_ptr->low_value;
            } else {
                temp_low_value = y_ptr->low_value;
                temp_high_value = y_ptr->high_value;
            }

/*
 *          If time elapsed is less than the start time, turn off
 *          plot flag.  If time expiration, turn off plot flag and
 *          signal end of plot.
 */
```

```
            if (plot_ptr->seconds_elapsed < temp_low_value)
                plot_flag = NO;
            else if (plot_ptr->seconds_elapsed > temp_high_value) {
                msid_info->first_pt = YES;
                y_ptr->end_of_plot = YES;
                plot_flag = NO;

/*
 *              If end of plot code is wrap or rescale, set done
 *              flag so will exit this routine for redraw of plot.
 */

                if (y_ptr->end_code >= 4)
                    done = YES;

/*
 *          If no time expiration, compute y coord for data value
 */

            } else {

/*
 *              If processing logarithmic type axis, then use the
 *              logarithmic calculation for the y point.
 */

                if (pr_axis_type == LOGARITHMIC) {

                    yvalue = (double) (plot_ptr->seconds_elapsed
                                       * ymultiply) - y_low_value;
                    ypoint = ((float) sqrt ((double) yvalue))
                                       / y_ptr->logval * 100;
                    point.y = (short) ((100.0 - (ypoint * y_ptr->scale_ratio))
                                     * factor_y);

/*
 *              Cartesian axis
 */

                } else

                    point.y = (short) ((100.0 - ((plot_ptr->seconds_elapsed
                        - y_ptr->low_value) * y_ptr->scale_ratio))
                        * factor_y);
            }

            x_axis_flag = NO;

/*
 *      Main value was a y axis msid, so need an x axis coordinate
 */

        } else {

/*
 *          Set temp low and high values
 */

            if (x_ptr->low_value > x_ptr->high_value) {
                temp_low_value = x_ptr->high_value;
                temp_high_value = x_ptr->low_value;
            } else {
                temp_low_value = x_ptr->low_value;
```

```
            temp_high_value = x_ptr->high_value;
        }

/*
 *          If time elapsed is less than the start time, turn off
 *          plot flag.  If time expiration, turn off plot flag and
 *          signal end of plot.
 */

        if (plot_ptr->seconds_elapsed < temp_low_value)
            plot_flag = NO;
        else if (plot_ptr->seconds_elapsed > temp_high_value) {
            x_ptr->end_of_plot = YES;

/*
 *              If end of plot code is wrap or rescale, set done
 *              flag so will exit this routine for redraw of plot.
 */

            if (x_ptr->end_code >= 4)
                done = YES;
                msid_info->first_pt = YES;
                plot_flag = NO;

/*
 *          If no time expiration, compute x coord for data value
 */

        } else {

/*
 *              If processing logarithmic type axis, then use
 *              the logarithmic calculation for the x point
 */

            if (pr_axis_type == LOGARITHMIC) {

                xvalue = (double) (plot_ptr->seconds_elapsed
                                   * xmultiply) - x_low_value;
                xpoint = ((float) sqrt ((double) xvalue))
                                   / x_ptr->logval * 100;
                point.x = (short)((xpoint * x_ptr->scale_ratio) * factor_x);

/*
 *              If processing polar type axis, then calculate
 *              both points using save_data for the y point.
 */

            } else if (pr_axis_type == POLAR) {

                xpoint = plot_ptr->seconds_elapsed;
                ypoint = save_data * (PI / 180.0);

                point.x = (short) ((((xpoint * (float) cos((double)ypoint))
                                * x_ptr->scale_ratio) + 50.0) * factor_x);

                scale_ratio =
                    100.0 / ((x_ptr->high_value - x_ptr->low_value) * 2);

                point.y = (short) ((((xpoint * (float) sin((double) ypoint))
                                * scale_ratio) + 50.0) * factor_y);

/*
 *              Cartesian axes
```

```
 */

                } else
                    point.x = (short)
                        ((plot_ptr->seconds_elapsed - x_ptr->low_value)
                        * x_ptr->scale_ratio * factor_x);

            }   /* end no time expiration */

        }   /* end y axis msid */


/*****************************************************************************
 *  If main msid is not being plotted against local time, it is being plotted
 *  against another msid...extract data from buffer for the pair axis msid.
 *****************************************************************************/

    } else {

/*
 *      Update decom entry pointer to point to pair msid
 */

        decom_entry = plot_ptr->plt_decom + msid_pair->msid_indx;

        if (decom_entry->error == NULL) {

/*
 *          Extract data from buffer for pair msid
 */

            strt_of_data = data_buffer + decom_entry->offset + 2;
            status = extract (strt_of_data, decom_entry);
            p_dataval(decom_entry);

            if (status & MISSING_DATA) {
                plot_flag = NO;
                miss_flag = YES;
            }

            if (status & DEAD_DATA)
                plot_flag = NO;

            if (status & STATIC_DATA) {
                loc_stat_flag = YES;

/*
 *                  Set static flag on. When the data starts back
 *                  up, the line between the last point before the
 *                  data went static and the first point when data
 *                  comes back, will be plotted in static color
 */

                static_flag = YES;

            } else
                loc_stat_flag = NO;


/*****************************************************************************
 *  If status check didn't negate plot flag, check if pair axis plot value is
 *  less than lowest plot-able value or higher than highest plot-able value.
 *  If it is,  either set flags to signal a rescale of the plot and exit
 *  (if auto-scale is enabled for this axis) or adjust the plot value to
```

```
 *  the low or high value axis and continue.
 ******************************************************************************/

    if (plot_flag == YES) {

/*
 *      Setup temp hi/low values
 */

        if (pr_low_value > pr_high_value) {
            temp_low_value = pr_high_value;
            temp_high_value = pr_low_value;
        } else {
            temp_low_value = pr_low_value;
            temp_high_value = pr_high_value;
        }

/*
 *      If extract data value is less than plot low value then
 *      setup variables to signal a rescale (if auto-scale is enabled)
 *      or adjust plot value to plot along low value axis.
 */

        if (Data.ddata < temp_low_value)

            if (pr_auto_flag == YES && pr_axis_type != POLAR) {
                done = YES;
                plot_flag = NO;

                if (pr_low_value > pr_high_value)
                    *pr_auto_scale = NEW_HIGH_SCALE;
                else
                    *pr_auto_scale = NEW_LOW_SCALE;

                *pr_new_scale = Data.ddata + (Data.ddata *.3);

            } else
                Data.ddata = temp_low_value;

/*
 *      If extract data value is greater than plot high value then
 *      setup variables to signal a rescale (if auto-scale is enabled)
 *      or adjust plot value to plot along high value axis.
 */

        else if (Data.ddata > temp_high_value)

            if (pr_auto_flag == YES) {
                done = YES;
                plot_flag = NO;

                if (pr_low_value > pr_high_value)
                    *pr_auto_scale = NEW_LOW_SCALE;
                else
                    *pr_auto_scale = NEW_HIGH_SCALE;

                *pr_new_scale = Data.ddata + (Data.ddata *.3);

            } else
                Data.ddata = temp_high_value;

/******************************************************************************
 *  If plot flag was not negated by the range check, compute the X pixel
```

```
 *  coordinate plot point for the pair msid on the appropriate axis.
 ******************************************************************************/

    if (plot_flag == YES) {

/*
 *      Main msid was a Y axis msid...compute an x axis coordinate.
 */

        if (x_axis_flag == NO) {

/*
 *          If processing logarithmic type axis, then use the
 *          logarithmic calculation for the x point.
 */

            if (pr_axis_type == LOGARITHMIC) {
                xvalue = (double) (Data.ddata * xmultiply) - x_low_value;
                xpoint = ((float) sqrt ((double) xvalue))
                                        / x_ptr->logval * 100;
                point.x = (short) ((xpoint * x_ptr->scale_ratio) * factor_x);

/*
 *          If processing polar type axis, then calculate both
 *          points using save_data for the y point.
 */

            } else if (pr_axis_type == POLAR) {
                xpoint = Data.ddata;
                ypoint = save_data * (PI / 180.0);
                point.x = (short) ((((xpoint * (float) cos((double)ypoint))
                                * x_ptr->scale_ratio) + 50.0) * factor_x);

                scale_ratio =
                        100.0 / ((x_ptr->high_value - x_ptr->low_value) * 2);

                point.y = (short) ((((xpoint * (float) sin((double) ypoint))
                                * scale_ratio) + 50.0) * factor_y);

/*
 *          Cartesian axis
 */

            } else
                point.x = (short) ((Data.ddata - x_ptr->low_value)
                    * x_ptr->scale_ratio * factor_x);

/*
 *      Main msid was an x axis msid...compute y axis coordinate
 */

        } else {

            x_axis_flag = NO;

/*
 *          If processing logarithmic type axis, then use the
 *          logarithmic calculation for the y point.
 */

            if (pr_axis_type == LOGARITHMIC) {
                yvalue = (double) (Data.ddata * ymultiply) - y_low_value;
                ypoint = ((float) sqrt ((double) yvalue))
                                        / y_ptr->logval * 100;
```

```
                point.y = (short) ((100.0 - (ypoint * y_ptr->scale_ratio))
                                    * factor_y);


/*
 *          If processing polar type axis, then calculate both
 *          points using save_data for the x point.
 */

            } else if (pr_axis_type == POLAR) {
                xpoint = save_data;
                ypoint = Data.ddata * (PI / 180.0);

                point.x = (short) ((((xpoint * (float) cos((double)ypoint))
                                    * x_ptr->scale_ratio) + 50.0) * factor_x);

                scale_ratio =
                        100.0 / ((x_ptr->high_value - x_ptr->low_value) * 2);

                point.y = (short) ((((xpoint * (float) sin((double) ypoint))
                                    * scale_ratio) + 50.0) * factor_y);

/*
 *          Cartesian axis
 */

            } else
                point.y = (short) ((100.0 - ((Data.ddata - y_ptr->low_value)
                            * y_ptr->scale_ratio)) * factor_y);

        }   /* end y axis */
    }   /* end plot flag check after 2nd axis range check */
    }   /* end plot flag check after status check of 2nd axis msid */

/*****************************************************************************
 *  If decom entry error on 2nd axis msid, negate plot flag
 *****************************************************************************/

        }   /* end of decom error check for 2nd axis msid */
        else
            plot_flag = NO;

    }   /* end of 2nd axis msid calculation */
    }   /* end plot flag check after status check of 1st axis msid */

/*****************************************************************************
 *  If decom entry error on 1st axis msid, negate plot flag
 *****************************************************************************/

        }   /* end of decom error check for 1st axis msid */
        else
            plot_flag = NO;

/*****************************************************************************
 *  If data is static and user does not request static data to be plotted,
 *  negate plot flag.
 *****************************************************************************/

/* RLK 9/17/90  Need to figure out how these static flags work and make sure
                they work right.   Goal: if lines connect the plot pts, is to
                plot the static color up to the pt where the data went
                non-static (this may also apply to missing data). */
```

```
    if ((msid_info->stat_flag == 0) && (loc_stat_flag == YES))
        plot_flag = NO;
    else
        loc_stat_flag = NO;

/*****************************************************************************
 *  If plot flag has not been negated, set up the graphics context for the
 *  plot and plot the value.
 *****************************************************************************/

    if (plot_flag == YES) {

        update = YES;

/*
 *      Set the color, line type, and line width
 */

        plot_color = msid_info->plot_color;

        if (status & STATIC_DATA)
            plot_color = msid_info->stat_color;

        if ((status & CRITICAL_HIGH) || (status & CRITICAL_LOW)) {

            plot_color = msid_info->crit_color;
            line_type = msid_info->crit_type;
            line_width = msid_info->crit_width;

        } else if ((status & LIMIT_HIGH) || (status & LIMIT_LOW)) {

            plot_color = msid_info->limt_color;
            line_type = msid_info->oper_type;
            line_width = msid_info->oper_width;

        } else if (plot_ptr->ovl_color_flg == YES) {

            plot_color = msid_info->ovl_color;

        } else {

            line_type = msid_info->line_type;
            line_width = msid_info->line_width;

        }

/*
 *          Adjust the value by the zoom offset
 *          (zero if zoom is not in effect).
 */

            point.x = point.x + tmplt_ptr->offset_x;
            point.y = point.y + tmplt_ptr->offset_y;

/*
 *      If this is not the first value on this plot, plot the point.
 */

        if (msid_info->first_pt == NO) {

/*
 *          Plot the value based on plot type
 */
```

```
        switch (msid_info->plot_type) {

/*
 *          If plot type is LINE, connect the previous point and
 *          the current point with a line.
 */

            case ('L'): /* LINE */

                if (miss_flag == YES) {
                    plot_color = msid_info->miss_color;
                    miss_flag = NO;
                }

                if (static_flag == YES) {
                    plot_color = msid_info->stat_color;
                    static_flag = NO;
                }

                if (gc_mask = set_gc(xdisplay, gc, gc_val, plot_color,
                                line_type, line_width,
                                NO_CHANGE, NO_CHANGE, NO_CHANGE, NO_CHANGE))
                    XChangeGC(xdisplay, gc, gc_mask, gc_val);

                XDrawLine(xdisplay, xwindow, gc,
                        msid_info->prev_pt_x, msid_info->prev_pt_y,
                        point.x, point.y);

                break;

/*
 *          If plot type is CHARACTER, set up the font and then
 *          determine whether to connect the plot characters and
 *          whether to erase previous plot characters based on
 *          the plot connection code.
 */

            case ('C'): /* CHARACTER */

/*
 *              Plot the value as a character.  If plot connection
 *              indicates, erase previous plot point and/or connect
 *              the previous and current plot points with a line.
 *
 *              NOTE: this is a progressive switch...there is no
 *              -break- stmt until the end of the last case.
 */

                switch (msid_info->plot_conn) {

/*
 *                  Connect points erasing prev. character
 */

                    case ('E'):

                        XSetBackground(xdisplay, gc, Bg_Rec.s_color);

                        XDrawString(xdisplay, xwindow, gc,
                            msid_info->prev_pt_x,
                            msid_info->prev_pt_y,
                            msid_info->plot_char, 1);

                        XSetBackground(xdisplay, gc, plot_color);
```

```
/*
 *                      Connect plot points with a line
 */

                        case ('C'):

                            if (miss_flag == YES) {
                                plot_color = msid_info->miss_color;
                                miss_flag = NO;
                            }

                            if (static_flag) {
                                plot_color = msid_info->stat_color;
                                static_flag = NO;
                            }

                            if (gc_mask = set_gc(xdisplay, gc, gc_val,
                                          plot_color, line_type,
                                          line_width, NO_CHANGE,
                                          NO_CHANGE, NO_CHANGE, NO_CHANGE))
                                XChangeGC(xdisplay, gc, gc_mask, gc_val);

                            XDrawLine(xdisplay, xwindow, gc,
                                msid_info->prev_pt_x,
                                msid_info->prev_pt_y,
                                point.x, point.y);

/*
 *                      Plot discrete values but don't connect plot points
 */

                        case ('D'):

                            if (gc_mask = set_gc(xdisplay, gc, gc_val,
                                        plot_color, NO_CHANGE, -1.0,
                                        NO_CHANGE, NO_CHANGE, NO_CHANGE,
                                        msid_info->plot_font))
                                XChangeGC(xdisplay, gc, gc_mask, gc_val);

                            XDrawString(xdisplay, xwindow, gc, point.x,
                                        point.y, msid_info->plot_char, 1);

                            break;

                        default:
                            break;

                    } /* end of plot connection of characters switch */

                default:
                    break;

            } /* end of plot type switch statement */

            msid_info->prev_pt_x = point.x;
            msid_info->prev_pt_y = point.y;

/*
 *      If initial point, setup previous point variables and
 *      negate first point flag.
 */

        } else {
```

```
            msid_info->first_pt = NO;
            msid_info->prev_pt_x = point.x;
            msid_info->prev_pt_y = point.y;
        }

    }   /* end of plot the value */

    }   /* end of both axis active */

            }   /* end of -if- msid_indx > k */

        }   /* end of -for- loop thru actual msids */

        plot_ptr->seconds_elapsed =
                plot_ptr->seconds_elapsed + plot_ptr->header->upd_rate -.15;

    }   /* end (while done == NO) */

    D(printf("END plot_msid\n"));

    return (update);
}
```

```
/*****************************************************************************
 * MODULE NAME: proc_plt
 *
 *  This routine calls the routine which plots the next point(s) on the
 *  given plot, and then checks to see if the plot needs to be redrawn
 *  due to the plot reaching the end of the display area or a data point
 *  has been calculated which lies outside the plot scales.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Richard Romeo   - Ford Aerospace Corporation/Houston
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Ronnie Killough - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <X11/Xlib.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/timeb.h>
#include <unistd.h>
#include <constants.h>
#include <disp.h>
#include <DDdisp.h>
#include <DDplot.h>
#include <wex/EXmsg.h>


extern struct dm_shmemory   *Dm_Address;    /* addr of DM shared memory     */


proc_plt(disp_num, plot_ptr)

    short   disp_num;                   /* effective display number         */

    struct plot_ptrs   *plot_ptr;       /* ptr to effective plot record     */
{

    struct axis_info    *axis_ptr;      /* ptr thru axis records info       */
    struct plot_tmplt   *tmplt_ptr;     /* ptr thru tmplt ptr info          */
    struct msid_info    *loc_msid_info; /* ptr thru msid info               */

    double  scale_diff;                 /* diff betwn hi & lo scale values  */

    short   loc_rewind;                 /* rewind flag                      */
    short   redraw_flg;                 /* redraw flag                      */
    short   low_flag;                   /* low value is gtr than high value */

    int     k, m;                       /* loop count variable              */
    int     total_nbr_axis;             /* total nbr of plot axes           */
    int     update;                     /* return flag from plot_msid       */


    D(printf("START proc_plt\n"));

/*
```

```
 *  Initialize flags
 */

    update = NO;
    redraw_flg = NO;
    loc_rewind = NO;
    low_flag = NO;

/*
 *  Call plot msid routine to plot next point(s) for this plot
 */

    update = plot_msid (disp_num, plot_ptr);

/*
 *  Check for plot point reaching end of plot area...process according
 *  to end code (for time plots) or auto-scale (for msid/msid plots).
 */

    total_nbr_axis = plot_ptr->header->xaxes_num + plot_ptr->header->yaxes_num;

/*
 *  Loop through each axis for this plot
 */

    for (m = 0; m < total_nbr_axis; m++) {

        axis_ptr = plot_ptr->axis + m;

/*
 *      Only process active axes
 */

        if (axis_ptr->axis_active == YES) {

/*
 *          If current axis is a time axis and end-of-plot
 *          has been signalled by plot_msid(), process the end code.
 */

            if ((axis_ptr->scal_type == 'T')
                    && (axis_ptr->end_of_plot == YES)) {

                if (axis_ptr->low_value > axis_ptr->high_value)
                    low_flag = YES;
                else
                    low_flag = NO;

/*
 *              If end code specifies audible and/or visible
 *              advisories, issue those advisories.
 */

                switch (axis_ptr->end_code) {

                    case PLOT_BELL_STOP:
                    case PLOT_BELL_RESCALE:
                    case PLOT_BELL_WRAP:
                        printf ("\007\n");
                        break;

                    case PLOT_ADV_STOP:
                    case PLOT_ADV_RESCALE:
                    case PLOT_ADV_WRAP:
```

```
                        tui_msg(M_YELLOW, "Time range on %c axis %d has been reached", axi
s_ptr->axis_xory, axis_ptr->axis_num);
                        break;

                    case PLOT_BELL_ADV_STOP:
                    case PLOT_BELL_ADV_RESCALE:
                    case PLOT_BELL_ADV_WRAP:
                        printf ("\007\n");
                        tui_msg(M_YELLOW, "Time range on %c axis %d has been reached", axi
s_ptr->axis_xory, axis_ptr->axis_num);
                        break;

                    default:
                        break;

                }   /* end of switch on end code advisories */

/*
 *              Determine whether to stop, rescale, or wrap the
 *              plot based on end code.
 */

                switch (axis_ptr->end_code) {

/*
 *                  If end code specifies is STOP, deactivate the axis
 */

                    case PLOT_STOP:
                    case PLOT_BELL_STOP:
                    case PLOT_ADV_STOP:
                    case PLOT_BELL_ADV_STOP:
                        axis_ptr->axis_active = NO;
                        break;

/*
 *                  If end code specifies a RESCALE, double the
 *                  high or low scale value for this axis and set
 *                  the redraw flag.
 */

/* RLK 9/28/90  Need to check to see if these calculations are right...
                if the low scale value > high scale value, why double
                the low scale value?  How is that going to rescale
                the plot properly?  What if the high scale value is zero?
                This needs work.  Suppose the low scale value is 60 secs
                and the high scale value is 0 secs (countdown to 0) and
                60 seconds elapses.  Doubling the low scale value will
                double the amount of time shown, but will right justify
                the current plot on the new 120...0 scale, when it should
                be left-justified on a 60...0...-60 scale. */

                    case PLOT_RESCALE:
                    case PLOT_BELL_RESCALE:
                    case PLOT_ADV_RESCALE:
                    case PLOT_BELL_ADV_RESCALE:

                        if (low_flag)
                            axis_ptr->low_value =
                                        axis_ptr->low_value * 2;
                        else
                            axis_ptr->high_value =
                                        axis_ptr->high_value * 2;
```

```
                    redraw_flg = YES;
                    loc_rewind = YES;
                    axis_ptr->end_of_plot = NO;
                    break;

/*                  If end code specifies a WRAP, set high scale value to
 *                  the low scale value and add the difference between the
 *                  the original high and low scale values to the new
 *                  low scale value to get the new high scale value.  Set the
 *                  redraw flag.
 */

                    case PLOT_WRAP:
                    case PLOT_BELL_WRAP:
                    case PLOT_ADV_WRAP:
                    case PLOT_BELL_ADV_WRAP:

                        if (low_flag)
                            scale_diff = (axis_ptr->low_value
                                          - axis_ptr->high_value) * .75;
                        else
                            scale_diff = (axis_ptr->high_value
                                          - axis_ptr->low_value) * .75;

                            axis_ptr->low_value = axis_ptr->low_value
                                                  + scale_diff;
                            axis_ptr->high_value = axis_ptr->high_value
                                                   + scale_diff;

/*
                        axis_ptr->low_value = axis_ptr->high_value;
                        axis_ptr->high_value = axis_ptr->low_value
                                    + scale_diff;
*/

                        redraw_flg = YES;
                        loc_rewind = YES;
                        axis_ptr->end_of_plot = NO;
                        break;

                    default:
                        break;

                }   /* end of switch on end code processing */

/*
 *          If axis is not a time axis, it is an msid axis.  If an
 *          auto-scale has been signalled by plot_msid() and
 *          auto-scaling is enabled for this axis, adjust
 *          the scales and set the redraw flag.
 */

            } else if ((axis_ptr->auto_scale != NO)
                    && (axis_ptr->auto_flag == 'Y')) {

/*
 *              If auto-scale indicates the data value was off the
 *              low scale, adjust the low scale value.
 */

                if (axis_ptr->auto_scale == NEW_LOW_SCALE)
                    axis_ptr->low_value = axis_ptr->new_scale;

/*
```

```
 *              If auto-scale indicates the data value was off the
 *              high scale, adjust the high scale value.
 */

                else
                    axis_ptr->high_value = axis_ptr->new_scale;


                axis_ptr->auto_scale = NO;
                loc_rewind = YES;
                redraw_flg = YES;
                loc_msid_info = plot_ptr->msids;

/*
 *              Reset all msids first point flag.
 */

                for (k = 0; k < plot_ptr->header->msid_num; k++) {
                    loc_msid_info->first_pt = YES;
                    loc_msid_info++;
                }

/*
 *          If not end-of-time-plot and no auto-scale has been signalled,
 *          negate auto scale since auto flag was NO
 */

            } else
                axis_ptr->auto_scale = NO;

        }   /* end of axis active flag */

    }   /* end of total nbr of axis  */

/*
 *  If redraw flag is set, erase the plot and redraw
 */

    if (redraw_flg == YES) {
        update = 1;
        tmplt_ptr = plot_ptr->plot_pos;

/*
 *      Erase the plot using the coordinates of the plot
 *      and the background color.
 */

        XClearArea(Dm_Address->xdisplay[disp_num], XtWindow(plot_ptr->draw_win),
                        0, 0, tmplt_ptr->drw_width, tmplt_ptr->drw_height,
                        False);

/*
 *      Rewind the plot file if needed
 */

        if (loc_rewind == YES) {
            lseek(plot_ptr->plot_fp, 0, SEEK_SET);
            lseek(plot_ptr->plot_fp,
                        80 + (plot_ptr->header->msid_num * 24), SEEK_SET);
            plot_ptr->seconds_elapsed = 0;
            loc_rewind = NO;
        } else
            plot_ptr->seconds_elapsed
                                -= (plot_ptr->header->upd_rate << 1);
```

```
/*
 *      Redraw the plot axes.
 */

        draw_plt(disp_num, plot_ptr);

/*
 *      If overlay flag set, draw overlay
 *

        if (plot_ptr->ovr_flg == YES)
            draw_ovl (plot_ptr);*/

/*
 *      Clear the redraw flag
 */

        redraw_flg = NO;

    }

/*
 *  Return 1 if update to display occurred
 */

    D(printf("END PROC_PLT\n"));
    return (update);

}
```

```
/*****************************************************************************
 * MODULE NAME: plot_ovl.c
 *
 *  This routine is invoked when the user selects either a predefined PF
 *  key or a menu selection for a plot save overlay or display overlay. The
 *  following display is drawn for save overlay or display overlay plots.
 *
 *
 * INTERNAL FUNCTIONS:
 *
 *      o   ovl_menu     -   This function displays the popup form which allows
 *                           the plot and overlay filenames to be entered.
 *
 *      o   cb_ovl       -   This callback function handles all callbacks
 *                           generated by the popup form.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  K. Noonan        - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <string.h>
#include <X11/Intrinsic.h>
#include <X11/Shell.h>
#include <Xm/SelectioB.h>
#include <Xm/Text.h>
#include <constants.h>
#include <disp.h>
#include <DDdisp.h>
#include <DDplot.h>
#include <pf_key.h>
#include <user_inter.h>
#include <wex/EXmsg.h>


extern Widget               Top;                /* Top level widget.              */
extern struct pfkey_defs    Current_Com;        /* Current commands definition.   */
extern struct dm_shmemory   *Dm_Address;        /* Shared memory address.         */
extern struct plot_ptrs     *Plot_info_ptr;     /* Ptr thru plot ptr files.       */

extern short                Disp_Num,           /* Display Manager number.        */
                            Nbr_of_plots;       /* Number of plots to display.    */

extern char                 Disp_Path[DNAME_LEN],  /* Path of displays.           */
                            Plot_Path[DNAME_LEN];  /* Path of plots.              */
extern int                  errno;


static Widget               shell,
                            t_plot,
                            t_ovl;

static char                 **list_plot = NULL,
```

```
                        **list_ovl  = NULL;

static int              num_ovls = 0,
                        flag;


int plot_ovl ( ready )

    short               ready;
{
    register char       *ptr,
                        *ptr1;

    struct plot_ptrs    *act_plot_ptr;      /* Ptr thru plot ptr files.            */
    struct disp_info    *display;           /* Ptr to display information table.   */

    short               i,                  /* Index counter.                      */
                        match,              /* YES if a match is found in lists.   */
                        access;             /* Access restriction code of the plot. */

    FILE                *fopen ( ),
                        *fp;                /* File ptr to plot file.              */

    char                plot_name      [DNAME_LEN + 4],
                        data_plot_name [DNAME_LEN + 4],
                        overlay_name   [DNAME_LEN + 4],
                        cmd            [110],
                        *malloc();

    D(printf("START plot_ovl\n"));
/*
 *  Save pointer to the display structure.
 */

    display = &Dm_Address->display[Disp_Num];
    list_plot = list_ovl = NULL;

/*
 *  If the number of plots for the current display is zero, return to the calling
 *  function.
 */

    if ( Nbr_of_plots == 0 ) {
        tui_msg ( M_YELLOW, "Current display does not have any plots" );
        return ( -1 );
    }

/*
 *  Build list of plots for the current display. This list is simply an array of
 *  pointers into the plot list. The individual pointers point directly to the file
 *  names without the paths.
 */

    if ( ( list_plot = (char **)malloc ( Nbr_of_plots * sizeof ( char * ) ) ) == NULL ) {
        tui_msg ( M_YELLOW, "Unable to allocate memory for list of plots" );
        return ( -1 );
    }

    for ( i = 0; i < Nbr_of_plots; i++ ) {
        ptr = (Plot_info_ptr + i)->plot_name;
        ptr1 = ptr + strlen ( ptr ) - 1;
        while ( ptr1 > ptr && *ptr1 != '/' )
            ptr1--;
        ptr1++;
```

```
        *(list_plot + i) = ptr1;
    }

/*
 *  Call (read_ovls) to read the directory and generate a list of overlay filenames. If
 *  this fails, return.
 */

    if ( ( num_ovls = read_ovls ( &list_ovl ) ) == -1 ) {
        free_lists ( );
        return ( -1 );
    }

/*
 *  If the command was to display an overlay and there are none available, generate a
 *  warning.
 */

    if ( num_ovls == 0 ) {
        if ( Current_Com.func_no == SAVE_OVRLAY )
            tui_msg ( M_YELLOW, "No plot data files available for display" );
        else
            tui_msg ( M_YELLOW, "No overlay files available for display" );

        free_lists ( );
        return ( -1 );
    }

/*
 *  If called from menu, display the popup form which allows entry of the display
 *  and overlay names.
 */

    if ( ready == NO )
        ovl_menu ( );

/*
 *  If called from the menu and the user aborted, return to calling function.
 */

    if ( ready == NO && flag == NO ) {
        free_lists ( );
        return ( -1 );
    }

/*
 *  Build the complete path of the plot if the path is not already given. If a
 *  plot data file path is "/WEX", then the data file is found in Plot_Path.
 */

    if ( Current_Com.disp_name[0] != '/' ) {
        if ( Current_Com.func_no == SAVE_OVRLAY ) {
            strcpy ( data_plot_name, Plot_Path              );
            strcat ( data_plot_name, Current_Com.disp_name );
        }
        strcpy ( plot_name, Disp_Path              );
        strcat ( plot_name, Current_Com.disp_name );
    } else {
        strcpy ( plot_name,      Current_Com.disp_name );
        strcpy ( data_plot_name, Current_Com.disp_name );
    }

    if ( Current_Com.ovr_name[0] != '/' ) {
        strcpy ( overlay_name, Plot_Path            );
```

```
        strcat ( overlay_name, Current_Com.ovr_name );
    } else
        strcpy ( overlay_name, Current_Com.ovr_name );

/*
 *  If the command is to save an overlay, then check to see if the plot is
 *  active. If the plot is active, reject the command.
 */

    if ( Current_Com.func_no == SAVE_OVRLAY ) {
        match = NO;
        i     = 0;
        while ( ( i < MAX_PLOTS ) && ( match == NO ) ) {
            if ( ( strcmp ( Dm_Address->plots.act_plots[i], plot_name ) ) == 0 ) {
                match = YES;
                tui_msg ( M_WHITE, "Plot %s is active - save overlay command rejected",
                        Current_Com.disp_name );
                free_lists ( );
                return ( -1 );
            } else
                i++;
        }

/*
 *      Open the plot file and retrieve the access code. If the open fails, generate
 *      an error and return.
 */

        strcat ( data_plot_name, ".pdt" );
        if ( ( fp = fopen ( data_plot_name, "r" ) ) == NULL ) {
            tui_msg ( M_YELLOW, "Error %d on opening plot data file %s", errno,
                    data_plot_name );
            free_lists ( );
            return ( -1 );
        }
        fscanf ( fp, "%70*c"        );
        fscanf ( fp, "%hd", &access );
        fclose ( fp );

/*
 *      Check to see if the access is restricted by MEDICAL or PAYLOAD users.
 *      If so, return.
 */

        if ( chk_res ( access, display->pos_id ) ) {
            free_lists ( );
            return ( -1 );
        }

/*
 *      Access has not been restricted, so build the system commands to
 *      move the plot data file over to the plot overlay file.
 */

        strncat ( overlay_name, ".ovr\0", 5 );
        sprintf ( cmd, "mv %s %s", data_plot_name, overlay_name );
        if ( system ( cmd ) )
            tui_msg ( M_YELLOW, "Error %d in saving plot overlay <%s>", errno,
                    data_plot_name );

/*
 *  Process the overlay display command. Search through the list of plots
 *  for a match with the specified plot. If a match is found, call (draw_ovl) to
 *  actually draw the overlay on the display. Upon return break out of the loop.
```

```
 */

    } else {
        act_plot_ptr = Plot_info_ptr;
        for ( i = 0; i < Nbr_of_plots; i++ ) {
            if ( ( strcmp ( plot_name, act_plot_ptr->plot_name ) ) == 0 ) {
                strcpy ( act_plot_ptr->plot_ovr, overlay_name );
                draw_ovl ( act_plot_ptr );
                break;
            } else
                act_plot_ptr++;
        }

/*
 *      If no match was found, generate a warning.
 */

        if ( i == Nbr_of_plots )
            tui_msg ( M_YELLOW, "Plot <%s> is not on this display", plot_name );
    }

/*
 *  Normal return.
 */

    free_lists ( );
    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: ovl_menu
 *
 *  This function initializes the popup form which allows the user to specify
 *  the plot and overlay file names.
 *****************************************************************************/

static int ovl_menu ( )
{
    register int    i;

    Arg             args[10];

    Widget          form,
                    f_data,
                    f_cmd;

    XtCallbackProc  cb_ovl();

    XEvent          event;

    char            *s;

    D(printf("START ovl_menu\n"));
/*
 *  Create the shell widget.
 */

    i = 0;
    s = ( Current_Com.func_no == PLOT_OVRLAY ) ? "Plot Overlay" : "Save Overlay";
    XtSetArg ( args[i], XmNtitle, s ); i++;

    shell = tui_create_trans_shell ( "Plot/Save Overlay", args, i );

/*
 *  Create the main and all sub-forms.
 */

    i = 0;
    form   = tui_create_form ( shell, "form",   TRUE,  args, i );
    f_data = tui_create_form ( form,  "f_data", FALSE, args, i );
    f_cmd  = tui_create_form ( form,  "f_cmd",  FALSE, args, i );

/*
 *  Create the two selection lists widgets for the plot and overlay filenames.
 */

    i = 0;
    t_plot = tui_create_sel ( f_data, "t_plot", list_plot, Nbr_of_plots,
                              "Plots in Display",
                               args, i );
    t_ovl  = tui_create_sel ( f_data, "t_ovl",  list_ovl,  num_ovls,
                    ( Current_Com.func_no == SAVE_OVRLAY ) ? "Data Files" : "Overlays",
                              args, i );

/*
 *  Create a separator widget.
 */

    i = 0;
    XtManageChild ( XmCreateSeparator ( form, "sep0", args, i ) );

/*
```

```
     *  Create the command widgets.
     */

        i = 0;
        tui_create_pushbutton ( f_cmd, "Cancel", cb_ovl, (caddr_t)0, args, i );
        tui_create_pushbutton ( f_cmd, "OK",     cb_ovl, (caddr_t)1, args, i );
        tui_create_pushbutton ( f_cmd, "Help",   cb_ovl, (caddr_t)2, args, i );

    /*
     *  Put all input widgets in a tab group.
     */

        XmAddTabGroup ( t_plot );
        XmAddTabGroup ( t_ovl  );

    /*
     *  Realize and popup the shell.
     */

        XtRealizeWidget ( shell );
        XtPopup ( shell, None );
        set_cmap ( shell );

    /*
     *  Wait until the user finishes with the popup.
     */

        flag = -1;
        while ( flag == -1 ) {
            XtNextEvent  ( &event );
            XtDispatchEvent ( &event );
        }

        XtDestroyWidget ( shell );

    /*
     *  Return the value selected by the user (0 is for not verified, 1 is for
     *  verified.
     */

        D(printf("END ovl_menu\n"));
        return ( flag );
    }
```

```
/*****************************************************************************
 * MODULE NAME: cb_ovl
 *
 *  This callback function processes the OK, CANCEL, and HELP callbacks from
 *  the popup form.
 *****************************************************************************/

static XtCallbackProc cb_ovl ( w, closure, calldata )

    Widget      w;              /* Set to widget which in which callback originated.    */

    caddr_t     closure,        /* Indicates selected command.                          */
                *calldata;      /* Widget-specific information.                         */
{
    char    *ptr;

    D(printf("START cb_ovl\n"));
/*
 *  Process the OK button. First save the plot filename in the current command structure
 *  and validate it. If invalid, return.
 */

    if ( (int)closure == 1 ) {
        strcpy ( Current_Com.disp_name, ptr = XmTextGetString ( t_plot ) );
        free ( ptr );
        if ( val_fn ( Current_Com.disp_name,
                      ( Current_Com.func_no == SAVE_OVRLAY ) ? NO : YES ) == 0 )
            return;

/*
 *      Save and validate the overlay filename. If valid, set the (flag) to the value
 *      of (closure) which will cause the popup to be removed.
 */

        strcpy ( Current_Com.ovr_name, ptr = XmTextGetString ( t_ovl ) );
        free ( ptr );
        if ( val_fn ( Current_Com.ovr_name, NO ) )
            flag = (int)closure;

/*
 *  Process CANCEL button. Simply set (flag) to the value of (closure) to cause
 *  removal of the form.
 */

    } else if ( (int)closure == 0 ) {
        flag = (int)closure;

/*
 *  If help button was selected, display the appropriate help text.
 */

    } else if ( (int)closure == 2 )
        cb_help ( 0, 19, 0 );

/*
 *  Normal return.
 */

    D(printf("END cb_ovl\n"));
    return;
}
```

```
/*****************************************************************************
 * MODULE NAME: free_lists
 *
 *  This function frees the list of plot and overlay files.
 *****************************************************************************/

int free_lists ( )
{
    register int    i;

/*
 *  Free the list of plots.
 */

    if ( list_plot )
        free ( (char *)list_plot );

/*
 *  Free the list of overlays.
 */

    if ( list_ovl ) {
        for ( i = 0; i < num_ovls; i++ )
            free ( *(list_ovl+i) );
        free ( (char *)list_ovl );
    }

/*
 *  Normal return.
 */

    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: read_disp.c
 *
 *  This function reads the displays in a directory and formats a list which
 *  is later used to present displays to the user. This list is retained
 *  locally so that it can be reused.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  K. Noonan       - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <constants.h>
#include <disp.h>
#include <stdio.h>
#include <fcntl.h>
#include <wex/EXmsg.h>


extern struct file_info *Disp_Info;             /* pointer to file information    */

extern char             Disp_Path[DNAME_LEN];   /* display path name              */

extern int              errno;                  /* system return error            */

extern short            Disp_Num;               /* display manager number         */


int read_disp ( )
{
    int                 num_disps = 0;          /* return number of displays      */

    struct file_info    *d_info_ptr;            /* pointer to file information    */

    FILE                *fp, *fp1, *fopen ( );  /* display file pointers          */

    char                *ptr,
                        file_name[DNAME_LEN + 4],/* display name                  */
                        disp_name[DNAME_LEN + 4],/* display name with path        */
                        *calloc ( ),            /* space allocation               */
                        str[80],                /* contains system command        */
                        temp_file[30];          /* temporary file name            */


    int                 j,                      /* loop indices                   */
                        num_files = 0,          /* number of files                */
                        length;                 /* file name length               */

    D(printf("START read_disp\n"));
/*
 *  Display wait cursor.
 */

    tui_start_wait ( );
```

```
/*
 *  Build the system commands to read all the display files in the display
 *  directory into a temporary file 'tdispX.dat'. ( ls [path] > tdisp.dat )
 */

    sprintf ( temp_file, "/user/display/tdisp%hd.dat", Disp_Num );
    sprintf ( str, "ls %s*.bg > %s", Disp_Path, temp_file );

    if ( system ( str ) ) {
        tui_msg ( M_YELLOW, "Error on reading display directory" );
        tui_stop_wait ( );
        return ( -1 );
    }

    if ( ( fp = fopen ( temp_file, "r" ) ) == NULL ) {
        tui_msg ( M_YELLOW, "Error %d on reading display directory file", errno );
        tui_stop_wait ( );
        return ( -1 );
    }

/*
 *  Read the display directory file for an initial count of the number of
 *  displays in the directory.
 */

    while ( fscanf ( fp, "%s", file_name ) != EOF )
        num_files++;

    rewind ( fp );

/*
 *  Allocate space for the display file name and description fields.
 */

    Disp_Info =
        ( struct file_info * ) calloc ( num_files + 1, sizeof ( struct file_info ) );

    if ( Disp_Info == NULL ) {
        tui_msg ( M_YELLOW, "Error %d on allocation of file info structure", errno );
        tui_stop_wait ( );
        return ( -1 );
    }

/*
 *  Save entry for DTE display.
 */

    d_info_ptr = Disp_Info;
    strncpy ( d_info_ptr, "DTE DISPLAY\0", 12 );
    num_disps = 1;
    d_info_ptr++;

/*
 *  Process each filename.
 */

    while ( fscanf ( fp, "%s", file_name ) != EOF ) {
        strcpy ( disp_name, file_name );

/*
 *      Extract just the filename.
 */
```

```
        ptr = file_name + strlen ( file_name ) - 1;
        while ( ptr > file_name && *ptr != '/' )
            ptr--;
        ptr++;

/*
 *      Open the filename.
 */

        if ( ( fp1 = fopen ( disp_name, "r" ) ) == NULL ) {
            free ( (char *)Disp_Info );
            tui_stop_wait ( );
            return( -1 );
        }

/*
 *      Read in the name field and blank the extension '.bg' and add in the ':'
 *      in column 8 and the 32-character description afterwards.
 */

        length = strlen ( ptr );
        for ( j = 8; j > length - 4; j-- )
            ptr[j] = ' ';

        strncpy ( d_info_ptr->name,     ptr,     8 );
        strncpy ( &d_info_ptr->name[8], " : \0", 4 );
        fscanf ( fp1, "%*3c" );
        fscanf ( fp1, "%32c", d_info_ptr->desc );
        d_info_ptr->inverse_flag = NO;

        d_info_ptr++;
        num_disps++;
        fclose ( fp1 );
    }

/*
 *  Close the temp file and remove it from system.
 */

    fclose ( fp );
    strncpy ( str, "rm \0", 4 );
    strcat ( str, temp_file );
    if ( system ( str ) )
        tui_msg ( M_YELLOW, "Error on removing temporary file" );

    tui_stop_wait ( );

    D(printf("END read_disp\n"));
    return ( num_disps );
}
```

```
/*****************************************************************************
 * MODULE NAME: read_fgr.c
 *
 *  This routine reads the foreground graphics records from the
 *  DDF foreground file into memory.
 *
 * DEVELOPMENT NOTES:
 *
 *  This routine has not been entirely translated to X.  Foreground graphics
 *  are not functional.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Robert Stanley  - Ford Aerospace Corporation/Houston
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Ronnie Killough - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <sys/types.h>
#include <X11/Xlib.h>
#include <wex/EXmsg.h>
#include <math.h>
#include <constants.h>
#include <disp.h>
#include <DDfg_graph.h>


extern struct dm_shmemory   *Dm_Address;     /* ptr to DM shared memory      */
extern struct fg_recs       Fg_rec;          /* fg graphics records          */

extern int  errno;

extern  short   Pixels[];                    /* index array into colormap    */


read_fgr(disp_num, ddf_ffp)

    short   disp_num;                        /* effective display number     */

    FILE           *ddf_ffp;                 /* ptr to open fg DDF file      */
{
    struct fgr_record       *fgr_ptr;
    struct fg_line_rec      *fg_line_ptr;
    struct label_index      *line_lbl;
    struct fg_rectangle_rec *fg_rect_ptr;
    struct label_index      *rect_lbl;
    struct fg_polygon_rec   *fg_poly_ptr;
    struct fg_graph_pts     *poly_pts_ptr;
    struct msid_index       *poly_msid;
    struct label_index      *poly_lbl;
    struct scale_index      *poly_scale;
    struct fg_curve_rec     *fg_cur_ptr;
    struct fg_graph_pts     *cur_pts_ptr;
    struct msid_index       *cur_msid;
    struct label_index      *cur_lbl;
```

```
    struct scale_index      *cur_scale;
    struct fg_circle_rec    *fg_cir_ptr;
    struct label_index      *cir_lbl;
    struct fg_arc_rec       *fg_arc_ptr;
    struct label_index      *arc_lbl;
    struct fg_ellipse_rec   *fg_ell_ptr;
    struct label_index      *ell_lbl;
    struct fg_piechrt_rec   *fg_pie_ptr;
    struct pie_msid_index   *pie_msid;
    struct fg_clkmtr_rec    *fg_clk_ptr;
    struct cm_msid_index    *clk_msid;
    struct scale_index      *clk_scale;
    struct fg_bar_rec       *fg_bar_ptr;

    double          fabs();

    float   fx1, fy1, fx2, fy2;
    float   factor_x, factor_y;

    int             fixed_flag;

    short           i, j, w;
    short   color;

    char            temp[2];
    char           *malloc();
    char           *calloc();


    D(printf("START read_fgr\n"));
/*
 *  Set up local world coordinate transformation factors
 */

    factor_x = Dm_Address->display[disp_num].factor_x;
    factor_y = Dm_Address->display[disp_num].factor_y;

/*
 *  Set up local pointer
 */

    fgr_ptr = Fg_rec.graph_rec;

    if (fgr_ptr == NULL) {
        tui_msg(M_YELLOW, "Error %d on graphical record calloc", errno);
        return (-1);
    }

/*
 *  Read in the Display Definition File foreground graphic records
 *  and store them into memory.
 */

    for (i = 0; i < Fg_rec.graph_num; i++) {
        fscanf(ddf_ffp, "%d", &(fgr_ptr->graph_ent));
        fscanf(ddf_ffp, "%hd", &(fgr_ptr->graph_typ));

        fgr_ptr->redraw_flag = NO;

/*
 *      Read in graphical record according to type.
 */

        switch (fgr_ptr->graph_typ) {
```

```
        case LINE:

            fgr_ptr->graph_ptr = malloc(sizeof(struct fg_line_rec));

            if (fgr_ptr->graph_ptr == NULL) {
                tui_msg(M_YELLOW,
                        "Error %d on foreground line record malloc", errno);
                return (-1);
            }

            fg_line_ptr = (struct fg_line_rec *) fgr_ptr->graph_ptr;

            fscanf(ddf_ffp, "%hd", &(fg_line_ptr->line_type));
            fscanf(ddf_ffp, "%f", &(fg_line_ptr->line_wdth));
            fscanf(ddf_ffp, "%f", &(fg_line_ptr->point1_x));
            fscanf(ddf_ffp, "%f", &(fg_line_ptr->point1_y));
            fscanf(ddf_ffp, "%f", &(fg_line_ptr->point2_x));
            fscanf(ddf_ffp, "%f", &(fg_line_ptr->point2_y));

/*
 *          Transform world coordinates.
 */

            fg_line_ptr->points[0].x = (short) (fx1 * factor_x);
            fg_line_ptr->points[0].y = (short) ((100.0 - fy1) * factor_y);
            fg_line_ptr->points[1].x = (short) (fx2 * factor_x);
            fg_line_ptr->points[1].y = (short) ((100.0 - fy2) * factor_y);

/* RLK 10/8/90  May need to transform these coordinates...what are they for? */

            fscanf(ddf_ffp, "%d", &(fg_line_ptr->msid1_x));
            fscanf(ddf_ffp, "%d", &(fg_line_ptr->msid1_y));
            fscanf(ddf_ffp, "%d", &(fg_line_ptr->msid2_x));
            fscanf(ddf_ffp, "%d", &(fg_line_ptr->msid2_y));

            fscanf(ddf_ffp, "%d", &(fg_line_ptr->scale_ind1));
            fscanf(ddf_ffp, "%d", &(fg_line_ptr->scale_ind2));
            fscanf(ddf_ffp, "%d", &(fg_line_ptr->scale_ind3));
            fscanf(ddf_ffp, "%d", &(fg_line_ptr->scale_ind4));
            fscanf(ddf_ffp, "%d", &(fg_line_ptr->ddd_ind));
            fscanf(ddf_ffp, "%d", &(fg_line_ptr->scale_ind));
            fscanf(ddf_ffp, "%d", &(fg_line_ptr->label_num));

            fg_line_ptr->points[0].x = fg_line_ptr->point1_x;
            fg_line_ptr->points[0].y = fg_line_ptr->point1_y;
            fg_line_ptr->points[1].x = fg_line_ptr->point2_x;
            fg_line_ptr->points[1].y = fg_line_ptr->point2_y;

/*
 *          Allocate memory for the label index structure and
 *          read in label indices.
 */

            fg_line_ptr->line_lbl_ptr = (struct label_index *)
                calloc(1, sizeof(struct label_index));

            if (fg_line_ptr->line_lbl_ptr == NULL) {
                tui_msg(M_YELLOW,
                        "Error %d on foreground line label calloc ", errno);
                return (-1);
            }

            line_lbl = fg_line_ptr->line_lbl_ptr;
```

```
        for (j = 0; j < fg_line_ptr->label_num; j++)
            fscanf(ddf_ffp, "%hd", &(line_lbl->label_ind[j]));

        fscanf(ddf_ffp, "%d", &(fg_line_ptr->rot_ind));
        fscanf(ddf_ffp, "%d", &(fg_line_ptr->vis_ind));

        fixed_flag =
            fg_line_ptr->msid1_x +
            fg_line_ptr->msid1_y +
            fg_line_ptr->msid2_x +
            fg_line_ptr->msid2_y +
            fg_line_ptr->scale_ind1 +
            fg_line_ptr->scale_ind2 +
            fg_line_ptr->scale_ind3 +
            fg_line_ptr->scale_ind4 +
            fg_line_ptr->ddd_ind +
            fg_line_ptr->rot_ind +
            fg_line_ptr->vis_ind;

        if (fixed_flag == 0)
            fg_line_ptr->pbi_ind = 1;

        fg_line_ptr->cur_color = -1;

        break;

    case RECTANGLE:

        fgr_ptr->graph_ptr= malloc(sizeof(struct fg_rectangle_rec));

        if (fgr_ptr->graph_ptr == NULL) {
            tui_msg(M_YELLOW,
                    "Error %d on foreground rectangle record malloc",
                        errno);
            return (-1);
        }

        fg_rect_ptr = (struct fg_rectangle_rec *)
                                            fgr_ptr->graph_ptr;

        fscanf(ddf_ffp, "%hd", &(fg_rect_ptr->line_type));
        fscanf(ddf_ffp, "%f", &(fg_rect_ptr->line_wdth));
        fscanf(ddf_ffp, "%hd", &(fg_rect_ptr->pat_type));
        fscanf(ddf_ffp, "%f", &(fg_rect_ptr->pat_sizex));
        fscanf(ddf_ffp, "%f", &(fg_rect_ptr->pat_sizey));

        fscanf(ddf_ffp, "%f", &fg_rect_ptr->ul_x);
        fscanf(ddf_ffp, "%f", &fg_rect_ptr->lr_y);
        fscanf(ddf_ffp, "%f", &fg_rect_ptr->lr_x);
        fscanf(ddf_ffp, "%f", &fg_rect_ptr->ul_y);

/*
 *      Transform world coordinates.
 */

        fg_rect_ptr->rect.x = (short)(fg_rect_ptr->ul_x * factor_x);
        fg_rect_ptr->rect.y = (short)
                        ((100.0 - fg_rect_ptr->ul_y) * factor_y);

        fg_rect_ptr->width = (short) ((fg_rect_ptr->lr_x
                                - fg_rect_ptr->ul_x) * factor_x);
        fg_rect_ptr->height = (short) ((fg_rect_ptr->ul_y
                                - fg_rect_ptr->lr_y) * factor_y);
```

```
/* RLK 10/9/90  Check these coordinates, etc...what are they for? */

                fscanf(ddf_ffp, "%d", &(fg_rect_ptr->msid_ul_x));
                fscanf(ddf_ffp, "%d", &(fg_rect_ptr->msid_ul_y));
                fscanf(ddf_ffp, "%d", &(fg_rect_ptr->msid_lr_x));
                fscanf(ddf_ffp, "%d", &(fg_rect_ptr->msid_lr_y));

                fscanf(ddf_ffp, "%d", &(fg_rect_ptr->scale_ind1));
                fscanf(ddf_ffp, "%d", &(fg_rect_ptr->scale_ind2));
                fscanf(ddf_ffp, "%d", &(fg_rect_ptr->scale_ind3));
                fscanf(ddf_ffp, "%d", &(fg_rect_ptr->scale_ind4));
                fscanf(ddf_ffp, "%d", &(fg_rect_ptr->ddd_ind));
                fscanf(ddf_ffp, "%d", &(fg_rect_ptr->scale_ind));
                fscanf(ddf_ffp, "%d", &(fg_rect_ptr->label_num));

/*
 *              Allocate memory for label index structure and read
 *              in label indices.
 */

                fg_rect_ptr->rect_lbl_ptr = (struct label_index *)
                    calloc(1, sizeof(struct label_index));

                if (fg_rect_ptr->rect_lbl_ptr == NULL) {
                    tui_msg(M_YELLOW, "Error %d on foreground rect label calloc ", errno);
                    return (-1);
                }

                rect_lbl = fg_rect_ptr->rect_lbl_ptr;

                if (fg_rect_ptr->label_num > 0)
                    for (j = 0; j < fg_rect_ptr->label_num; j++)
                        fscanf(ddf_ffp, "%hd", &(rect_lbl->label_ind[j]));

                fscanf(ddf_ffp, "%d", &(fg_rect_ptr->vis_ind));

                fixed_flag =
                    fg_rect_ptr->msid_ul_x +
                    fg_rect_ptr->msid_ul_y +
                    fg_rect_ptr->msid_lr_x +
                    fg_rect_ptr->msid_lr_y +
                    fg_rect_ptr->scale_ind1 +
                    fg_rect_ptr->scale_ind2 +
                    fg_rect_ptr->scale_ind3 +
                    fg_rect_ptr->scale_ind4 +
                    fg_rect_ptr->ddd_ind +
                    fg_rect_ptr->vis_ind;

                if (fixed_flag == 0)
                    fg_rect_ptr->pbi_ind = 1;

                fg_rect_ptr->cur_color = -1;

                break;

            case POLYGON:

                fgr_ptr->graph_ptr = malloc(sizeof(struct fg_polygon_rec));

                if (fgr_ptr->graph_ptr == NULL) {
                    tui_msg(M_YELLOW, "Error %d on foreground polygon record malloc", errn
o);
                    return (-1);
```

```
    }

    fg_poly_ptr = (struct fg_polygon_rec *) fgr_ptr->graph_ptr;

    fscanf(ddf_ffp, "%hd", &(fg_poly_ptr->line_type));
    fscanf(ddf_ffp, "%f", &(fg_poly_ptr->line_wdth));
    fscanf(ddf_ffp, "%hd", &(fg_poly_ptr->pat_type));
    fscanf(ddf_ffp, "%f", &(fg_poly_ptr->pat_sizex));
    fscanf(ddf_ffp, "%f", &(fg_poly_ptr->pat_sizey));
    fscanf(ddf_ffp, "%d", &(fg_poly_ptr->fnmbr_pts));
    fscanf(ddf_ffp, "%d", &(fg_poly_ptr->mnmbr_pts));

/*
 *      Allocate memory for the fix points and
 *      assign local polygon ptr to read fixed
 *      points.
 */

    fg_poly_ptr->poly_pts_ptr = (struct fg_graph_pts *)
        calloc(fg_poly_ptr->fnmbr_pts, sizeof(struct fg_graph_pts));

    if (fg_poly_ptr->poly_pts_ptr == NULL) {
        tui_msg(M_YELLOW, "Error on poly points calloc ");
        return (-1);
    }

    poly_pts_ptr = fg_poly_ptr->poly_pts_ptr;

    for (w = 0; w < fg_poly_ptr->fnmbr_pts; w++) {
        fscanf(ddf_ffp, "%f", &(poly_pts_ptr->point_x));
        fscanf(ddf_ffp, "%f", &(poly_pts_ptr->point_y));

/*
 *      Transform world coordinates
 */

        fg_poly_ptr->points[w].x = (short)
                    (poly_pts_ptr->point_x * factor_x);
        fg_poly_ptr->points[w].y = (short)
                    ((100.0 - poly_pts_ptr->point_y) * factor_y);

        poly_pts_ptr++;
    }

/*
 *      Allocate memory for the msid indices and
 *      assign local polygon ptr to read msid
 *      indices.
 */

    fg_poly_ptr->msid_ind_ptr = (struct msid_index *)
        calloc((fg_poly_ptr->mnmbr_pts * 2), sizeof(struct msid_index));
    if (fg_poly_ptr->msid_ind_ptr == NULL) {
        tui_msg(M_YELLOW, "Error on poly msid points calloc ");
        return (-1);
    }

    poly_msid = fg_poly_ptr->msid_ind_ptr;

    for (j = 0; j < (fg_poly_ptr->mnmbr_pts * 2); j++) {
        fscanf(ddf_ffp, "%d", &(poly_msid->msid_ind));
        fixed_flag += poly_msid->msid_ind;
        poly_msid++;
    }
```

```
/*
 *              Allocate memory for the scale indices and
 *              assign local polygon ptr to read scale
 *              indices.
 */

                fg_poly_ptr->poly_scale_ptr = (struct scale_index *)
                    calloc((fg_poly_ptr->mnmbr_pts * 2), sizeof(struct scale_index));

                if (fg_poly_ptr->poly_scale_ptr == NULL) {
                    tui_msg(M_YELLOW, "Error on poly scale calloc ");
                    return (-1);
                }

                poly_scale = fg_poly_ptr->poly_scale_ptr;

                for (j = 0; j < (fg_poly_ptr->mnmbr_pts * 2); j++) {
                    fscanf(ddf_ffp, "%d", &(poly_scale->scale_ind_num));
                    fixed_flag += poly_scale->scale_ind_num;
                    poly_scale++;
                }

                fscanf(ddf_ffp, "%d", &(fg_poly_ptr->ddd_ind));
                fscanf(ddf_ffp, "%d", &(fg_poly_ptr->scale_ind));
                fscanf(ddf_ffp, "%d", &(fg_poly_ptr->label_num));

/*
 *              Allocate memory for label index structure and read
 *              into memory
 */

                fg_poly_ptr->poly_lbl_ptr = (struct label_index *)
                    calloc(1, sizeof(struct label_index));

                if (fg_poly_ptr->poly_lbl_ptr == NULL) {
                    EXmsg("Error %d on foreground poly label calloc ", errno);
                    return (-1);
                }

                poly_lbl = fg_poly_ptr->poly_lbl_ptr;

                for (j = 0; j < fg_poly_ptr->label_num; j++)
                    fscanf(ddf_ffp, "%hd", &(poly_lbl->label_ind[j]));

                fscanf(ddf_ffp, "%d", &(fg_poly_ptr->rot_ind));
                fscanf(ddf_ffp, "%d", &(fg_poly_ptr->vis_ind));

                fixed_flag += fg_poly_ptr->ddd_ind +
                    fg_poly_ptr->vis_ind +
                    fg_poly_ptr->rot_ind;

                if (fixed_flag == 0)
                    fg_poly_ptr->pbi_ind = 1;

                fg_poly_ptr->cur_color = -1;

                break;

            case CURVE:

                fgr_ptr->graph_ptr =
                    malloc(sizeof(struct fg_curve_rec));
```

```
    if (fgr_ptr->graph_ptr == NULL) {
        tui_msg(M_YELLOW, "Error %d on foreground curvical record malloc", err
no);
        return (-1);
    }

    fg_cur_ptr = (struct fg_curve_rec *) fgr_ptr->graph_ptr;

    fscanf(ddf_ffp, "%hd", &(fg_cur_ptr->line_type));
    fscanf(ddf_ffp, "%f", &(fg_cur_ptr->line_wdth));
    fscanf(ddf_ffp, "%d", &(fg_cur_ptr->fnmbr_pts));
    fscanf(ddf_ffp, "%d", &(fg_cur_ptr->mnmbr_pts));

/*
 *
 *  Allocate memory for the fixed points, msid points,
 *  scale indices, and label indices.
 */

    fg_cur_ptr->cur_pts_ptr = (struct fg_graph_pts *)
        calloc(fg_cur_ptr->fnmbr_pts, sizeof(struct fg_graph_pts));

    if (fg_cur_ptr->cur_pts_ptr == NULL) {
        tui_msg(M_YELLOW, "Error %d on foreground curve record malloc", errno)
;
        return (-1);
    }

    fg_cur_ptr->msid_ind_ptr = (struct msid_index *)
        calloc((fg_cur_ptr->mnmbr_pts * 2),
                                    sizeof(struct msid_index));

    if (fg_cur_ptr->msid_ind_ptr == NULL) {
        tui_msg(M_YELLOW, "Error on curve msid points calloc ");
        return (-1);
    }

    fg_cur_ptr->cur_scale_ptr = (struct scale_index *)
        calloc((fg_cur_ptr->mnmbr_pts * 2),
                        sizeof(struct scale_index));

    if (fg_cur_ptr->cur_scale_ptr == NULL) {
        tui_msg(M_YELLOW, "Error on curve scale calloc ");
        return (-1);
    }

    fg_cur_ptr->cur_lbl_ptr = (struct label_index *)
                        calloc(1, sizeof(struct label_index));

    if (fg_cur_ptr->cur_lbl_ptr == NULL) {
        tui_msg(M_YELLOW, "Error %d on foreground curve label calloc ", errno)
;
        return (-1);
    }

/*
 *
 *  Read fixed points
 */

    cur_pts_ptr = fg_cur_ptr->cur_pts_ptr;

    for (w = 0; w < fg_cur_ptr->fnmbr_pts; w++) {
        fscanf(ddf_ffp, "%f", &(cur_pts_ptr->point_x));
        fscanf(ddf_ffp, "%f", &(cur_pts_ptr->point_y));
```

```
/*
 *              Transform world coordinates
 */

                fg_cur_ptr->points[w].x = (short)
                                (cur_pts_ptr->point_x * factor_x);
                fg_cur_ptr->points[w].y = (short)
                            ((100.0 - cur_pts_ptr->point_y) * factor_y);

                cur_pts_ptr++;
            }

/*
 *          Read msid indices
 */

            cur_msid = fg_cur_ptr->msid_ind_ptr;

            for (j = 0; j < (fg_cur_ptr->mnmbr_pts * 2); j++) {
                fscanf(ddf_ffp, "%d", &(cur_msid->msid_ind));
                fixed_flag += cur_msid->msid_ind;
                cur_msid++;
            }

/*
 *          Read scale indices
 */

            cur_scale = fg_cur_ptr->cur_scale_ptr;

            for (j = 0; j < (fg_cur_ptr->mnmbr_pts * 2); j++) {
                fscanf(ddf_ffp, "%d", &(cur_scale->scale_ind_num));
                fixed_flag += cur_scale->scale_ind_num;
                cur_scale++;
            }

            fscanf(ddf_ffp, "%d", &(fg_cur_ptr->ddd_ind));
            fscanf(ddf_ffp, "%d", &(fg_cur_ptr->scale_ind));
            fscanf(ddf_ffp, "%d", &(fg_cur_ptr->label_num));

/*
 *          Read label indices
 */

            cur_lbl = fg_cur_ptr->cur_lbl_ptr;

            for (j = 0; j < fg_cur_ptr->label_num; j++)
                fscanf(ddf_ffp, "%hd", &(cur_lbl->label_ind[j]));

            fscanf(ddf_ffp, "%d", &(fg_cur_ptr->vis_ind));

            fixed_flag += fg_cur_ptr->ddd_ind +
                fg_cur_ptr->vis_ind;

            if (fixed_flag == 0)
                fg_cur_ptr->pbi_ind = 1;

            fg_cur_ptr->cur_color = -1;

            break;

        case CIRCLE:

            fgr_ptr->graph_ptr = malloc(sizeof(struct fg_circle_rec));
```

```
if (fgr_ptr->graph_ptr == NULL) {
    EXmsg("Error %d on foreground circle record malloc", errno);
    return (-1);
}

fg_cir_ptr = (struct fg_circle_rec *) fgr_ptr->graph_ptr;

fscanf(ddf_ffp, "%hd", &(fg_cir_ptr->line_type));
fscanf(ddf_ffp, "%f", &(fg_cir_ptr->line_wdth));
fscanf(ddf_ffp, "%hd", &(fg_cir_ptr->pat_type));
fscanf(ddf_ffp, "%f", &(fg_cir_ptr->pat_sizex));
fscanf(ddf_ffp, "%f", &(fg_cir_ptr->pat_sizey));
fscanf(ddf_ffp, "%f", &(fg_cir_ptr->center_x));
fscanf(ddf_ffp, "%f", &(fg_cir_ptr->center_y));
fscanf(ddf_ffp, "%f", &(fg_cir_ptr->radius));
fscanf(ddf_ffp, "%d", &(fg_cir_ptr->msid_cen_x));
fscanf(ddf_ffp, "%d", &(fg_cir_ptr->msid_cen_y));
fscanf(ddf_ffp, "%d", &(fg_cir_ptr->msid_radius));
fscanf(ddf_ffp, "%d", &(fg_cir_ptr->scale_ind1));
fscanf(ddf_ffp, "%d", &(fg_cir_ptr->scale_ind2));
fscanf(ddf_ffp, "%d", &(fg_cir_ptr->scale_ind3));
fscanf(ddf_ffp, "%d", &(fg_cir_ptr->ddd_ind));
fscanf(ddf_ffp, "%d", &(fg_cir_ptr->scale_ind));
fscanf(ddf_ffp, "%d", &(fg_cir_ptr->label_num));

fg_cir_ptr->cur_rad = fg_cir_ptr->radius * factor_x;

fg_cir_ptr->bb.x = (short) (fg_cir_ptr->center_x * factor_x
                            - fg_cir_ptr->cur_rad);
fg_cir_ptr->bb.y = (short) ((100.0 - fg_cir_ptr->center_y)
                    * factor_y - fg_cir_ptr->cur_rad);

fg_cir_ptr->cir_lbl_ptr = (struct label_index *)
                    calloc(1, sizeof(struct label_index));

if (fg_cir_ptr->cir_lbl_ptr == NULL) {
    tui_msg(M_YELLOW, "Error %d on foreground cir label calloc ", errno);
    return (-1);
}

cir_lbl = fg_cir_ptr->cir_lbl_ptr;

for (j = 0; j < fg_cir_ptr->label_num; j++)
    fscanf(ddf_ffp, "%hd", &(cir_lbl->label_ind[j]));

fscanf(ddf_ffp, "%d", &(fg_cir_ptr->vis_ind));

fixed_flag =
    fg_cir_ptr->msid_cen_x +
    fg_cir_ptr->msid_cen_y +
    fg_cir_ptr->msid_radius +
    fg_cir_ptr->scale_ind1 +
    fg_cir_ptr->scale_ind2 +
    fg_cir_ptr->scale_ind3 +
    fg_cir_ptr->ddd_ind +
    fg_cir_ptr->scale_ind +
    fg_cir_ptr->vis_ind;

if (fixed_flag == 0)
    fg_cir_ptr->pbi_ind = 1;

fg_cir_ptr->cur_color = -1;
```

```
            break;

        case ARC:

            fgr_ptr->graph_ptr = malloc(sizeof(struct fg_arc_rec));

            if (fgr_ptr->graph_ptr == NULL) {
                tui_msg(M_YELLOW, "Error %d on foreground arc record malloc", errno);
                return (-1);
            }

            fg_arc_ptr = (struct fg_arc_rec *) fgr_ptr->graph_ptr;

            fscanf(ddf_ffp, "%hd", &(fg_arc_ptr->line_type));
            fscanf(ddf_ffp, "%f", &(fg_arc_ptr->line_wdth));
            fscanf(ddf_ffp, "%hd", &(fg_arc_ptr->pat_type));
            fscanf(ddf_ffp, "%f", &(fg_arc_ptr->pat_sizex));
            fscanf(ddf_ffp, "%f", &(fg_arc_ptr->pat_sizey));
            fscanf(ddf_ffp, "%f", &(fg_arc_ptr->center_x));
            fscanf(ddf_ffp, "%f", &(fg_arc_ptr->center_y));
            fscanf(ddf_ffp, "%lf", &(fg_arc_ptr->angle1));
            fscanf(ddf_ffp, "%lf", &(fg_arc_ptr->angle2));
            fscanf(ddf_ffp, "%f", &(fg_arc_ptr->maj_axis));
            fscanf(ddf_ffp, "%f", &(fg_arc_ptr->min_axis));
            fscanf(ddf_ffp, "%d", &(fg_arc_ptr->msid_cen_x));
            fscanf(ddf_ffp, "%d", &(fg_arc_ptr->msid_cen_y));
            fscanf(ddf_ffp, "%d", &(fg_arc_ptr->msid_ang1));
            fscanf(ddf_ffp, "%d", &(fg_arc_ptr->msid_ang2));
            fscanf(ddf_ffp, "%d", &(fg_arc_ptr->msid_maj));
            fscanf(ddf_ffp, "%d", &(fg_arc_ptr->msid_min));
            fscanf(ddf_ffp, "%d", &(fg_arc_ptr->scale_ind1));
            fscanf(ddf_ffp, "%d", &(fg_arc_ptr->scale_ind2));
            fscanf(ddf_ffp, "%d", &(fg_arc_ptr->scale_ind3));
            fscanf(ddf_ffp, "%d", &(fg_arc_ptr->scale_ind4));
            fscanf(ddf_ffp, "%d", &(fg_arc_ptr->scale_ind5));
            fscanf(ddf_ffp, "%d", &(fg_arc_ptr->scale_ind6));
            fscanf(ddf_ffp, "%d", &(fg_arc_ptr->ddd_ind));
            fscanf(ddf_ffp, "%d", &(fg_arc_ptr->scale_ind));
            fscanf(ddf_ffp, "%d", &(fg_arc_ptr->label_num));

            fg_arc_ptr->smajor = (short) (fg_arc_ptr->maj_axis * factor_x);
            fg_arc_ptr->sminor = (short)
                                 ((100.0 - fg_arc_ptr->min_axis) * factor_y);
            fg_arc_ptr->center.x = (short)(fg_arc_ptr->center_x * factor_x);
            fg_arc_ptr->center.y = (short)
                                 ((100.0 - fg_arc_ptr->center_y) * factor_y);
            fg_arc_ptr->cur_ang1 = fg_arc_ptr->angle1;
            fg_arc_ptr->cur_ang2 = fg_arc_ptr->angle2;

            if (fg_arc_ptr->label_num > 0) {
                fg_arc_ptr->arc_lbl_ptr = (struct label_index *)
                                 calloc(1, sizeof(struct label_index));

                if (fg_arc_ptr->arc_lbl_ptr == NULL) {
                    tui_msg(M_YELLOW, "Error %d on foreground arc label calloc ", errn
o);
                    return (-1);
                }

                arc_lbl = fg_arc_ptr->arc_lbl_ptr;

                for (j = 0; j < fg_arc_ptr->label_num; j++)
                    fscanf(ddf_ffp, "%hd", &(arc_lbl->label_ind[j]));
            }
```

```
            fscanf(ddf_ffp, "%d", &(fg_arc_ptr->rot_ind));
            fscanf(ddf_ffp, "%d", &(fg_arc_ptr->vis_ind));

            fixed_flag =
                fg_arc_ptr->msid_cen_x +
                fg_arc_ptr->msid_cen_y +
                fg_arc_ptr->msid_ang1 +
                fg_arc_ptr->msid_ang2 +
                fg_arc_ptr->msid_maj +
                fg_arc_ptr->msid_min +
                fg_arc_ptr->scale_ind1 +
                fg_arc_ptr->scale_ind2 +
                fg_arc_ptr->scale_ind3 +
                fg_arc_ptr->scale_ind4 +
                fg_arc_ptr->scale_ind5 +
                fg_arc_ptr->scale_ind6 +
                fg_arc_ptr->ddd_ind +
                fg_arc_ptr->scale_ind +
                fg_arc_ptr->rot_ind +
                fg_arc_ptr->vis_ind;

            if (fixed_flag == 0)
                fg_arc_ptr->pbi_ind = 1;

            fg_arc_ptr->cur_color = -1;

            break;

        case ELLIPSE:

            fgr_ptr->graph_ptr = malloc(sizeof(struct fg_ellipse_rec));

            if (fgr_ptr->graph_ptr == NULL) {
                tui_msg(M_YELLOW, "Error %d on foreground ellipse record malloc", errn
o);

                return (-1);
            }

            fg_ell_ptr = (struct fg_ellipse_rec *) fgr_ptr->graph_ptr;

            fscanf(ddf_ffp, "%hd", &(fg_ell_ptr->line_type));
            fscanf(ddf_ffp, "%f", &(fg_ell_ptr->line_wdth));
            fscanf(ddf_ffp, "%hd", &(fg_ell_ptr->pat_type));
            fscanf(ddf_ffp, "%f", &(fg_ell_ptr->pat_sizex));
            fscanf(ddf_ffp, "%f", &(fg_ell_ptr->pat_sizey));
            fscanf(ddf_ffp, "%f", &(fg_ell_ptr->center_x));
            fscanf(ddf_ffp, "%f", &(fg_ell_ptr->center_y));
            fscanf(ddf_ffp, "%f", &(fg_ell_ptr->maj_axis));
            fscanf(ddf_ffp, "%f", &(fg_ell_ptr->min_axis));
            fscanf(ddf_ffp, "%d", &(fg_ell_ptr->msid_cen_x));
            fscanf(ddf_ffp, "%d", &(fg_ell_ptr->msid_cen_y));
            fscanf(ddf_ffp, "%d", &(fg_ell_ptr->msid_len));
            fscanf(ddf_ffp, "%d", &(fg_ell_ptr->msid_hgh));
            fscanf(ddf_ffp, "%d", &(fg_ell_ptr->scale_ind1));
            fscanf(ddf_ffp, "%d", &(fg_ell_ptr->scale_ind2));
            fscanf(ddf_ffp, "%d", &(fg_ell_ptr->scale_ind3));
            fscanf(ddf_ffp, "%d", &(fg_ell_ptr->scale_ind4));
            fscanf(ddf_ffp, "%d", &(fg_ell_ptr->ddd_ind));
            fscanf(ddf_ffp, "%d", &(fg_ell_ptr->scale_ind));
            fscanf(ddf_ffp, "%d", &(fg_ell_ptr->label_num));

            fg_ell_ptr->center.x = (short)(fg_ell_ptr->center_x * factor_x);
            fg_ell_ptr->center.y = (short)
```

```
                                ((100.0 - fg_ell_ptr->center_y) * factor_y);
                    fg_ell_ptr->smajor = (short)(fg_ell_ptr->maj_axis * factor_x);
                    fg_ell_ptr->sminor = (short)
                                ((100.0 - fg_ell_ptr->min_axis) * factor_y);

                    fg_ell_ptr->ell_lbl_ptr = (struct label_index *)
                                        calloc(1, sizeof(struct label_index));

                    if (fg_ell_ptr->ell_lbl_ptr == NULL) {
                        tui_msg(M_YELLOW, "Error %d on foreground ellipse label calloc ", errn
o);
                        return (-1);
                    }

                    ell_lbl = fg_ell_ptr->ell_lbl_ptr;

                    for (j = 0; j < fg_ell_ptr->label_num; j++)
                        fscanf(ddf_ffp, "%hd", &(ell_lbl->label_ind[j]));

                    fscanf(ddf_ffp, "%d", &(fg_ell_ptr->vis_ind));

                    fixed_flag =
                        fg_ell_ptr->msid_cen_x +
                        fg_ell_ptr->msid_cen_y +
                        fg_ell_ptr->msid_len +
                        fg_ell_ptr->msid_hgh +
                        fg_ell_ptr->scale_ind1 +
                        fg_ell_ptr->scale_ind2 +
                        fg_ell_ptr->scale_ind3 +
                        fg_ell_ptr->scale_ind4 +
                        fg_ell_ptr->ddd_ind +
                        fg_ell_ptr->vis_ind;

                    if (fixed_flag == 0)
                        fg_ell_ptr->pbi_ind = 1;

                    fg_ell_ptr->cur_color = -1;

                    break;

                case PIE_CHART:

    /* RLK 10/9/90  Assuming no pie charts for now...code not corrected */

                    fgr_ptr->graph_ptr = malloc(sizeof(struct fg_piechrt_rec));

                    if (fgr_ptr->graph_ptr == NULL) {
                        tui_msg(M_YELLOW, "Error %d on foreground pie chart record malloc", er
rno);
                        return (-1);
                    }

                    fg_pie_ptr = (struct fg_piechrt_rec *)
                        fgr_ptr->graph_ptr;

                    fscanf(ddf_ffp, "%hd", &color);
                    fg_pie_ptr->def_col = (long) Pixels[color];
                    fscanf(ddf_ffp, "%hd", &(fg_pie_ptr->pat_type));
                    fscanf(ddf_ffp, "%f", &(fg_pie_ptr->pat_sizex));
                    fscanf(ddf_ffp, "%f", &(fg_pie_ptr->pat_sizey));
                    fscanf(ddf_ffp, "%f", &(fg_pie_ptr->center_x));
                    fscanf(ddf_ffp, "%f", &(fg_pie_ptr->center_y));
                    fscanf(ddf_ffp, "%f", &(fg_pie_ptr->radius));
                    fscanf(ddf_ffp, "%hd", &(fg_pie_ptr->sum_flag));
```

```
                fscanf(ddf_ffp, "%lf", &(fg_pie_ptr->sum_pie));
                fscanf(ddf_ffp, "%d", &(fg_pie_ptr->num_msid));

                fg_pie_ptr->pie_msid_ptr = (struct pie_msid_index *)
                    calloc(fg_pie_ptr->num_msid, sizeof(struct pie_msid_index));
                if (fg_pie_ptr->pie_msid_ptr == NULL) {
                    return (-1);
                }
                pie_msid = fg_pie_ptr->pie_msid_ptr;

                for (j = 0; j < fg_pie_ptr->num_msid; j++) {
                    fscanf(ddf_ffp, "%d", &(pie_msid->msid_ind));
                    pie_msid->cur_color = -1;
                    pie_msid++;
                }
                break;

            case CLOCK_METER:

/* RLK 10/9/90  Assuming no clock meters for now...code not corrected */

                fgr_ptr->graph_ptr = malloc(sizeof(struct fg_clkmtr_rec));

                if (fgr_ptr->graph_ptr == NULL) {
                    tui_msg(M_YELLOW, "Error %d on foreground clock meter record malloc",
errno);
                    return (-1);
                }

                fg_clk_ptr = (struct fg_clkmtr_rec *) fgr_ptr->graph_ptr;

                fscanf(ddf_ffp, "%hd", &(fg_clk_ptr->line_type));
                fscanf(ddf_ffp, "%f", &(fg_clk_ptr->line_wdth));
                fscanf(ddf_ffp, "%d", &color);
                fg_clk_ptr->clkmtr_col = Pixels[color];
                fscanf(ddf_ffp, "%hd", &(fg_clk_ptr->pat_type));
                fscanf(ddf_ffp, "%f", &(fg_clk_ptr->pat_sizex));
                fscanf(ddf_ffp, "%f", &(fg_clk_ptr->pat_sizey));
                fscanf(ddf_ffp, "%f", &(fg_clk_ptr->center_x));
                fscanf(ddf_ffp, "%f", &(fg_clk_ptr->center_y));
                fscanf(ddf_ffp, "%f", &(fg_clk_ptr->radius));
                fscanf(ddf_ffp, "%lf", &(fg_clk_ptr->angle_1));
                fscanf(ddf_ffp, "%lf", &(fg_clk_ptr->angle_2));
                fg_clk_ptr->angle_diff = fabs(fg_clk_ptr->angle_2 -
                                              fg_clk_ptr->angle_1);
                fscanf(ddf_ffp, "%d", &(fg_clk_ptr->num_msid));

                fg_clk_ptr->clk_msid_ptr = (struct cm_msid_index *)
                    calloc(fg_clk_ptr->num_msid, sizeof(struct cm_msid_index));

                if (fg_clk_ptr->clk_msid_ptr == NULL) {
                    return (-1);
                }

                clk_msid = fg_clk_ptr->clk_msid_ptr;

                for (j = 0; j < fg_clk_ptr->num_msid; j++) {
                    fscanf(ddf_ffp, "%d", &(clk_msid->msid_ind));
                    clk_msid->cur_color = -1;
                    clk_msid++;
                }

                for (j = 0; j < fg_clk_ptr->num_msid; j++)
                    fscanf(ddf_ffp, "%hd", &(fg_clk_ptr->hand_type[j]));
```

```
                fg_clk_ptr->clk_scale_ptr = (struct scale_index *)
                    calloc(fg_clk_ptr->num_msid, sizeof(struct scale_index));

                if (fg_clk_ptr->clk_scale_ptr == NULL) {
                    return (-1);
                }

                clk_scale = fg_clk_ptr->clk_scale_ptr;

                for (j = 0; j < fg_clk_ptr->num_msid; j++) {
                    fscanf(ddf_ffp, "%d", &(clk_scale->scale_ind_num));
                    clk_scale++;
                }

                fg_clk_ptr->init_draw = YES;

                break;

            case BAR_CHART:

/* RLK 10/9/90  Assuming no bar charts for now...code not corrected */

                fgr_ptr->graph_ptr = malloc(sizeof(struct fg_bar_rec));

                if (fgr_ptr->graph_ptr == NULL) {
                    tui_msg(M_YELLOW, "Error %d on foreground bar chart record malloc", er
rno);
                    return (-1);
                }

                fg_bar_ptr = (struct fg_bar_rec *) fgr_ptr->graph_ptr;

                fscanf(ddf_ffp, "%hd", &(fg_bar_ptr->line_type));
                fscanf(ddf_ffp, "%f", &(fg_bar_ptr->line_wdth));
                fscanf(ddf_ffp, "%s", temp);
                fg_bar_ptr->direction = temp[0];
                fscanf(ddf_ffp, "%hd", &(fg_bar_ptr->pat_type));
                fscanf(ddf_ffp, "%f", &(fg_bar_ptr->pat_sizex));
                fscanf(ddf_ffp, "%f", &(fg_bar_ptr->pat_sizey));
                fscanf(ddf_ffp, "%f", &(fg_bar_ptr->ul_x));
                fscanf(ddf_ffp, "%f", &(fg_bar_ptr->lr_y));
                fscanf(ddf_ffp, "%f", &(fg_bar_ptr->lr_x));
                fscanf(ddf_ffp, "%f", &(fg_bar_ptr->ul_y));
                fscanf(ddf_ffp, "%d", &(fg_bar_ptr->msid_indx));
                fscanf(ddf_ffp, "%d", &(fg_bar_ptr->scale_indx));

                fg_bar_ptr->cur_color = -1;

                break;

            default:
                break;
        }
        fgr_ptr++;
    }

    D(printf("END read_fgr\n"));
    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: read_files.c
 *
 *  The function searches a directory for either limit or plot files and gen-
 *  erates a list which is later presented to the user.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  D. Rice      - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <constants.h>
#include <disp.h>
#include <stdio.h>
#include <fcntl.h>
#include <wex/EXmsg.h>


extern struct dm_shmemory   *Dm_Address;
extern struct limit_file    *First_lim_ptr;
extern struct file_info     *Disp_Info;             /* pointer to file information     */

extern char                 Disp_Path[DNAME_LEN];   /* display path name               */

extern int                  errno;                  /* system return error             */

extern short                Disp_Num;               /* display manager number          */


int read_files ( limit_list )

    int                 limit_list;
{
    int                 num_disps = 0;

    struct limit_file   *temp_ptr;
    struct file_info    *d_info_ptr;                /* pointer to file information     */

    char                *ptr,
                        file_name[DNAME_LEN + 4],   /* display name                    */
                        disp_name[DNAME_LEN + 4],   /* display name with path          */
                        name[DNAME_LEN];            /* disp name w/path not extension  */

    FILE                *fp, *fp1, *fopen ( );      /* display file pointers           */

    int                 num_files = 0,
                        i, j,                       /* loop indices                    */
                        length;                     /* file name length                */

    char                *calloc ( ),                /* space allocation                */
                        str[80],                    /* contains system command         */
                        temp_file[30],              /* temporary file name             */
                        match;                      /* tells if plot is active or not  */
```

```
    D(printf("START read_files\n"));
/*
 *  Display wait cursor.
 */

    tui_start_wait ( );

/*
 *  Build the system commands to read all the plot files in the display
 *  directory into a temporary file 'tdispX.dat'. ( ls [path] > tdisp.dat )
 */

    sprintf ( temp_file, "/user/display/tdisp%hd.dat", Disp_Num );
    sprintf ( str, "ls %s*.%s > %s", Disp_Path, ( limit_list == ON ) ? "lmt" : "plt",
              temp_file );

    if ( system ( str ) ) {
        tui_msg ( M_YELLOW, "Error on reading file directory" );
        tui_stop_wait ( );
        return ( -1 );
    }
    if ( ( fp = fopen ( temp_file, "r" ) ) == NULL ) {
        tui_msg ( M_YELLOW, "Error %d on reading file directory file", errno );
        tui_stop_wait ( );
        return ( -1 );
    }

/*
 *  Read the directory file for an initial count of the number of
 *  files in the directory.
 */

    while ( ( fscanf ( fp, "%s", file_name ) ) != EOF )
        num_files++;

    rewind ( fp );

/*
 *  Allocate space for the file name and active flag
 */

    Disp_Info = (struct file_info *)calloc ( num_files, sizeof ( struct file_info ) );
    if ( Disp_Info == NULL ) {
        tui_msg ( M_YELLOW, "Error %d on allocation of file info structure", errno );
        tui_stop_wait ( );
        return ( -1 );
    }
    d_info_ptr = Disp_Info;

/*
 *  Process each filename.
 */

    while ( ( fscanf ( fp, "%s", file_name ) ) != EOF ) {
        strcpy ( disp_name, file_name );

/*
 *      Extract just the filename (less path).
 */

        ptr = file_name + strlen ( file_name ) - 1;
        while ( ptr > file_name && *ptr != '/' )
            ptr--;
        ptr++;
```

```
        if ( ( fp1 = fopen ( disp_name, "r" ) ) == NULL ) {
            tui_msg ( M_YELLOW, "Error %d on display <%s> open", errno, disp_name );
            free ( (char *)Disp_Info );
            tui_stop_wait ( );
            return ( -1 );
        }

/*
 *      Read in the name field and blank the extension '.plt' and add in the ':'
 *      in column 8. Also, check the active plot table and if the plot is active
 *      store ACTIVE in memory, else store INACTIVE.
 */

        length = strlen ( ptr );
        for ( j = 8; j > length - 5; j-- )
            *(ptr + j) = ' ';

        strncpy ( d_info_ptr->name,     ptr,     8 );
        strncpy ( &d_info_ptr->name[8], " : \0", 4 );
        d_info_ptr->inverse_flag = NO;

        i = 0;
        match = NO;
        length = strlen ( disp_name );
        strncpy ( name, disp_name, length - 4 );
        name[length - 4] = '\0';

/*
 *      If processing for plots, then look in plot active table for the status
 */

        if ( limit_list != YES ) {
            while ( ( i < MAX_PLOTS ) && ( match == NO ) ) {
                if ( ( strcmp ( Dm_Address->plots.act_plots[i], name ) ) == 0 ) {
                    match = YES;
                    strncpy ( &d_info_ptr->act_flag[0], "ACTIVE  \0", 9 );
                }
                i++;
            }

/*
 *      Else if processing for limits, then look in the limit link list for the
 *      status.
 */

        } else {
            temp_ptr = First_lim_ptr;
            while ( match == NO && temp_ptr != NULL ) {
                if ( ( strcmp ( temp_ptr->file_name, name ) ) == 0 ) {
                    match = YES;
                    strncpy ( &d_info_ptr->act_flag[0], "ACTIVE  \0", 9 );
                }
                temp_ptr = temp_ptr->next_ptr;
            }
        }

        if ( match == NO )
            strncpy ( &d_info_ptr->act_flag[0], "INACTIVE\0", 9 );

        d_info_ptr++;
        num_disps++;
        fclose ( fp1 );
    }
```

```
    /*
     *  Close the temp file and remove it from system.
     */

        fclose ( fp );
        strncpy ( str, "rm \0", 4 );
        strcat ( str, temp_file );
        if ( system ( str ) )
            tui_msg ( M_YELLOW, "Error on removing temporary file" );

        tui_stop_wait ( );

        D(printf("END read_files\n"));
        return ( num_disps );
    }
```

```
/*****************************************************************************
 * MODULE NAME: read_ovls.c
 *
 *  This function reads the plot data files or overlay files in a directory
 *  and builds a list which makes it easier for the user to specify a source
 *  file.
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <fcntl.h>
#include <constants.h>
#include <disp.h>
#include <pf_key.h>
#include <wex/EXmsg.h>


extern struct pfkey_defs    Current_Com;            /* Current command.              */

extern char                 Disp_Path[DNAME_LEN];   /* Display path name.            */

extern int                  errno;                  /* System return error.          */

extern short                Disp_Num;               /* Display manager number.       */


int read_ovls ( list_file )

    char                    ***list_file;           /* Pointer to the list of overlays. */
{
    FILE                    *fp,
                            *fopen ( );             /* Display file pointers.        */

    char                    *ptr,
                            *ptr1,
                            **list,
                            file_name[DNAME_LEN+4], /* Display name.                 */
                            str      [80],          /* Contains system command.      */
                            temp_file[30],          /* Temporary file name.          */
                            *malloc();

    int                     num_files = 0;

    D(printf("START read_ovls\n"));
/*
 *  Display wait cursor.
 */

    tui_start_wait ( );

/*
 *  Build the system commands to read all overlay files into a temporary file. If the
 *  system call fails, generate an error and return.
 */

    sprintf ( temp_file, "/user/display/tdisp%hd.dat", Disp_Num );
```

```
    sprintf ( str, "ls %s*.%s > %s", Disp_Path,
              ( Current_Com.func_no == SAVE_OVRLAY ) ? "pdt" : "ovr", temp_file );

    if ( system ( str ) ) {
        tui_msg ( M_YELLOW, "Error on reading display directory or no overlay files" );
        tui_stop_wait ( );
        return ( -1 );
    }

/*
 *  Open the file containing the directory information. If this fails, generate an error
 *  and return.
 */

    if ( ( fp = fopen ( temp_file, "r" ) ) == NULL ) {
        tui_msg ( M_YELLOW, "Error %d on reading display directory file", errno );
        tui_stop_wait ( );
        return ( -1 );
    }

/*
 *  Read the display directory file for an initial count of the number of
 *  displays in the directory.
 */

    while ( fscanf ( fp, "%s", file_name ) != EOF )
        num_files++;

    rewind ( fp );

/*
 *  Allocate space for list of file names. If this fails, generate a warning and exit.
 */

    if ( ( list = (char **)malloc ( num_files * sizeof ( char * ) ) ) == NULL ) {
        tui_msg ( M_YELLOW, "Error %d on allocation of list", errno );
        tui_stop_wait ( );
        return ( -1 );
    }

/*
 *  Process each filename. Extract each name and find the actual filename (less the
 *  path). Change the name not to include the extension.
 */

    num_files = 0;
    while ( fscanf ( fp, "%s", file_name ) != EOF ) {
        ptr = file_name + strlen ( file_name ) - 1;
        while ( ptr > file_name && *ptr != '/' )
            ptr--;
        ptr++;
        if ( ptr1 = index ( ptr, '.' ) )
            *ptr1 = '\0';

/*
 *      Allocate memory for the new name and add to the list. MDC - add error processing.
 */

        *(list + num_files) = malloc ( strlen ( ptr ) );
        strcpy ( *(list + num_files), ptr );
        num_files++;
    }

/*
```

```
     *  Close the temporary file and remove it from system.
     */

        fclose ( fp );

        strcpy ( str, "rm "     );
        strcat ( str, temp_file );
        if ( system ( str ) )
            tui_msg ( M_YELLOW, "Error on removing temporary file" );

    /*
     *  Remove the wait cursor.
     */

        tui_stop_wait ( );

    /*
     *  Return the number of files and the list.
     */

        *list_file = list;

        D(printf("END read_ovls\n"));
        return ( num_files );
    }
```

```
/*****************************************************************************
 * MODULE NAME: read_pbi.c
 *
 *  This routine reads the PBI records from the DDF foreground
 *  file.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  R. Romeo        - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <X11/Xlib.h>
#include <fcntl.h>
#include <ctype.h>
#include <sys/types.h>
#include <constants.h>
#include <DDdisp.h>
#include <disp.h>
#include <wex/FCpbi.h>
#include <wex/EXmsg.h>


extern PBI_ENTRY            *Pbi_Ptr;          /* pbi emulation interface ptr      */
extern PBI_TABLE            *Pbi_Table;        /* Pbi Emulation interface ptr      */
extern struct pbi_def       *Pbi_Def;          /* pbi definition table pointer     */

extern struct dm_shmemory   *Dm_Address;       /*  structure pointer               */
extern short                Pbi_Num,           /* Number of Pbi's for display      */
                            Disp_Num;          /* display number to display info   */

extern int      errno;                         /* return error value               */


int read_pbi ( )
{
    struct pbi_def *pbi_def_ptr;               /* pbi defintion pointer            */
    struct pbi_msid_rec *pbi_nxt_msid;         /* pbi defintion pointer            */

    FILE           *ddf_ffp, *fopen ( );

    PBI_ENTRY      *pbi_ptr;                   /* Pbi Emulation interface ptr      */

    unsigned        size;                      /* size for memory allocation       */

    int             i,                         /* loop count variable              */
                    j,                         /* loop count variable              */
                    k,                         /* loop count variable              */
                    m,                         /* loop count variable              */
                    mult_entry,                /* Multilevel text entry number     */
                    pbi_entry_num,             /* Pbi Entry number input           */
                    num_values,                /* Number of multilevel values      */
                    prev_group_num,            /* Prev group number in counting grps */
                    version;                   /* version of the builder file read */
```

```
    char            *pbi_nxt,                   /* local pointer for read loop      */
                    *malloc ( ),                /* get malloc as a pointer          */
                    *calloc ( ),
                    fg_file_name[50],           /* local file path file name        */
                    temp_active_flag,           /* local active flag read in        */
                    access_code;                /* access restriction code          */

    short           grph_indx,                  /* dummy for grph_indx not used     */
                    grph_color,                 /* dummy for graph color not        */
                    inter_type,                 /* type of interactive input object */
                    label_ndx,                  /* label index for reading labels   */
                    labels;                     /* label index for reading labels   */

    long            tab_num,                    /* number of tabular entries        */
                    entry_num,                  /* number of msid entries           */
                    limit_num,                  /* number of limit entries          */
                    pbi_num,                    /* number of pbi entries            */
                    icon_num,                   /* number of icon entries           */
                    tmplt_num,                  /* number of template entries       */
                    mltxt_num,                  /* number of template entries       */
                    label_num,                  /* number of template entries       */
                    scale_num,                  /* number of template entries       */
                    ddd_num,                    /* number of template entries       */
                    inter_num,                  /* number of interact. input objects */
                    htab_num,                   /* number of template entries       */
                    ppl_num;                    /* number of ppl entries            */

    D(printf("START read_pbi\n"));
/*
 *  Build the Display Definiton file name using the display name and the
 *  extension.
 */

    strcpy ( fg_file_name, Dm_Address->display[Disp_Num].display_name );
    strcat ( fg_file_name, ".fg" );

/*
 *  Read the directory of display files into a file.
 */

    ddf_ffp = fopen ( fg_file_name, "r" );
    if ( ddf_ffp == NULL ) {
        Pbi_Num = 0;
        tui_msg ( M_YELLOW, " Error %d on reading DDF foreground file", errno );
        return ( -1 );
    }

/*
 *  Read in total number of entries.
 */

    fscanf ( ddf_ffp, "%d", &version );
    if ( version > VERSION ) {
        fclose ( ddf_ffp );
        return ( -1 );
    }

    if ( version > 2 ) {
        fscanf ( ddf_ffp, "%*68c" );
        fscanf ( ddf_ffp, "%d", &tab_num );
        fscanf ( ddf_ffp, "%d", &entry_num );
        fscanf ( ddf_ffp, "%d", &pbi_num );
        fscanf ( ddf_ffp, "%d", &icon_num );
```

```
        fscanf ( ddf_ffp, "%d", &tmplt_num );
        fscanf ( ddf_ffp, "%d", &mltxt_num );
        fscanf ( ddf_ffp, "%d", &limit_num );
        fscanf ( ddf_ffp, "%d", &label_num );
        fscanf ( ddf_ffp, "%d", &scale_num );
        fscanf ( ddf_ffp, "%d", &ddd_num );
        fscanf ( ddf_ffp, "%d", &inter_num );
        fscanf ( ddf_ffp, "%d", &htab_num );
        fscanf ( ddf_ffp, "%d", &ppl_num );
        fscanf ( ddf_ffp, "%hd ", &access_code );
    } else
        pbi_num = 0;

    Pbi_Num = pbi_num;

/*
 *  Check for pbi foreground records.
 *  If there are none then return, otherwise allocate the PBI_TABLE entry
 */

    if ( pbi_num == 0 ) {
        fclose ( ddf_ffp );
        return ( -1 );
    } else {
        Pbi_Table = ( PBI_TABLE  * ) malloc ( sizeof ( PBI_TABLE ) );

        if ( Pbi_Table == NULL ) {
            Pbi_Num = 0;
            tui_msg ( M_YELLOW, "Error in allocation of PBI table" );
            return ( -1 );
        }
        prev_group_num = -1;
        Pbi_Table->group_count = 0;
    }

/*
 *  Calculate the size of the Pbi Interface Table and allocate the
 *  memory.
 */

    Pbi_Ptr = ( PBI_ENTRY  * ) calloc ( pbi_num, sizeof ( PBI_ENTRY ) );

    if ( Pbi_Ptr == NULL ) {
        free ( Pbi_Table );
        Pbi_Num = 0;
        tui_msg ( M_YELLOW, "error %d in allocation of Pbi entries", errno );
        fclose ( ddf_ffp );
        return ( -1 );
    }

/*
 *  Calculate the size of the Pbi Definition Table and allocate the
 *  memory.
 */

    Pbi_Def = ( struct pbi_def  * ) calloc ( pbi_num, sizeof ( struct pbi_def ) );

    if ( Pbi_Def == NULL ) {
        Pbi_Num = 0;
        tui_msg ( M_YELLOW, "error %d in allocation of Pbi definition entries", errno );
        fclose ( ddf_ffp );
        return ( -1 );
    }
```

```
/*
 *  Skip through records before pbis to get the pbi definitions
 */

    if ( entry_num > 0 )
        fseek ( ddf_ffp, ( 91  * entry_num ), 1 );
    if ( limit_num > 0 )
        fseek ( ddf_ffp, ( 88  * limit_num ), 1 );
    if ( tab_num > 0 )
        fseek ( ddf_ffp, ( 40  * tab_num ), 1 );
    if ( tmplt_num > 0 )
        fseek ( ddf_ffp, ( 45  * tmplt_num ), 1 );

/*
 *  Skip the interactive input object records
 */

    if ( inter_num > 0 ) {

        for ( i = 0; i < inter_num; i++ ) {

            fscanf ( ddf_ffp, "%hd", &entry_num );
            fscanf ( ddf_ffp, "%hd", &inter_type );
            if ( inter_type == SLIDER_BAR ) {

                fscanf ( ddf_ffp, "%95 *c" );

            } else if ( inter_type == TOGGLE_SWITCH ) {

                fscanf ( ddf_ffp, "%26 *c" );

            }
        }
    }

/*
 *  Skip the history tab records
 */

    if ( htab_num > 0 )
        fscanf ( ddf_ffp, "%*c", ( 28 * htab_num ) );

/*
 *  position pointer at the beginning of Mtext structure and read into
 *  memory the mtext structure.
 */

    for ( j = 0; j < mltxt_num; j++ ) {
        fscanf ( ddf_ffp, "%d", & ( mult_entry ) );
        fscanf ( ddf_ffp, "%hd", & ( num_values ) );
        k = 0;
        getc ( ddf_ffp );
        do {
            getc ( ddf_ffp );
            k++;
        } while ( k < 6 );

        for ( k = 0; k < num_values; k++ ) {
            fscanf ( ddf_ffp, "%d" );
            m = 0;
            getc ( ddf_ffp );
            do {
                getc ( ddf_ffp );
                m++;
```

```
            } while ( m < 6 );
        }
    }

/*
 *  Position pointer at the beginning of Pbi structure and read into
 *  memory Pbi Emulation Interface table and Pbi Definition Table
 */

    pbi_ptr = Pbi_Ptr;
    pbi_def_ptr = Pbi_Def;

    for ( i = 0; i < pbi_num; i++ ) {
        fscanf ( ddf_ffp, "%d", & ( pbi_entry_num ) );

/*
 *      Store values in the Pbi Emulation Interface table for calling WSA process
 */

        fscanf ( ddf_ffp, "%10s", ( pbi_ptr->pbi_msid ) );

        for ( j = 0; j < MSID_LEN; j++ ) {
            if ( ! ( isalnum ( pbi_ptr->pbi_msid[j] ) ) )
                pbi_ptr->pbi_msid[j] = ' ';
        }

        fscanf ( ddf_ffp, "%1s", &temp_active_flag );
        pbi_ptr->active_flag = atoi ( &temp_active_flag );
        fscanf ( ddf_ffp, "%*5c" );
        fscanf ( ddf_ffp, "%hd", & ( pbi_ptr->pbi_type ) );
        fscanf ( ddf_ffp, "%hd", & ( pbi_ptr->lock_num ) );
        fscanf ( ddf_ffp, "%hd", & ( pbi_ptr->group_num ) );
        fscanf ( ddf_ffp, "%hd", & ( pbi_ptr->panel_num ) );
        fscanf ( ddf_ffp, "%d", & ( pbi_ptr->button_num ) );

/*
 *       pre-Set the backlighting if any is required
 */

        fscanf ( ddf_ffp, "%hd", & ( pbi_def_ptr->pbi_bklght ) );

        if ( pbi_def_ptr->pbi_bklght == PBI_PRESET_ON )
            pbi_ptr->feedback_ind = PBI_BKLGHT_ON;
        else
            pbi_ptr->feedback_ind = PBI_BKLGHT_OFF;

        pbi_ptr->modify_flag = 0;

        fscanf ( ddf_ffp, "%hd", &grph_indx );

        fscanf ( ddf_ffp, "%hd", &grph_color );

/*
 *      Store values in the Pbi Display Definition Table for internal processing.
 *      Note that the Y values are swapped.
 */

        fscanf ( ddf_ffp, "%lf", &pbi_def_ptr->pbi_ul_x );
        fscanf ( ddf_ffp, "%lf", &pbi_def_ptr->pbi_ul_y );
        fscanf ( ddf_ffp, "%lf", &pbi_def_ptr->pbi_lr_x );
        fscanf ( ddf_ffp, "%lf", &pbi_def_ptr->pbi_lr_y );

        pbi_def_ptr->pbi_ul_y = 100.0 - pbi_def_ptr->pbi_ul_y;
        pbi_def_ptr->pbi_lr_y = 100.0 - pbi_def_ptr->pbi_lr_y;
```

```
/*
 *      Save pixel versions of the corner coordinates.
 */

        pbi_def_ptr->pbi_ul_x_p =
            (int)(pbi_def_ptr->pbi_ul_x * Dm_Address->display[Disp_Num].factor_x);
        pbi_def_ptr->pbi_lr_x_p =
            (int)(pbi_def_ptr->pbi_lr_x * Dm_Address->display[Disp_Num].factor_x);
        pbi_def_ptr->pbi_ul_y_p =
            (int)(pbi_def_ptr->pbi_ul_y * Dm_Address->display[Disp_Num].factor_y);
        pbi_def_ptr->pbi_lr_y_p =
            (int)(pbi_def_ptr->pbi_lr_y * Dm_Address->display[Disp_Num].factor_y);

/*
 *      Read in the Pbi destination
 */

        size = 4;
         ( pbi_def_ptr->pbi_dest ) = ( char  * ) malloc ( size );
        pbi_def_ptr->pbi_dest[3] = '/0';

        pbi_nxt = pbi_def_ptr->pbi_dest;

        fscanf ( ddf_ffp, "%s", pbi_nxt );
        pbi_def_ptr->pbi_dest_len = strlen ( pbi_nxt );

/*
 *      Skip labels -- for use in the Displayer only
 */

        fscanf ( ddf_ffp, "%hd", &labels );

        for ( j = 0; j < labels; j++ ) {
            fscanf ( ddf_ffp, "%hd", &label_ndx );
        }

/*
 *      Read in the Pbi variable length message
 */

        fscanf ( ddf_ffp, "%hd", & ( pbi_def_ptr->pbi_mesg_len ) );
        if ( ( pbi_def_ptr->pbi_mesg_len ) > 0 ) {
             ( pbi_def_ptr->pbi_message ) = ( char  * )
                calloc ( 1, ( pbi_def_ptr->pbi_mesg_len ) );

            pbi_nxt = pbi_def_ptr->pbi_message;
            fscanf ( ddf_ffp, "%1s", pbi_nxt );
            pbi_nxt++;
            for ( k = 0; k < pbi_def_ptr->pbi_mesg_len - 1; k++ ) {
                fscanf ( ddf_ffp, "%c", pbi_nxt );
                pbi_nxt++;
            }

/*
 *          If the type is DTE or MED then set the PBI EMULATION arguments
 */

            if ( ( pbi_ptr->pbi_type == DT )
                || ( pbi_ptr->pbi_type == ME ) ) {
                pbi_ptr->arg_len = pbi_def_ptr->pbi_mesg_len;
                pbi_ptr->arg_ptr = pbi_def_ptr->pbi_message;
            }
        }
```

```
/*
 *      Read in the Pbi variable length dependent Msids
 */

        fscanf ( ddf_ffp, "%hd", & ( pbi_def_ptr->pbi_dep_msid_cnt ) );

        if ( ( pbi_def_ptr->pbi_dep_msid_cnt ) > 0 ) {

             ( pbi_def_ptr->pbi_dep_msids ) =
                 ( char  * ) calloc ( pbi_def_ptr->pbi_dep_msid_cnt, MSID_LEN );

            pbi_nxt_msid = ( struct pbi_msid_rec  * ) pbi_def_ptr->pbi_dep_msids;

            for ( k = 0; k < ( pbi_def_ptr->pbi_dep_msid_cnt ); k++ ) {
                fscanf ( ddf_ffp, "%s", pbi_nxt_msid );
                for ( j = 0; j < MSID_LEN; j++ ) {
                    if ( ! ( isalnum ( pbi_nxt_msid->pbi_msid[j] ) ) )
                        pbi_nxt_msid->pbi_msid[j] = ' ';
                }
                pbi_nxt_msid++;
            }

            if ( pbi_ptr->pbi_type == DG ) {
                pbi_ptr->arg_len = pbi_def_ptr->pbi_dep_msid_cnt  * MSID_LEN;
                pbi_ptr->arg_ptr = pbi_def_ptr->pbi_dep_msids;
            }
        }

/*
 *      Read New Line Character
 */

        fscanf ( ddf_ffp, "%*c", 1 );

/*
 *      Track group count in header table
 */

        if ( pbi_ptr->group_num != prev_group_num ) {
             ( Pbi_Table->group_count ) ++;
            prev_group_num = pbi_ptr->group_num;
        }
        pbi_def_ptr++;
        pbi_ptr++;

    }

    fclose ( ddf_ffp );

    D(printf("END read_pbi\n"));
    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: read_pf.c
 *
 *  This function reads in function key definitions.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  K. Noonan       - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <constants.h>
#include <pf_key.h>
#include <disp.h>
#include <wex/EXmsg.h>


extern struct pfkey_defs    Def_Pfkeys[PFKEY_COUNT];
extern struct pfkey_defs    Act_Pfkeys[PFKEY_COUNT];

extern int                  errno;

extern char                 Disp_Path[DNAME_LEN];


int read_pf ( default_flag, disp_name )

    char            default_flag,           /* default flag for pf keys         */
                    disp_name[DNAME_LEN];   /* key definition file display name */
{
    static char     pfkey[PFKEY_COUNT][10] =
        {"PFKEY1\0", "SPFKEY1\0", "PFKEY2\0", "SPFKEY2\0",
        "PFKEY3\0", "SPFKEY3\0", "PFKEY4\0", "SPFKEY4\0",
        "PFKEY5\0", "SPFKEY5\0", "PFKEY6\0", "SPFKEY6\0",
        "PFKEY7\0", "SPFKEY7\0", "PFKEY8\0", "SPFKEY8\0",
        "PFKEY9\0", "SPFKEY9\0", "PFKEY10\0", "SPFKEY10\0",
        "PFKEY11\0", "SPFKEY11\0", "PFKEY12\0", "SPFKEY12\0",
        "PFKEY13\0", "SPFKEY13\0", "PFKEY14\0", "SPFKEY14\0"
    };

    FILE           *pf_fp, *fopen ( );

    char            *cmdstring,             /* command string to send to parse_cmd      */
                    *current_key,           /* current pf key                           */
                    *instring,              /* string read in from the file             */
                    pf_file_name[80],       /* pfkey file name to be opened             */
                    default_file_name[80];  /* default file name to be opened           */

    short           version;                /* version of pf key definition file        */

    int             i,                      /* temporary loop variable                  */
                    j = 0,                  /* temporary loop variable                  */
                    k,                      /* temporary loop variable                  */
                    l,                      /* temporary loop variable                  */
```

```
                mark = 0;               /* temporary indexing variable          */
    unsigned    total_bytes;            /* total bytes copy from active pfkey struc */

    D(printf("START read_pf\n"));
/*
 *  Build the pf key default file name and open it. If cannot open advise.
 */
    if ( default_flag == ON ) {

        strcpy ( default_file_name, Disp_Path );
        strcat ( default_file_name, "default.pf" );

        if ( ( pf_fp = fopen ( default_file_name, "r" ) ) == NULL ) {

            tui_msg ( M_YELLOW, "Error %d on open for %s PF key file ", errno,
                  default_file_name );
            return ( -1 );

        }
    }
/*
 *  Else build the pf key file name and open it.
 */
    else {

        strcpy ( pf_file_name, disp_name );
        strcat ( pf_file_name, ".pf" );

        if ( ( pf_fp = fopen ( pf_file_name, "r" ) ) == NULL )
            return ( -1 );

    }

/*
 *  Read the version of the pf definition file and check the status.
 */
    fscanf ( pf_fp, "%*8c" );
    fscanf ( pf_fp, "%hd", &version );

/*
 *  Initialization of active pfkey structure
 */
    if ( version <= VERSION ) {

        for ( k = 0; k < PFKEY_COUNT; k++ ) {

            Act_Pfkeys[k].valid_flag = VALID;
            Act_Pfkeys[k].defined = NO;
            Act_Pfkeys[k].prompt_flag = NO;

        }

/*
 *  Loop through all the PF keys of the keyboard.
 */
        for ( i = 0; i < PFKEY_COUNT; i++ ) {

/*
```

```
 *  Read the current key from the pfkey file. If pfkey is not in the
 *  correct order advise and return to calling routine.
 */

            current_key = ( char * ) calloc ( 1, 8 );
            fscanf ( pf_fp, "%s", current_key );

            if ( feof ( pf_fp ) )
                break;

            if ( strcmp ( current_key, pfkey[i] ) != 0 ) {

                if ( ( current_key[0] != '-' ) && ( current_key[1] != '-' ) ) {

                    tui_msg ( M_WHITE,
                            "function key definition file is not in the proper format" );
                    return ( -1 );

                } else {

                    instring = ( char * ) calloc ( 1, 120 );
                    fgets ( instring, 120, pf_fp );
                    free ( instring );
                    i--;

                }
            }

/*
 *  Read the next string from the pfkey file. If the next string reads
 *  in as the next pfkey, discontinue process for current pfkey and
 *  label it as INVALID.
 */

            else {

                instring = ( char * ) calloc ( 1, 120 );
                fgets ( instring, 120, pf_fp );
                cmdstring = ( char * ) calloc ( 1, strlen ( instring ) );

                for ( l = 0; l < strlen ( instring ); l++ ) {
                    if ( instring[l] == ' ' )
                        mark = l + 1;
                    else
                        break;
                }

                strncpy ( cmdstring, & ( instring[mark] ), strlen(instring) - (mark + 1) )
;

                if ( i % 2 == 0 )
                    j = i >> 1;
                else
                    j = i / 2 + 14;

                if ( strlen ( cmdstring ) > 1 )
                    parse_cmd ( & ( Act_Pfkeys[j] ), cmdstring, strlen ( cmdstring ),
                                PFKEY, version );

                if ( Act_Pfkeys[j].valid_flag == INVALID ) {

                    if ( j > 14 )
                        tui_msg ( M_YELLOW, "Shift PFkey %d improperly defined", j - 13 );
                    else
```

```
                        tui_msg ( M_YELLOW, "PFkey %d improperly defined", j + 1 );

                    }
                    free ( cmdstring );
                    free ( instring );

                }

                free ( current_key );

            }

    /*
     *  Check the users command string to proceed with further processing.
     */

            fclose ( pf_fp );

    /*
     *  If default flag is set copy active pfkey structure into default pfkey
     *  struct
     */

            if ( default_flag == YES ) {

                total_bytes = sizeof ( struct pfkey_defs ) * PFKEY_COUNT;
                memcpy ( ( char * ) Def_Pfkeys, ( char * ) Act_Pfkeys, total_bytes );

            }
        } else {

            tui_msg ( M_WHITE, "Illegal version of pfkey definition file with this software" )
    ;
            return ( -1 );

        }

        D(printf("END read_pf\n"));
        return ( 0 );
    }
```

```
/*****************************************************************************
 * MODULE NAME: read_plt.c
 *
 *  This routine reads the DDF plot file into memory.
 *  Returns:        0 if successful, -1 on error
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Richard Romeo - Ford Aerospace Corporation/Houston
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Ronnie Killough - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include <Xm/ScrolledW.h>
#include <Xm/DrawingA.h>
#include <fcntl.h>
#include <sys/types.h>
#include <constants.h>
#include <disp.h>
#include <DDdisp.h>
#include <DDplot.h>
#include <wex/EXmsg.h>


extern struct dm_shmemory   *Dm_Address;    /* ptr to DM shared memory      */
extern struct bg_recs       Bg_Rec;         /* bg records                   */
extern int                  errno;          /* system error number          */
extern short                Pixels[128];    /* index array into colormap    */
extern Widget               Draw_Win;       /* widget ID of disp drawg area */


read_plt (disp_num, plot_ptr)

    short   disp_num;                   /* effective display number         */

    struct plot_ptrs   *plot_ptr;       /* ptr to current plot record       */
{
    FILE   *fopen();
    XtCallbackProc  cb_expose_plot();   /* cb proc for expose on plot wndow */

    Arg     args[10];                   /* arg list for widgets             */

    FILE    *plot_fp,                   /* ptr to plot information file     */
            *lim_fp;                    /* ptr to plot limit line file      */

    struct plot_hdr     *header_ptr;    /* ptr to plot header information   */
    struct plot_tmplt   *tmplt_ptr;     /* ptr to plot positional info.     */
    struct axis_info    *axis_ptr;      /* ptr thru plot axis records       */
    struct msid_info    *msid_ptr;      /* ptr thru plot msid records       */
    struct lim_lines    *nline_ptr;     /* ptr thru plot nominal line recs  */
    struct lim_lines    *lline_ptr;     /* ptr thru plot limit line records */
    struct plot_pts     *nline_pts_ptr; /* ptr thru nominal line pt pairs   */
    struct plot_pts     *lline_pts_ptr; /* ptr thru limit line pt pairs     */
```

```
    double  diff;                       /* diff. between high & low scale   */

    short   upd_rate,                   /* time plot update rate from DDF   */
            version,                    /* plot file version from DDF       */
            color,                      /* temp color # read from DDF file  */
            access_rs,                  /* access restriction code from DDF */
            plot_hori,                  /* horizontal font size             */
            plot_vert,                  /* vertical font size               */
            match;                      /* MSID match flag                  */

    int     i, j, k, m, p,              /* loop count variables             */
            xaxes_num,                  /* nbr of x axis records            */
            yaxes_num,                  /* nbr of y axis records            */
            msid_num,                   /* nbr of msid records              */
            actual_msids,               /* nbr of actual msid records       */
            nline_num,                  /* nbr of nominal line records      */
            lline_num,                  /* nbr of limit line records        */
            total_nbr_records,          /* total nbr of plot records        */
            total_nbr_axes,             /* total nbr of axis records        */
            name_len,                   /* length of plot file name         */
            dw_width, dw_height;        /* size of top-lvl drawing area     */

    char    temp[15],                   /* used to read character fields    */
            plot_file_name[DNAME_LEN+5], /* plot file name                  */
            file_name[DNAME_LEN+5],     /* limit/nom file name              */
            wex_path_name[DNAME_LEN + 5],   /* plot name with WEX path name */
            user_disp_name[DNAME_LEN + 5],  /* plot name w/ user-display name */
            unv_file[DNAME_LEN + 5],    /* plot name with .unv ext          */
            plt_file[DNAME_LEN + 5],    /* plot name with .plt ext          */
            sample[4],                  /* used to read sample rate         */
            plot_style[5];              /* used to read the character style */


    D(printf("START read_plt\n"));
/*
 *  Set local plot template ptr variable.
 */

    tmplt_ptr = plot_ptr->plot_pos;

/*
 *  Save the current size of the top-level drawing area
 *  for later reset.  This is a kluge to overcome an
 *  apparent bug in Motif which causes the size of the
 *  drawing area to be altered in a systematic fashion
 *  whenever a widget is made a child of that drawing area.
 */

    i=0;
    XtSetArg(args[i], XmNwidth, &dw_width); i++;
    XtSetArg(args[i], XmNheight, &dw_height); i++;
    XtGetValues(Draw_Win, args, i);

/*
 *  Set the size of the drawing area widget to the size of the drawing
 *  area dimensions and create the drawing area.  Place the drawing area
 *  at the appropriate location relative to the scrolled widget.
 *  Set the background color of the drawing area.
 */

    i = 0;
    XtSetArg(args[i], XmNx, tmplt_ptr->bb_xul); i++;
    XtSetArg(args[i], XmNy, tmplt_ptr->bb_yul); i++;
```

```
    XtSetArg(args[i], XmNwidth,  tmplt_ptr->drw_width); i++;
    XtSetArg(args[i], XmNheight, tmplt_ptr->drw_height); i++;
    XtSetArg(args[i], XmNbackground, Bg_Rec.s_color); i++;

    XtManageChild(plot_ptr->draw_win = XmCreateDrawingArea
                          (Draw_Win, "Plot Draw", args, i));

/*
 *  Restore the size of the display top-level drawing area.
 *  See related comment above.
 */

    i=0;
    XtSetArg(args[i], XmNwidth, dw_width); i++;
    XtSetArg(args[i], XmNheight, dw_height); i++;
    XtSetValues(Draw_Win, args, i);

/*
 *  Add expose event callback routine for the plot drawable.
 */

    XtAddCallback(plot_ptr->draw_win, XmNexposeCallback,
                                          cb_expose_plot, disp_num);

/*
 *  Build the universal plot file name
 */

    if (!strncmp(Dm_Address->display[disp_num].display_name, "/WEX/", 5)) {
        strcpy (wex_path_name, Dm_Address->display[disp_num].display_name);
        name_len = (int) strlen(wex_path_name);

        while (wex_path_name[name_len] != '/' && name_len > 0)
            name_len--;

        wex_path_name[name_len + 1] = NULL;
        strcat (wex_path_name, plot_ptr->plot_pos->tmplt_nam);
        strcpy (plot_ptr->plot_name, wex_path_name);
        strcpy (plot_file_name, wex_path_name);
        strcat (wex_path_name, ".plt");
        strcpy (plt_file, wex_path_name);
        strcpy (user_disp_name, Dm_Address->display[disp_num].plot_path);
        strcat (user_disp_name, plot_ptr->plot_pos->tmplt_nam);
        strcpy (plot_ptr->plot_data_file, user_disp_name);
        strcat (user_disp_name, ".unv");
        strcpy (unv_file, user_disp_name);
    }
    else {
        strcpy (plot_file_name, Dm_Address->display[disp_num].display_name);
        name_len = (int) strlen (plot_file_name);

        while (plot_file_name[name_len] != '/' && name_len > 0)
            name_len--;

        plot_file_name[name_len + 1] = NULL;
        strcat (plot_file_name, plot_ptr->plot_pos->tmplt_nam);
        strcpy (plot_ptr->plot_data_file, plot_file_name);
        strcpy (plot_ptr->plot_name, plot_file_name);
        strcpy (unv_file, plot_file_name);
        strcat (unv_file, ".unv");
        strcpy (plt_file, plot_file_name);
        strcat (plt_file, ".plt");
    }
```

```
    plot_fp = fopen(unv_file, "r");

    if (plot_fp == NULL) {
        plot_fp = fopen (plt_file, "r");

        if (plot_fp == NULL) {
            tui_msg (M_YELLOW, "Error %d on opening plot file", errno);
            return (-1);
        }
    }

/*
 *  Read the software version. If correct version continue processing by read-
 *  ing in the plot file information.
 */

    fscanf (plot_fp, "%hd", &version);

    if (version > VERSION) {
        tui_msg (M_WHITE, "Incorrect software version for this release");
        fclose(plot_fp);
        return (-1);
    }

    fscanf (plot_fp, "%*51c");
    fscanf (plot_fp, "%d %d %d %d %d %d %hd %hd", &xaxes_num, &yaxes_num,
            &msid_num, &actual_msids, &nline_num, &lline_num, &upd_rate, &access_rs);

    if (version >= 3)
        fscanf (plot_fp, "%*5c");

/*
 *  Check the access restriction code to see if the display is either a
 *  Medical or Payload restricted display.  If the display is access
 *  restricted and the position Id does not match the access restriction, then
 *  exit out of this routine.
 */

    switch (access_rs) {

        case MEDICAL_USR:

            if (strcmp(Dm_Address->display[disp_num].pos_id, "MED\0") != 0) {
                tui_msg(M_YELLOW, "Medical Display - access restricted");
                fclose(plot_fp);
                return(0);
            }

            break;

        case PAYLOAD_USR:

            if (strcmp(Dm_Address->display[disp_num].pos_id, "PAY\0") != 0) {
                tui_msg (M_YELLOW, "Payload Display - access restricted");
                fclose(plot_fp);
                return(0);
            }

            break;

        default:
            break;

    }   /* end switch */
```

```
    total_nbr_records =
                xaxes_num + yaxes_num + actual_msids + nline_num + lline_num;

    if (total_nbr_records == 0) {
        tui_msg(M_WHITE, "There are no plot records ");
        fclose (plot_fp);
        return (-1);
    }

/*
 *  Allocate memory for plot header structure and copy
 *  plot header info into it.
 */

    header_ptr = (struct plot_hdr  *) calloc (1, sizeof (struct plot_hdr));

    if (header_ptr == NULL) {
        tui_msg (M_YELLOW, "Error %d allocating plot header memory", errno);
        fclose (plot_fp);
        return (-1);
    }

    plot_ptr->header = header_ptr;

    header_ptr->xaxes_num = xaxes_num;
    header_ptr->yaxes_num = yaxes_num;
    header_ptr->msid_num = msid_num;
    header_ptr->actual_msids = actual_msids;
    header_ptr->nline_num = nline_num;
    header_ptr->lline_num = lline_num;
    header_ptr->upd_rate = upd_rate;
    header_ptr->access_rs = access_rs;

    if (msid_num > 0) {
        plot_ptr->plt_decom = (struct shm_decom *)
            calloc(plot_ptr->header->msid_num, sizeof(struct shm_decom));

        if (plot_ptr->plt_decom == NULL) {
            tui_msg(M_YELLOW, "Error on allocating memory for plot decom");
            fclose(plot_fp);
            return (-1);
        }

    }

/*
 *  Read in the plot definition file msid records and store them into memory.
 */

    if (actual_msids > 0) {

        msid_ptr = (struct msid_info *)
                        calloc(actual_msids, sizeof(struct msid_info));

        if (msid_ptr == NULL) {
            tui_msg(M_YELLOW, "Error %d allocating plot msid memory", errno);
            fclose (plot_fp);
            return (-1);
        }

        plot_ptr->msids = msid_ptr;

        for (j = 0; j < actual_msids; j++) {
```

```
            fscanf (plot_fp, "%hd", &msid_ptr->msid_indx);
            msid_ptr->msid_indx = j;
            fscanf (plot_fp, "%s", msid_ptr->msid_name);
            fscanf (plot_fp, "%s", sample);

            if (sample[0] != 'L')
                msid_ptr->sample = atoi (sample);
            else
                msid_ptr->sample = -1;

            fscanf (plot_fp, "%s", msid_ptr->data_src);

    /*
     *      Skip the ppl file and occr numbers
     */

            if (version >= 3)
                fscanf (plot_fp, "%*10c");

            fscanf (plot_fp, "%s", temp);
            msid_ptr->xory_axis = temp[0];
            fscanf (plot_fp, "%d", &msid_ptr->axis_num);
            fscanf (plot_fp, "%s", msid_ptr->plot_msid);

            fscanf (plot_fp, "%s", temp);
            msid_ptr->plot_type = temp[0];

            fscanf (plot_fp, "%d", &msid_ptr->line_type);
            fscanf (plot_fp, "%f", &msid_ptr->line_width);
            fscanf (plot_fp, "%s", msid_ptr->plot_char);
            fscanf (plot_fp, "%s", plot_style);
            fscanf (plot_fp, "%hd", &plot_hori);
            fscanf (plot_fp, "%hd", &plot_vert);
            msid_ptr->plot_font = font_num(disp_num, plot_style, plot_hori, plot_vert);
            fscanf (plot_fp, "%hd", &msid_ptr->icon_indx);
            fscanf (plot_fp, "%s", temp);
            msid_ptr->plot_conn = temp[0];
            fscanf (plot_fp, "%hd", &color);
            msid_ptr->plot_color = Pixels[color];
            fscanf (plot_fp, "%d", &msid_ptr->stat_flag);
            fscanf (plot_fp, "%d", &msid_ptr->miss_flag);
            fscanf (plot_fp, "%hd", &color);
            msid_ptr->stat_color = Pixels[color];
            fscanf (plot_fp, "%hd", &color);
            msid_ptr->miss_color = Pixels[color];
            fscanf (plot_fp, "%hd", &color);
            msid_ptr->ovl_color = Pixels[color];
            fscanf (plot_fp, "%hd", &color);
            msid_ptr->limt_color = Pixels[color];
            fscanf (plot_fp, "%hd", &color);
            msid_ptr->crit_color = Pixels[color];
            fscanf (plot_fp, "%d", &msid_ptr->oper_type);
            fscanf (plot_fp, "%f", &msid_ptr->oper_width);
            fscanf (plot_fp, "%d", &msid_ptr->crit_type);
            fscanf (plot_fp, "%f", &msid_ptr->crit_width);
            msid_ptr->pair_ptr = NULL;
            msid_ptr->first_pt = YES;

            msid_ptr++;

        }   /* End of -for- (total nbr of msids) */

    /*
     *      Set the pair index pointers
```

```
 */

        msid_ptr = plot_ptr->msids;

        for (i = 0; i < actual_msids; i++) {

            /* the current msid is represented by plot_ptr->msids + i  */

            if ((msid_ptr + i)->pair_ptr == NULL) {
                match = NO;
                k = i + 1;

                while (match == NO && k < actual_msids) {

                    if ((msid_ptr + k)->pair_ptr == NULL) {

                        if (    !strcmp((msid_ptr + i)->msid_name,
                                        (msid_ptr + k)->plot_msid)
                            &&  !strcmp((msid_ptr + i)->plot_msid,
                                        (msid_ptr + k)->msid_name)) {
                            (msid_ptr + i)->pair_ptr = msid_ptr + k;
                            (msid_ptr + k)->pair_ptr = msid_ptr + i;
                            match = YES;
                        }
                        else
                            k++;
                    }
                    else
                        k++;

                }   /* end while */

            }       /* end of if ... == NULL */

        }           /* end of -for- (i) */

    }   /* end -if- (actual msid > 0) */

/*
 *  Read in the plot definition file axis records and store them into memory.
 */

    total_nbr_axes = xaxes_num + yaxes_num;

    if (total_nbr_axes > 0) {

        axis_ptr = (struct axis_info *)
                        calloc(total_nbr_axes, sizeof(struct axis_info));

        if (axis_ptr == NULL) {
            tui_msg(M_YELLOW, "Error %d allocating plot axis memory", errno);
            fclose (plot_fp);
            return (-1);
        }

        plot_ptr->axis = axis_ptr;

        for (m = 0; m < total_nbr_axes; m++) {

/*
 *          Read next axis record from plot file
 */

            fscanf (plot_fp, "%s", temp);
```

```
            axis_ptr->axis_xory = temp[0];
            fscanf (plot_fp, "%d", &axis_ptr->axis_num);
            fscanf (plot_fp, "%hd", &axis_ptr->axis_type);
            fscanf (plot_fp, "%s", temp);
            axis_ptr->scal_type = temp[0];
            fscanf (plot_fp, "%d", &axis_ptr->end_code);
            fscanf (plot_fp, "%hd", &axis_ptr->axis_pos);
            fscanf (plot_fp, "%hd", &color);
            axis_ptr->axis_col = Pixels[color];
            fscanf (plot_fp, "%s", axis_ptr->low_scale);
            fscanf (plot_fp, "%s", axis_ptr->high_scal);
            fscanf (plot_fp, "%s", temp);
            axis_ptr->auto_flag = temp[0];
            fscanf (plot_fp, "%hd", &axis_ptr->grad_vals);
            fscanf (plot_fp, "%s", temp);
            axis_ptr->vis_flag = temp[0];
            fscanf (plot_fp, "%s", temp);
            axis_ptr->grid_flag = temp[0];
            fscanf (plot_fp, "%hd", &axis_ptr->grid_gran);
            fscanf (plot_fp, "%hd", &axis_ptr->grid_type);
            fscanf (plot_fp, "%hd", &color);
            axis_ptr->grd_color = Pixels[color];
            fscanf (plot_fp, "%hd", &axis_ptr->maj_ticks);
            fscanf (plot_fp, "%hd", &axis_ptr->min_ticks);

    /*
     *          Adjust # graduations, grid granularity, and major ticks
     *          for overflow plotting surface.

            axis_ptr->grad_vals = (axis_ptr->grad_vals * 2) - 1;
            axis_ptr->grid_gran = (axis_ptr->grid_gran * 2) - 1;
            axis_ptr->maj_ticks = (axis_ptr->maj_ticks * 2) - 1;
     */

    /*
     *          Format low/high scales into numeric form.  Save the original
     *          scale values.  Adjust the low/high scale values to encompass
     *          the overflow plot area.
     */

            if (axis_ptr->scal_type == 'T') {
                axis_ptr->low_value = (double) p_atimei(axis_ptr->low_scale);
                axis_ptr->high_value = (double) p_atimei(axis_ptr->high_scal);

                axis_ptr->org_low_val = axis_ptr->low_value;
                axis_ptr->org_high_val = axis_ptr->high_value;
            } else {
                sscanf(axis_ptr->low_scale, "%lf", &axis_ptr->low_value);
                sscanf (axis_ptr->high_scal, "%lf", &axis_ptr->high_value);

                axis_ptr->org_low_val = axis_ptr->low_value;
                axis_ptr->org_high_val = axis_ptr->high_value;

    /* RLK 10/17/90
                diff = (axis_ptr->high_value - axis_ptr->low_value) / 2.0;
                axis_ptr->low_value = (double) axis_ptr->low_value - diff;
                axis_ptr->high_value = (double) axis_ptr->high_value + diff;
    */
            }

    /*
     *          Transform axis position to pixel coordinates.  Set
     *          current axis position to permanent pixel axis.
     */
```

```
            if (axis_ptr->axis_xory == 'X')
                    axis_ptr->cur_axis_pos = axis_ptr->pixel_axis_pos
                            = (short) ((100.0 - axis_ptr->axis_pos)
                                    * plot_ptr->plot_pos->factor_y);
            else
                    axis_ptr->cur_axis_pos = axis_ptr->pixel_axis_pos
                            = (short) (axis_ptr->axis_pos
                                    * plot_ptr->plot_pos->factor_x);

/*
 *          Set axis to active state.
 */

            axis_ptr->axis_active = YES;

            axis_ptr++;
        }
    }

    fclose(plot_fp);

/*
 * Read the limit/nominal line file (if exists)
 */


    if (lline_num <= 0 && nline_num <= 0)
        return(0);

    /* build file name for limit/nominal line w/ name & .nll extension. */

    strcpy (file_name, plot_file_name);
    strcat (file_name, ".lln");

    /* open limit plot file if one exists */

    lim_fp = fopen (file_name, "r");

    if (lim_fp == NULL) {
        tui_msg(M_YELLOW, "Error %d on opening limit or nominal line file %s",
                    errno, file_name);
        header_ptr->lline_num = 0;
        header_ptr->nline_num = 0;
        return (0);
    }

/*
 *  Check for incompatible limit line file version.
 */

    fscanf (lim_fp, "%hd", &version);

    if (version > VERSION) {
        tui_msg(M_YELLOW, "Incorrect software version for limit line file");
        header_ptr->lline_num = 0;
        header_ptr->nline_num = 0;
        fclose(lim_fp);
        return (0);
    }

    fscanf (lim_fp, "%*51c");
    fscanf (lim_fp, "%hd", &xaxes_num);
    fscanf (lim_fp, "%hd", &yaxes_num);
```

```
    fscanf (lim_fp, "%d", &msid_num);
    fscanf (lim_fp, "%d", &actual_msids);
    fscanf (lim_fp, "%d", &nline_num);
    fscanf (lim_fp, "%d", &lline_num);
    fscanf (lim_fp, "%hd", &upd_rate);
    fscanf (lim_fp, "%hd", &access_rs);

/*
 *  Read in the plot definition file nom records
 */

    if (nline_num > 0) {

        nline_ptr = (struct lim_lines *)
                calloc(nline_num,  sizeof(struct lim_lines));

        if (nline_ptr == NULL) {
            tui_msg(M_YELLOW, "Error %d allocating plot nline memory", errno);
            fclose (lim_fp);
            return (-1);
        }

        plot_ptr->nline = nline_ptr;

        for (p = 0; p < nline_num; p++) {
            fscanf (lim_fp, "%*s");
            fscanf (lim_fp, "%s", temp);
            nline_ptr->line_type = temp[0];
            fscanf (lim_fp, "%hd", &color);
            nline_ptr->line_color = Pixels[color];
            fscanf (lim_fp, "%d", &nline_ptr->xaxis_num);
            fscanf (lim_fp, "%d", &nline_ptr->yaxis_num);
            fscanf (lim_fp, "%s", temp);
            nline_ptr->line_def = temp[0];
            fscanf (lim_fp, "%hd", &nline_ptr->point_num);
            fscanf (lim_fp, "%hd", &nline_ptr->polyn_num);

            nline_pts_ptr = (struct plot_pts *)
                    calloc(nline_ptr->point_num, sizeof(struct plot_pts));

            if (nline_pts_ptr == NULL) {
                tui_msg(M_YELLOW, "Error %d allocating plot nline point memory",
                            errno);
                fclose(lim_fp);
                return (-1);
            }

            nline_ptr->plot_pts_ptr = nline_pts_ptr;

            for (i = 0; i < nline_ptr->point_num; i++) {
                fscanf (lim_fp, "%s", nline_pts_ptr->point_x);
                fscanf (lim_fp, "%s", nline_pts_ptr->point_y);
                nline_pts_ptr++;
            }

            for (i = 0; i < (nline_ptr->polyn_num); i++)
                fscanf (lim_fp, "%s", nline_ptr->coeff[i]);

            nline_ptr++;
        }
    }

/*
 *  Read in the plot definition file lim records and store them into memory.
```

```
*/

    if (lline_num > 0) {

        lline_ptr = (struct lim_lines  *)
                            calloc(lline_num, sizeof(struct lim_lines));

        if (lline_ptr == NULL) {
            tui_msg(M_YELLOW, "Error %d allocating plot lline memory", errno);
            fclose(lim_fp);
            return (-1);
        }

        plot_ptr->lline = lline_ptr;

        for (p = 0; p < lline_num; p++) {
            fscanf (lim_fp, "%*s");
            fscanf (lim_fp, "%s", temp);
            lline_ptr->line_type = temp[0];
            fscanf (lim_fp, "%hd", &color);
            lline_ptr->line_color = Pixels[color];
            fscanf (lim_fp, "%d", &lline_ptr->xaxis_num);
            fscanf (lim_fp, "%d", &lline_ptr->yaxis_num);
            fscanf (lim_fp, "%s", temp);
            lline_ptr->line_def = temp[0];
            fscanf (lim_fp, "%hd", &lline_ptr->point_num);
            fscanf (lim_fp, "%hd", &lline_ptr->polyn_num);

            lline_pts_ptr = (struct plot_pts *)
                    calloc(lline_ptr->point_num, sizeof(struct plot_pts));

            if (lline_pts_ptr == NULL) {
                tui_msg(M_YELLOW, "Error %d allocating plot lline point memory",
                                errno);
                fclose(lim_fp);
                return(-1);
            }

            lline_ptr->plot_pts_ptr = lline_pts_ptr;

            for (i = 0; i < lline_ptr->point_num; i++) {
                fscanf (lim_fp, "%s", lline_pts_ptr->point_x);
                fscanf (lim_fp, "%s", lline_pts_ptr->point_y);
                lline_pts_ptr++;
            }

            for (i = 0; i < (lline_ptr->polyn_num); i++)
                fscanf (lim_fp, "%s", lline_ptr->coeff[i]);

            lline_ptr++;
        }

    }   /* end -if- (lline > 0) */

    D(printf("END read_plt\n"));

    return (0);
}
```

```
/*****************************************************************************
 * MODULE NAME: readbg.c
 *
 *  This routine reads into memory the DDF background file.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Richard Romeo - Ford Aerospace Corporation/Houston
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Ronnie Killough - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <X11/Xlib.h>
#include <constants.h>
#include <DDdisp.h>
#include <disp.h>
#include <wex/EXmsg.h>


extern struct dm_shmemory   *Dm_Address;    /* ptr to DM shared memory      */
extern struct bg_recs       Bg_Rec;         /* background records           */

extern int                  errno;          /* system error return value    */

extern short                Pixels[128];    /* index into colormap          */


int readbg (disp_num)

    short   disp_num;                       /* effective display number     */
{
    FILE    *fopen();
    char    *malloc(),
            *calloc();

    FILE   *ddf_fp;                         /* ptr to background DDF file   */

    struct disp_info        *display;       /* ptr to display info struct   */
    struct rec_header       *bg_text_ptr;   /* ptr thru bg text records     */
    struct graph_record     *bg_graph_ptr;  /* ptr thru bg graphical recs   */
    struct vtext_record     *vtext_ptr;     /* ptr to vector text entry     */
    struct line_record      *line_ptr;      /* ptr to line record           */
    struct rectangle_record *rect_ptr;      /* ptr to rectangle record      */
    struct polygon_record   *poly_ptr;      /* ptr to polygon record        */
    struct graph_pts        *poly_pts_ptr;  /* ptr thru polygon vertices    */
    struct circle_record    *circle_ptr;    /* ptr to circle record         */
    struct arc_record       *arc_ptr;       /* ptr to arc record            */
    struct ellipse_record   *ellipse_ptr;   /* ptr to ellipse record        */
    struct curve_record     *curve_ptr;     /* ptr to curve record          */
    struct graph_pts        *curve_pts_ptr; /* ptr thru curve points        */

    float   factor_x, factor_y,     /* coordinates transformation factors   */
            x_size, y_size,         /* size of display window in wrld coord */
            fx1, fy1, fx2, fy2,     /* temporary float x/y values           */
            radius,                 /* temp for radius                      */
```

```
            angle1, angle2;          /* temp for radian angles              */

    long    horz_size, vert_size;    /* font size read from bg file         */

    short   version,                 /* background DDF file version         */
            access_rs;               /* access restriction code             */

    int     i, j, k, w, h,           /* loop counters                       */
            total_nbr_records,       /* total number of background records  */
            color,                   /* local var for index into Pixels map */
            disp_x_size,             /* size of physical display terminal   */
            disp_y_size,             /*      in millimeters.                */
            screen;                  /* X screen id of display terminal     */

    char    *text_char_ptr,          /* ptr thru characters of text items   */
            ddf_file_name[DNAME_LEN],   /* name of bg DDF file              */
            font_style[5];           /* font style read from bg file        */


    D(printf("START readbg\n"));
/*
 *  Initialize local display variable (not X display...disp_info struct)
 */

    display = &Dm_Address->display[disp_num];

/*
 *  Retrieve display name and open bg file
 */

    strcpy (ddf_file_name, display->display_name);
    strcat (ddf_file_name, ".bg");

    ddf_fp = fopen (ddf_file_name, "r");

    if (ddf_fp == NULL) {
        tui_msg(M_YELLOW, "Error %d on reading DDF background file", errno);
        return (-1);
    }

/*
 *  Read the software version. If correct version continue processing by read-
 *  ing in the number of graphical, character and subdrawing records. Read in
 *  access restriction code.
 */

    fscanf (ddf_fp, "%hd", &version);

    if (version > VERSION) {
        tui_msg(M_WHITE, "Incorrect software version for this release");
        close (ddf_fp);
        return (-1);
    }

/*
 *  Read the Display Definition Background header file and store into memory
 */

    fscanf(ddf_fp, "%*49c");
    fscanf(ddf_fp, "%f", &x_size);
    if ((x_size < 0) || (x_size > 100))
        tui_msg(M_WHITE, "x coordinate is out of range");

    fscanf(ddf_fp, "%f", &y_size);
```

```
    if ((y_size < 0) || (y_size > 100))
        tui_msg(M_WHITE, "y coordinate is out of range");

    fscanf(ddf_fp, "%d", &color);
    if ((color < 0) || (color > 128))
        tui_msg(M_WHITE, "Incorrect color parameter");
    Bg_Rec.s_color = Pixels[color];

    fscanf(ddf_fp, "%hd", &(Bg_Rec.graph_num));
    fscanf(ddf_fp, "%hd", &(Bg_Rec.char_num));
    fscanf(ddf_fp, "%*5c");
    fscanf(ddf_fp, "%hd", &access_rs);
    fscanf(ddf_fp, "%*10c");

/*
 *  Calculate the world coordinate transformation
 *  factors based on the size of the display
 */

    screen = DefaultScreen ( Dm_Address->xdisplay[disp_num] );
    disp_x_size = DisplayWidth  ( Dm_Address->xdisplay[disp_num], screen );
    disp_y_size = DisplayHeight ( Dm_Address->xdisplay[disp_num], screen );

    display->size_x   = (int)( (double)disp_x_size * (x_size / 100.0) );
    display->size_y   = (int)( (double)disp_y_size * (y_size / 100.0) );
    display->factor_x = (float)display->size_x / 100.0;
    display->factor_y = (float)display->size_y / 100.0;

    factor_x = display->factor_x;
    factor_y = display->factor_y;

/*
 *  Calculate the number of bg records.  If no records,
 *  close bg DDF file and exit.
 */

    total_nbr_records = Bg_Rec.graph_num + Bg_Rec.char_num;

    if (total_nbr_records == 0) {
        fclose (ddf_fp);
        return (0);
    }

/*
 *  Check the access restriction code to see if the display is either a
 *  Medical or Payload restricted display.  If the display is access
 *  restricted and the position Id does not match the access restriction, then
 *  exit out of this routine.
 */

    switch (access_rs) {

        case MEDICAL_USR:
            if ((strcmp (display->pos_id, "MED\0") != 0)) {
                tui_msg(M_YELLOW, "Medical Display - access restricted");
                fclose (ddf_fp);
                return (total_nbr_records);
            }
            break;
        case PAYLOAD_USR:
            if ((strcmp (display->pos_id, "PAY\0") != 0)) {
                tui_msg(M_YELLOW, "Payload Display - access restricted");
                fclose (ddf_fp);
                return (total_nbr_records);
```

```
            }
            break;
        default:
            break;
    }

/*
 *  Check for graphical records...allocate memory
 */

    if (Bg_Rec.graph_num > 0) {

        Bg_Rec.graph_rec = (struct graph_record *)
                    calloc(Bg_Rec.graph_num, sizeof (struct graph_record));

        bg_graph_ptr = Bg_Rec.graph_rec;

        if (bg_graph_ptr == NULL) {
            tui_msg(M_YELLOW, "Error %d on graphical record calloc", errno);
            return (-1);
        }
    }

/*
 *  Read bg graphic records
 */

    for (j = 0; j < Bg_Rec.graph_num; j++) {
        fscanf (ddf_fp, "%hd", &(bg_graph_ptr->graph_typ));
        bg_graph_ptr->redraw_flag = NO;

        switch (bg_graph_ptr->graph_typ) {

            case LINE:

                /* Allocate memory for a line record and setup local pointer */

                bg_graph_ptr->graph_ptr = malloc(sizeof(struct line_record));

                if (bg_graph_ptr->graph_ptr == NULL) {
                    tui_msg(M_YELLOW, "Error %d on background line record malloc", errno);
                    return (-1);
                }

                line_ptr = (struct line_record *) bg_graph_ptr->graph_ptr;

                /* read in a single line record */

                fscanf (ddf_fp, "%d", &color);
                line_ptr->graph_col = Pixels[color];
                fscanf (ddf_fp, "%hd", &(line_ptr->line_type));
                fscanf (ddf_fp, "%f", &(line_ptr->line_wdth));
                fscanf (ddf_fp, "%f", &fx1);
                fscanf (ddf_fp, "%f", &fy1);
                fscanf (ddf_fp, "%f", &fx2);
                fscanf (ddf_fp, "%f", &fy2);

                /* convert world coordinate line endpoints to X pixel coord */

                line_ptr->point1_x = (int) (fx1 * factor_x);
                line_ptr->point1_y = (int) ((100.0 - fy1) * factor_y);
                line_ptr->point2_x = (int) (fx2 * factor_x);
                line_ptr->point2_y = (int) ((100.0 - fy2) * factor_y);
```

```
                break;


            case RECTANGLE:

                /* allocate memory for a rectangle record and setup local ptr */

                bg_graph_ptr->graph_ptr =
                                malloc(sizeof(struct rectangle_record));

                if (bg_graph_ptr->graph_ptr == NULL) {
                    tui_msg(M_YELLOW, "Error %d on background rectangle record malloc" ,er
rno);
                    return (-1);
                }

                rect_ptr = (struct rectangle_record *) bg_graph_ptr->graph_ptr;

                /* read in a single rectangle record */

                fscanf (ddf_fp, "%d", &color);
                rect_ptr->graph_col = Pixels[color];
                fscanf (ddf_fp, "%hd", &(rect_ptr->line_type));
                fscanf (ddf_fp, "%f", &(rect_ptr->line_wdth));
                fscanf (ddf_fp, "%hd", &(rect_ptr->pat_type));

/* RLK 10/23/90 Set pattern type to NO_CHANGE since not implemented */
                rect_ptr->pat_type = NO_CHANGE;

                fscanf (ddf_fp, "%f", &(rect_ptr->pat_sizex));
                fscanf (ddf_fp, "%f", &(rect_ptr->pat_sizey));

/* RLK 9/6/90    strange ordering of rectangle coordinates...might make
                sure this is right at some point. */

                fscanf (ddf_fp, "%f", &fx1);
                fscanf (ddf_fp, "%f", &fy2);
                fscanf (ddf_fp, "%f", &fx2);
                fscanf (ddf_fp, "%f", &fy1);

                /* convert world coordinates to X pixel coordinates */

                /* upper left x coord of rectangle */

                rect_ptr->ul_x = (int) (fx1 * factor_x);

                /* subtract ul y-coord from 100 since X-windows origin is
                    upper left */

                rect_ptr->ul_y = (int) ((100.0 - fy1) * factor_y);

                /* convert lower right coords to width/height by subtracting
                    upper left coords. */

                rect_ptr->width = (int) ((fx2 - fx1) * factor_x);
                rect_ptr->height = (int) ((fy1 - fy2) * factor_y);

                break;


            case POLYGON:

                /* allocate memory for a polygon record and setup local ptr */
```

```
                bg_graph_ptr->graph_ptr=malloc(sizeof(struct polygon_record));

                if (bg_graph_ptr->graph_ptr == NULL) {
                    tui_msg(M_YELLOW, "Error %d on background polygon record malloc", errn
o);
                    return (-1);
                }

                poly_ptr = (struct polygon_record  *) bg_graph_ptr->graph_ptr;

                /* read in a single polygon record */

                fscanf (ddf_fp, "%d", &color);
                poly_ptr->graph_col = Pixels[color];
                fscanf (ddf_fp, "%hd", &(poly_ptr->line_type));
                fscanf (ddf_fp, "%f", &(poly_ptr->line_wdth));
                fscanf (ddf_fp, "%hd", &(poly_ptr->pat_type));

/* RLK 10/23/90 Set pattern type to NO_CHANGE since not implemented */
                poly_ptr->pat_type = NO_CHANGE;

                fscanf (ddf_fp, "%f", &(poly_ptr->pat_sizex));
                fscanf (ddf_fp, "%f", &(poly_ptr->pat_sizey));
                fscanf (ddf_fp, "%d", &(poly_ptr->nmbr_pts));

                /* allocate memory for set of vertices & setup local ptr */

                poly_ptr->poly_pts_ptr = (struct graph_pts *)
                        calloc(poly_ptr->nmbr_pts, sizeof(struct graph_pts));

                if (poly_ptr->poly_pts_ptr == NULL) {
                    tui_msg(M_YELLOW, "Error %d on background polygon record malloc", errn
o);
                    return (-1);
                }

                poly_pts_ptr = poly_ptr->poly_pts_ptr;

                /* read in polygon vertices */

                for (w = 0; w < poly_ptr->nmbr_pts; w++) {
                    fscanf (ddf_fp, "%f", &fx1);
                    fscanf (ddf_fp, "%f", &fy1);
                    poly_pts_ptr->point_x = (int) (fx1 * factor_x);
                    poly_pts_ptr->point_y = (int) ((100.0 - fy1) * factor_y);
                    poly_pts_ptr++;
                }

                break;


            case CIRCLE:

                /* allocate memory for circle record & setup local pointer */

                bg_graph_ptr->graph_ptr = malloc(sizeof(struct circle_record));

                if (bg_graph_ptr->graph_ptr == NULL) {
                    tui_msg(M_YELLOW, "Error %d on background circle record malloc", errnc
);
                    return (-1);
                }

                circle_ptr = (struct circle_record *) bg_graph_ptr->graph_ptr;
```

```
                /* read in a single circle record */

                fscanf (ddf_fp, "%d", &color);
                circle_ptr->graph_col = Pixels[color];
                fscanf (ddf_fp, "%hd", &(circle_ptr->line_type));
                fscanf (ddf_fp, "%f", &(circle_ptr->line_wdth));
                fscanf (ddf_fp, "%hd", &(circle_ptr->pat_type));

/* RLK 10/23/90 Set pattern type to NO_CHANGE since not implemented */
                circle_ptr->pat_type = NO_CHANGE;

                fscanf (ddf_fp, "%f", &(circle_ptr->pat_sizex));
                fscanf (ddf_fp, "%f", &(circle_ptr->pat_sizey));
                fscanf (ddf_fp, "%f", &fx1);
                fscanf (ddf_fp, "%f", &fy1);
                fscanf (ddf_fp, "%f", &radius);

/* RLK 9/10/90  Converting radius from world coord to X pixel coord based on
                x transformation factor since transforming on both x and y
                would cause an ellipse to form instead of a circle.  Need to
                find out how the Display Builder computes the world coord
                radius in order to compute this properly. */

                circle_ptr->radius = radius * factor_x;

                /* convert world coord center & radius to X pixel coords */

                circle_ptr->bb_x =
                    (int) ((fx1 * factor_x) - circle_ptr->radius);
                circle_ptr->bb_y =
                    (int) (((100.0 - fy1) * factor_y) - circle_ptr->radius);

                break;


            case ARC:

                /* allocate memory for arc record & setup local pointer */

                bg_graph_ptr->graph_ptr = malloc (sizeof (struct arc_record));

                if (bg_graph_ptr->graph_ptr == NULL) {
                    tui_msg(M_YELLOW, "Error %d on background arc record malloc", errno);
                    return (-1);
                }

                arc_ptr = (struct arc_record   *) bg_graph_ptr->graph_ptr;

                /* read in a single arc record */

                fscanf (ddf_fp, "%d", &color);
                arc_ptr->graph_col = Pixels[color];
                fscanf (ddf_fp, "%hd", &(arc_ptr->line_type));
                fscanf (ddf_fp, "%f", &(arc_ptr->line_wdth));
                fscanf (ddf_fp, "%hd", &(arc_ptr->pat_type));

/* RLK 10/23/90 Set pattern type to NO_CHANGE since not implemented */
                arc_ptr->pat_type = NO_CHANGE;

                fscanf (ddf_fp, "%f", &(arc_ptr->pat_sizex));
                fscanf (ddf_fp, "%f", &(arc_ptr->pat_sizey));
                fscanf (ddf_fp, "%f", &fx1);
                fscanf (ddf_fp, "%f", &fy1);
```

```
                fscanf (ddf_fp, "%f", &fx2);
                fscanf (ddf_fp, "%f", &fy2);
                fscanf (ddf_fp, "%f", &angle1);
                fscanf (ddf_fp, "%f", &angle2);

                /* convert world coord center & axes to X pixel coordinates */

                arc_ptr->bb_x = (int) ((fx1 - (fx2 * 0.5)) * factor_x);
                arc_ptr->bb_y = (int)(((100.0 - fy1) - (fy2 * 0.5)) * factor_y);
                arc_ptr->maj_axis = (int) (fx2 * factor_x);
                arc_ptr->min_axis = (int) (fy2 * factor_y);

                /* convert radian angles to 64th degree angles */

/* RLK 9/11/90  Need to find out what the reference point is for the angles
                given in the DDF files.  They are given in radians. */

/*
                arc_ptr->angle1 = ?
                arc_ptr->angle2 = ?
*/

                break;


            case ELLIPSE:

                /* allocate memory for an ellipse record & setup local ptr */

                bg_graph_ptr->graph_ptr =
                                    malloc(sizeof(struct ellipse_record));

                if (bg_graph_ptr->graph_ptr == NULL) {
                    tui_msg(M_YELLOW, "Error %d on background ellipse record malloc", errn
o);

                    return (-1);
                }

                ellipse_ptr = (struct ellipse_record *)bg_graph_ptr->graph_ptr;

                /* read in a single ellipse record */

                fscanf (ddf_fp, "%d", &color);
                ellipse_ptr->graph_col = Pixels[color];
                fscanf (ddf_fp, "%hd", &(ellipse_ptr->line_type));
                fscanf (ddf_fp, "%f", &(ellipse_ptr->line_wdth));
                fscanf (ddf_fp, "%hd", &(ellipse_ptr->pat_type));

/* RLK 10/23/90 Set pattern type to NO_CHANGE since not implemented */
                ellipse_ptr->pat_type = NO_CHANGE;

                fscanf (ddf_fp, "%f", &(ellipse_ptr->pat_sizex));
                fscanf (ddf_fp, "%f", &(ellipse_ptr->pat_sizey));
                fscanf (ddf_fp, "%f", &fx1);
                fscanf (ddf_fp, "%f", &fy1);
                fscanf (ddf_fp, "%f", &fx2);
                fscanf (ddf_fp, "%f", &fy2);

                /* convert world coordinate centerpoint
                    and major/minor axes to X pixel coord */

                ellipse_ptr->bb_x = (int) (fx1 * factor_x);
                ellipse_ptr->bb_y = (int) ((100.0 - fy1) * factor_y);
                ellipse_ptr->maj_axis = (int) (fx2 * factor_x);
```

```
            ellipse_ptr->min_axis = (int) (fy2 * factor_y);

            break;


        case CURVE:

            /* allocate memory for an curve record & setup local ptr */

            bg_graph_ptr->graph_ptr = malloc(sizeof(struct curve_record));

            if (bg_graph_ptr->graph_ptr == NULL) {
                tui_msg(M_YELLOW, "Error %d on background curvical record malloc", err
no);
                return (-1);
            }

            curve_ptr = (struct curve_record   *) bg_graph_ptr->graph_ptr;

            /* read in a single curve record */

            fscanf (ddf_fp, "%d", &color);
            curve_ptr->graph_col = Pixels[color];
            fscanf (ddf_fp, "%hd", &(curve_ptr->line_type));
            fscanf (ddf_fp, "%f", &(curve_ptr->line_wdth));
            fscanf (ddf_fp, "%d", &(curve_ptr->nmbr_pts));

            /* allocate memory for curve points and setup local pointer */

            curve_ptr->curve_pts_ptr = (struct graph_pts *)
                    calloc(curve_ptr->nmbr_pts, sizeof(struct graph_pts));

            if (curve_ptr->curve_pts_ptr == NULL) {
                tui_msg(M_YELLOW, "Error %d on background curve record malloc", errno)
;
                return (-1);
            }

            curve_pts_ptr = curve_ptr->curve_pts_ptr;

            /* read in curve pts, converting world coord to X pixel coord */

            for (k = 0; k < curve_ptr->nmbr_pts; k++) {
                fscanf (ddf_fp, "%f", &fx1);
                fscanf (ddf_fp, "%f", &fy1);
                curve_pts_ptr->point_x = (int) (fx1 * factor_x);
                curve_pts_ptr->point_y = (int) ((100.0 - fy1) * factor_y);
                curve_pts_ptr++;
            }

            break;


        case VECT_TXT:

            /* allocate memory for an vector text rec & setup local ptr */

            bg_graph_ptr->graph_ptr = malloc(sizeof (struct vtext_record));

            if (bg_graph_ptr->graph_ptr == NULL) {
                tui_msg(M_YELLOW, "Error %d on background vector text record malloc",
errno);
            return (-1);
            }
```

```
                vtext_ptr = (struct vtext_record *) bg_graph_ptr->graph_ptr;

                /* read in a vector text font and alignment info */

                fscanf (ddf_fp, "%d", &color);
                vtext_ptr->graph_col = Pixels[color];
                fscanf (ddf_fp, "%d", &(vtext_ptr->font_style));
                fscanf (ddf_fp, "%f", &(vtext_ptr->vert_size));
                fscanf (ddf_fp, "%f", &(vtext_ptr->char_width));
                fscanf (ddf_fp, "%f", &(vtext_ptr->char_spac));
                fscanf (ddf_fp, "%d", &(vtext_ptr->char_angl));

                vtext_ptr->font_num = font_num(disp_num, "def", 0, 0);

                fscanf (ddf_fp, "%f", &fx1);
                fscanf (ddf_fp, "%f", &fy1);

                /* convert world coordinate positions to X pixel coordinates */

                vtext_ptr->x_position = (int) (fx1 * factor_x);
                vtext_ptr->y_position = (int) ((100.0 - fy1) * factor_y);

                /* allocate memory for vector text chars and setup local ptr */

                fscanf (ddf_fp, "%d", &(vtext_ptr->char_len));
                vtext_ptr->record_item = malloc (vtext_ptr->char_len + 1);

                if (vtext_ptr->record_item == NULL) {
                    tui_msg(M_YELLOW, "error %d on vector text record item malloc ", errno
);
                    fclose (ddf_fp);
                }

                text_char_ptr = vtext_ptr->record_item;

                /* read in vector text string */

                fscanf (ddf_fp, "%*c");

                for (h = 0; h < vtext_ptr->char_len; h++) {
                    fscanf (ddf_fp, "%c", text_char_ptr);
                    text_char_ptr++;
                }

                *text_char_ptr = NULL;
                vtext_ptr++;

                break;

            default:
                break;

        }   /* End of switch (graph_typ) */

        bg_graph_ptr++;

    }   /* End of graphical records -for- loop */

/*
 *  If have text records, alloc memory for all text records & setup local ptr
 */

    if (Bg_Rec.char_num > 0) {
```

```
        Bg_Rec.record = (struct rec_header *)
                            calloc(Bg_Rec.char_num, sizeof(struct rec_header));

        if (Bg_Rec.record == NULL) {
            tui_msg(M_YELLOW, "Error %d on character record calloc", errno);
            return (-1);
        }

        bg_text_ptr = Bg_Rec.record;
    }

/*
 *  Read in text records
 */

    for (i = 0; i < Bg_Rec.char_num; i++) {

        /* read in font info */

        fscanf (ddf_fp, "%s", font_style);
        fscanf (ddf_fp, "%ld", &horz_size);
        fscanf (ddf_fp, "%ld", &vert_size);

        /* fetch font number */

        bg_text_ptr->font_num =
                        font_num(disp_num, font_style, horz_size, vert_size);

        /* read text attributes, convert world coord to X pixel coord */

        fscanf (ddf_fp, "%d", &color);
        bg_text_ptr->color = Pixels[color];
        fscanf (ddf_fp, "%f", &fx1);
        fscanf (ddf_fp, "%f", &fy1);
        bg_text_ptr->x_position = (int) (fx1 * factor_x);
        bg_text_ptr->y_position = (int) ((100.0 - fy1) * factor_y);
        fscanf (ddf_fp, "%d", &(bg_text_ptr->char_len));

        /* allocate space for a single text item & setup local ptr */

        bg_text_ptr->record_item = malloc (bg_text_ptr->char_len + 1);

        if (bg_text_ptr->record_item == NULL) {
            tui_msg(M_YELLOW, "error %d on record item malloc ", errno);
            fclose (ddf_fp);
        }

        text_char_ptr = bg_text_ptr->record_item;

        /* read in text item */

        fscanf (ddf_fp, "%*c");

        for (h = 0; h < bg_text_ptr->char_len; h++) {
            fscanf (ddf_fp, "%c", text_char_ptr);
            text_char_ptr++;
        }

        *text_char_ptr = NULL;
        bg_text_ptr->redraw_flag = NO;
        bg_text_ptr++;

    }   /* End of for (total # rec...) */
```

```
/*
 *  Close file
 */

    fclose (ddf_fp);

    D(printf("END readbg\n"));

    return (total_nbr_records);
}
```

```
/*****************************************************************************
 * MODULE NAME: readfg.c
 *
 *  This routine reads the foreground DDF records into memory.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Richard Romeo - Ford Aerospace Corporation/Houston
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Ronnie Killough - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <X11/Xlib.h>
#include <fcntl.h>
#include <sys/types.h>
#include <constants.h>
#include <DDdisp.h>
#include <disp.h>
#include <DDplot.h>
#include <DDfg_graph.h>
#include <wex/EXmsg.h>


extern GC   FGC[MAX_GC];     /* pre-allocated GC array */

extern struct dm_shmemory   *Dm_Address;     /* ptr to DM shared memory      */
extern struct msid_ent      *Msid;           /* ptr to msid entries          */
extern struct limit_ent     *Limit;          /* ptr to limit entries         */
extern struct mtext_ent     *Mtext;          /* ptr to multilvl text entries */
extern struct pbi_ent       *Pbi;            /* ptr to PBI entries           */
extern struct tabular_ent   *Tab;            /* ptr to tabular entries       */
extern struct plot_tmplt    *Tmplt;          /* ptr to template entries      */
extern struct hist_tab      *Htab;           /* ptr to history tab entries   */
extern struct fg_file_header    *Ffile;      /* ptr to fg file hdr records   */
extern struct plot_ptrs     *Plot_info_ptr; /* ptr to plot records          */
extern struct fg_recs       Fg_rec;          /* fg graphics records          */
extern struct label_ent     *Lab;            /* ptr to label records         */
extern struct scale_ent     *Scale;          /* ptr to scale records         */
extern struct ddd_ent       *Ddd;            /* ptr to ddd records           */

extern int  errno;                           /* system return error value    */

extern short    Nbr_of_plots,                /* number of plot records       */
                Pixels[128];                 /* index array into colormap    */


int readfg(disp_num)

    short   disp_num;               /* display # associated with fg records */

{
    FILE   *fopen();
    char   *malloc();
    char   *calloc();
```

```
FILE    *ddf_ffp;                /* ptr to foreground DDF file          */

struct msid_ent *msid_ptr;       /* ptr thru msid entries               */
struct tabular_ent *tab_ptr;     /* ptr thru tabular entries            */
struct plot_tmplt  *tmplt_ptr;   /* ptr thru template entries           */
struct limit_ent   *limit_ptr;   /* ptr thru limit entries              */
struct mtext_ent   *mtext_ptr;   /* ptr thru multi-lvl text entries     */
struct val_txt *text_ptr;        /* ptr thru multi-lvl text string      */
struct plot_ptrs   *plot_ptr;    /* ptr thru plot records               */
struct hist_tab *htab_ptr;       /* ptr thru history tab records        */
struct scale_ent   *scale_ptr;   /* ptr thru scale entries              */
struct label_ent   *label_ptr;   /* ptr thru label entries              */
struct ddd_ent *ddd_ptr;         /* ptr thru ddd entries                */
struct pbi_ent *pbi_ptr;         /* ptr thru pbi entries                */
struct label_indices    *pbi_label_ptr;
                                 /* ptr thru each pbi's labels          */

double  wwidth, wheight;     /* dim. of plot bbx in world coord         */
float   high_scale_x,        /* local high x scale value                */
        high_scale_y,        /* local high y scale value                */
        x_size, y_size,      /* temp x/y size of display in wrld coord  */
        factor_x, factor_y,  /* world-to-X transformation factors       */
        fx1, fy1, fx2, fy2;  /* local holders for world coords read in  */

long    mltxt_num = 0,       /* number of multi-level text entries      */
        tab_num = 0,         /* number of tabular entries               */
        entry_num = 0,       /* number of msid entries                  */
        limit_num = 0,       /* number of limit entries                 */
        pbi_num = 0,         /* number of pbi entries                   */
        icon_num = 0,        /* number of icon entries                  */
        tmplt_num = 0,       /* number of template entries              */
        label_num = 0,       /* number of label entries                 */
        scale_num = 0,       /* number of scale entries                 */
        ddd_num = 0,         /* number of ddd entries                   */
        inter_num = 0,       /* number of inter entries                 */
        htab_num = 0,        /* number of history tab entries           */
        ppl_num = 0,         /* number of ppl entries                   */
        horz_size,           /* local variable for horizontal size      */
        vert_size;           /* local variable for vertical size        */


unsigned int    size,        /* size calculated for memory allocation   */
                font_count[MAX_FONTS],
                             /* array for counting font freq            */
                color_count[129];
                             /* array for counting color freq           */

int     total_nbr_fgrecords,     /* total nbr of foreground records     */
        error,                   /* read_plt return value holder        */
        dep_msids,               /* number of dependent msids           */
        mesg_len,                /* pbi message length                  */
        tmp,                     /* skip record variable                */
        i, j, k,                 /* loop count variables                */
        color,                   /* temp holder for color values read in */
        one, two, three;         /* vars to hold color/font counts      */

short   *loc_ddd_aptr,           /* ptr to ddd msid indx                */
        version;                 /* version of the fg ddf file read     */

char    sample[4],               /* sample value                        */
        local_text[7],           /* temp for mlti-lvl txt string        */
        font_style[5],           /* local variable for font style       */
        fg_file_name[50],        /* fg file path name                   */
        disp_name[33],           /* temp holder for display name        */
```

```
            position[17],           /* temp holder for display position     */
            temp[2],                /* variable for reading in one char     */
            *loc_lbl_ptr,           /* temp ptr to label string             */
            *rec_pointer;           /* ptr to newly allocated memory        */


    D(printf("START readfg\n"));
/*
 *  Setup local world coordinate transformation factors
 */

    factor_x = Dm_Address->display[disp_num].factor_x;
    factor_y = Dm_Address->display[disp_num].factor_y;

/*
 *  Build DDF filename and open the file
 */

    strcpy( fg_file_name, Dm_Address->display[disp_num].display_name );
    strcat( fg_file_name, ".fg" );

    ddf_ffp = fopen( fg_file_name, "r" );

    if ( ddf_ffp == NULL ) {
        tui_msg( M_YELLOW, "Error %d on reading DDF foreground file", errno );
        return( -1 );
    }

/*
 *  Read in header info
 */

    fscanf( ddf_ffp, "%hd", &version );

    if ( version > VERSION ) {
        tui_msg( M_YELLOW, "The Display Builder version is incorrect" );
        fclose ( ddf_ffp );
        return ( 0 );
    }

    fscanf( ddf_ffp, "%32c", disp_name );
    fscanf( ddf_ffp, "%18c", position );
    fscanf( ddf_ffp, "%f", &x_size );
    fscanf( ddf_ffp, "%f", &y_size );
    fscanf( ddf_ffp, "%d", &color );
    fscanf( ddf_ffp, "%d", &tab_num );
    fscanf( ddf_ffp, "%d", &entry_num );
    fscanf( ddf_ffp, "%d", &pbi_num );
    fscanf( ddf_ffp, "%d", &icon_num );
    fscanf( ddf_ffp, "%d", &tmplt_num );
    fscanf( ddf_ffp, "%d", &mltxt_num );
    fscanf( ddf_ffp, "%d", &limit_num );

    if ( version > 2 ) {
        fscanf( ddf_ffp, "%d", &label_num );
        fscanf( ddf_ffp, "%d", &scale_num );
        fscanf( ddf_ffp, "%d", &ddd_num );
        fscanf( ddf_ffp, "%d", &inter_num );
        fscanf( ddf_ffp, "%d", &htab_num );
        fscanf( ddf_ffp, "%d", &ppl_num );
    }

/*
 *  Sum records, check for no records
```

```
 */

    total_nbr_fgrecords = tab_num + entry_num + mltxt_num + limit_num +
                          tmplt_num + htab_num + icon_num + label_num +
                          scale_num + ddd_num + pbi_num;

    if ( total_nbr_fgrecords == 0 ) {
        tui_msg( M_WHITE, "There are zero foreground records" );
        fclose( ddf_ffp );
        return( 0 );
    }

/*
 *  Calculate the size of the Display Definition File and allocate the
 *  memory.
 */

    size =  sizeof( struct fg_file_header ) +
            sizeof( struct msid_ent ) * entry_num +
            sizeof( struct plot_tmplt ) * tmplt_num +
            sizeof( struct hist_tab ) * htab_num +
            sizeof( struct tabular_ent ) * tab_num +
            sizeof( struct mtext_ent ) * mltxt_num +
            sizeof( struct fgr_record ) * icon_num +
            sizeof( struct limit_ent ) * limit_num +
            sizeof( struct scale_ent ) * scale_num +
            sizeof( struct ddd_ent ) * ddd_num +
            sizeof( struct pbi_ent ) * pbi_num +
            sizeof( struct label_ent ) * label_num;

    rec_pointer = malloc ( size + 1 );

    if ( rec_pointer == NULL ) {
        tui_msg( M_YELLOW, "error %d on malloc of file size ", errno );
        fclose( ddf_ffp );
        return( -1 );
    }

/*
 *  Set up pointer to start of foreground file header
 */

    Ffile = ( struct fg_file_header * ) rec_pointer;

/*
 *  Copy the foreground DDF header values
 *  into the global header structure
 */

/* RLK 9/4/90...doesn't fix any out-of-ranges in code below...why? */

    Ffile->Version = version;
    strncpy(Ffile->Disp_Name, disp_name, 32);
    strncpy(Ffile->Position, position, 17);

    Ffile->X_Size = x_size;
    if ( ( Ffile->X_Size < 0 ) || ( Ffile->X_Size > 100 ) )
        tui_msg( M_WHITE, "The X size of the display is out of range" );

    Ffile->Y_Size = y_size;
    if ( ( Ffile->Y_Size < 0 ) || ( Ffile->Y_Size > 100 ) )
        tui_msg( M_WHITE, "The Y size of the display is out of range" );

    Ffile->S_Color = Pixels[color];
```

```
    if ( ( color < 0 ) || ( color > 128 ) )
        tui_msg(M_WHITE,
                    "The foreground screen color is out of range %d", color);

    Ffile->Tab_Num = tab_num;
    if ( ( Ffile->Tab_Num < 0 ) || ( Ffile->Tab_Num > 3000 ) )
        tui_msg( M_WHITE, "Number of tabular entries is out of range" );

    Ffile->Entry_Num = entry_num;
    if ( ( Ffile->Entry_Num < 0 ) || ( Ffile->Entry_Num > 3000 ) )
        tui_msg( M_WHITE, "Number of msid entries is out of range" );

    Ffile->PBI_Num = pbi_num;
    Ffile->Icon_Num = icon_num;
    Ffile->Tmplt_Num = tmplt_num;
    Ffile->Mltxt_Num = mltxt_num;

    Ffile->Limit_Num = limit_num;
    if ( ( Ffile->Limit_Num < -1 ) || ( Ffile->Limit_Num > 3000 ) )
        tui_msg( M_WHITE, "Number of limit entries is out of range" );

    Ffile->Htab_Num = htab_num;

/*
 *  Read access restriction code
 */

    fscanf( ddf_ffp, "%2c", Ffile->Access_Rs );

/*
 *  Initialize color count array (GC optimization code)
 */

    for (i=0; i<129; i++)
        color_count[i]=0;

/*
 *  Read the DDF MSID entries record file and store into memory
 */

    Msid = ( struct msid_ent * ) ( Ffile + 1 );
    msid_ptr = Msid;

    for ( i = 0; i < entry_num; i++ ) {
        fscanf ( ddf_ffp, "%d", & ( msid_ptr->MSID_Entr ) );
        fscanf ( ddf_ffp, "%s", msid_ptr->MSID );
        fscanf ( ddf_ffp, "%s", sample );

        if ( sample[0] != 'L' ) {
            msid_ptr->Sample = atoi ( sample );
            if ( msid_ptr->Sample < -2 || msid_ptr->Sample == 0 ||
                    msid_ptr->Sample > 200 )
                tui_msg( M_WHITE, "The sample number is out of range" );
        } else
            msid_ptr->Sample = -1;

        fscanf ( ddf_ffp, "%s", msid_ptr->Data_Src );
        fscanf ( ddf_ffp, "%hd", & ( msid_ptr->Scrn_Type ) );

        if ( ( msid_ptr->Scrn_Type < 0 ) || ( msid_ptr->Scrn_Type > 55 ) )
            tui_msg( M_WHITE, "The screen data type is out of range %hd",
                        msid_ptr->Scrn_Type );

        fscanf ( ddf_ffp, "%d", & ( color ) );
```

```
msid_ptr->Nom_Color = Pixels[color];

if ( ( color < 0 ) || ( color > 128 ) )
    tui_msg( M_WHITE, "The nominal color parameter is out of range %d",
                color );
else
    color_count[color]++;

fscanf ( ddf_ffp, "%d", & ( color ) );
msid_ptr->Sta_Color = Pixels[color];

if ( ( color < 0 ) || ( color > 128 ) )
    tui_msg( M_WHITE, "The static color parameter is out of range %d",
                color );
else
    color_count[color]++;

fscanf ( ddf_ffp, "%d", & ( color ) );
msid_ptr->Ovr_Color = Pixels[color];

if ( ( color < 0 ) || ( color > 128 ) )
    tui_msg( M_WHITE, "The override color parameter is out of range %d",
                color );

fscanf ( ddf_ffp, "%d", & ( color ) );
msid_ptr->Dead_Color = Pixels[color];

if ( ( color < 0 ) || ( color > 128 ) )
    tui_msg( M_WHITE, "The dead color parameter is out of range %d",
                color );

fscanf ( ddf_ffp, "%f", & ( msid_ptr->X_NDC_St ) );

if ( ( msid_ptr->X_NDC_St < 0 ) || ( msid_ptr->X_NDC_St > 100 ) )
    tui_msg( M_WHITE,
                "The X coordinate status character is out of range" );

fscanf ( ddf_ffp, "%f", & ( msid_ptr->Y_NDC_St ) );

if ( ( msid_ptr->Y_NDC_St < 0 ) || ( msid_ptr->Y_NDC_St > 100 ) )
    tui_msg( M_WHITE,
                "The Y coordinate status character is out of range" );

fscanf ( ddf_ffp, "%hd", & ( msid_ptr->Stat_Flag ) );

if ( ( msid_ptr->Stat_Flag < 0 ) || ( msid_ptr->Stat_Flag > 1 ) )
    tui_msg( M_WHITE, "The status character flag is out of range" );

fscanf( ddf_ffp, "%d", & ( msid_ptr->Tab_Index ) );

if ( ( msid_ptr->Tab_Index < 0 ) || ( msid_ptr->Tab_Index > 3000 ) )
    tui_msg( M_WHITE, "The tabular entry index is out of range" );

fscanf ( ddf_ffp, "%d", & ( msid_ptr->Txt_Index ) );
fscanf ( ddf_ffp, "%d", & ( msid_ptr->PBI_Indx ) );
fscanf ( ddf_ffp, "%d", &tmp );
fscanf ( ddf_ffp, "%d", & ( msid_ptr->hist_ind ) );
fscanf ( ddf_ffp, "%d", & ( msid_ptr->Limit_Ind ) );

if ( ( msid_ptr->Limit_Ind < -1 ) || ( msid_ptr->Limit_Ind > 3000 ) )
    tui_msg( M_WHITE, "The limit sense index is out of range" );

msid_ptr->ddd0_latch = NO;
msid_ptr->ddd1_latch = NO;
```

```
        msid_ptr++;
    }

/*
 *  Determine three highest frequency colors (2 Nominal, 1 Static, hopefully)
 */

    one = two = three = 0;

    for (i=0; i<129; i++) {
        if (color_count[i] > color_count[one]) {
            three = two;
            two = one;
            one = i;
        }
    }

    /* set GC color index to FGC index */

    GC_Index[one] = 0;
    GC_Index[two] = 1;
    GC_Index[three] = 2;

/*
 *  Skip ppl's
 */

    for ( i = 0; i < ppl_num; i++ )
        fscanf ( ddf_ffp, "%*47c" );

/*
 *  Initialize color count array
 */

    for (i=0; i<129; i++)
        color_count[i]=0;

/*
 *  Position pointer at the beginning of Limit structure and read into
 *  memory the limit structure.
 */

    Limit = ( struct limit_ent  * ) ( Msid + entry_num );
    limit_ptr = Limit;

    for ( i = 0; i < limit_num; i++ ) {
        fscanf ( ddf_ffp, "%d", & ( limit_ptr->Limt_Entr ) );
        fscanf ( ddf_ffp, "%hd", & ( limit_ptr->Limt_Flag ) );

        if ( ( limit_ptr->Limt_Flag < 0 ) || ( limit_ptr->Limt_Flag > 1 ) )
            tui_msg( M_WHITE, "The limit sense flag is out of range" );

        fscanf ( ddf_ffp, "%lf", & ( limit_ptr->Low_Limit ) );
        fscanf ( ddf_ffp, "%lf", & ( limit_ptr->Hi_Limit ) );
        fscanf ( ddf_ffp, "%d", & ( color ) );
        limit_ptr->Lo_Color = Pixels[color];

        if ( ( color < 0 ) || ( color > 128 ) )
            tui_msg( M_WHITE, "The low limit color is out of range" );
        else
            color_count[color]++;

        fscanf ( ddf_ffp, "%d", & ( color ) );
```

```
        limit_ptr->Hi_Color = Pixels[color];

        if ( ( color < 0 ) || ( color > 128 ) )
            tui_msg( M_WHITE, "The high limit color is out of range" );

        fscanf ( ddf_ffp, "%hd", & ( limit_ptr->Crit_Flag ) );

        if ( ( limit_ptr->Crit_Flag < 0 ) || ( limit_ptr->Crit_Flag > 1 ) )
            tui_msg(M_WHITE, "The critical limit check flag is out of range");

        fscanf ( ddf_ffp, "%lf", & ( limit_ptr->Crit_Low ) );
        fscanf ( ddf_ffp, "%lf", & ( limit_ptr->Crit_Hi ) );
        fscanf ( ddf_ffp, "%d", & ( color ) );
        limit_ptr->Cr_Lcolor = Pixels[color];

        if ( ( color < 0 ) || ( color > 128 ) )
            tui_msg(M_WHITE, "The critical low limit color is out of range");

        fscanf ( ddf_ffp, "%d", & ( color ) );
        limit_ptr->Cr_Hcolor = Pixels[color];

        if ( ( color < 0 ) || ( color > 128 ) )
            tui_msg(M_WHITE, "The critical high limit color is out of range");

        limit_ptr++;
    }

/*
 *  Determine two highest frequency colors (1 op high, 1 op low, hopefully)
 */

    one = two = 0;

    for (i=0; i<129; i++) {
        if (color_count[i] > color_count[one]) {
            two = one;
            one = i;
        }
    }

    /* set GC color index to FGC index */

    if (GC_Index[one] == -1)
        GC_Index[one] = 3;

    /* if no color assigned to two, then there is only 1 color for operational
       high and low */

    if (two > 0)
        if (GC_Index[two] == -1) {
            GC_Index[two] = 4;
        }

/*
 *  Initialize font count arrays
 */

    for (i=0; i<MAX_FONTS; i++)
        font_count[i]=0;

/*
 *  Position pointer at the beginning of Tabular structure and read into
 *  memory tabular structure.
 */
```

```
    Tab = ( struct tabular_ent  * ) ( Limit + limit_num );
    tab_ptr = Tab;

    for ( i = 0; i < tab_num; i++ ) {
        fscanf ( ddf_ffp, "%d", & ( tab_ptr->Tab_Entry ) );
        fscanf ( ddf_ffp, "%d", & ( tab_ptr->Data_Width ) );

        if ( ( tab_ptr->Data_Width < 0 ) || ( tab_ptr->Data_Width > 100 ) )
            tui_msg( M_WHITE, "The data field width is out of range" );

        fscanf ( ddf_ffp, "%hd", & ( tab_ptr->Dig_Right ) );

        if ( ( tab_ptr->Dig_Right < 0 ) || ( tab_ptr->Dig_Right > 100 ) )
            tui_msg(M_WHITE,
                        "The digits right of the decimal exceeds the limit");

        fscanf ( ddf_ffp, "%hd", & ( tab_ptr->Just_Flag ) );

        if ( ( tab_ptr->Just_Flag < 0 ) || ( tab_ptr->Just_Flag > 1 ) )
            tui_msg( M_WHITE, "The truncate/justification flag is incorrect" );

        fscanf ( ddf_ffp, "%f", &fx1);

        if ( ( fx1 < 0 ) || ( fx1 > 100 ) )
            tui_msg( M_WHITE, "The starting X coordinate is out of range" );

        fscanf ( ddf_ffp, "%f", &fy1);

        if ( ( fy1 < 0 ) || ( fy1 > 100 ) )
            tui_msg( M_WHITE, "The starting Y coordinate is out of range" );

        fscanf ( ddf_ffp, "%s", font_style );
        fscanf ( ddf_ffp, "%d", &horz_size );
        fscanf ( ddf_ffp, "%d", &vert_size );

        /* convert coordinates to X pixel coordinates */

        tab_ptr->X_XC = (int) (fx1 * factor_x);
        tab_ptr->Y_XC = (int) ((100.0 - fy1) * factor_y);

        tab_ptr->font_num =
                        font_num(disp_num, font_style, horz_size, vert_size);

/*
        font_count[tab_ptr->font_num]++;
*/

        tab_ptr->redraw_flag = NO;

        tab_ptr++;
    }

/*
 * Determine highest frequency font

    one = -1;

    for (i=0; i<MAX_FONTS; i++) {
        if (
    }

*/
```

```
/*
 *  Position pointer at the beginning of Template structure and read into
 *  memory template structure.
 */

    Tmplt = ( struct plot_tmplt * ) ( Tab + tab_num );
    tmplt_ptr = Tmplt;

    for ( i = 0; i < tmplt_num; i++ ) {

        /* read in a single plot (template) record */

        fscanf ( ddf_ffp, "%ld", & ( tmplt_ptr->tmplt_entr ) );
        fscanf ( ddf_ffp, "%hd", & ( tmplt_ptr->tmplt_type ) );
        fscanf ( ddf_ffp, "%f", &fx1);
        fscanf ( ddf_ffp, "%f", &fy2);
        fscanf ( ddf_ffp, "%f", &fx2);
        fscanf ( ddf_ffp, "%f", &fy1);
        fscanf ( ddf_ffp, "%s", ( tmplt_ptr->tmplt_nam ) );

/*
 *      Calculate the pixel coordinates of the upper left corner of the plot
 *      bounding box and the dimensions of the plot bouunding box for use
 *      in placing the scrolled window widget.
 */


        tmplt_ptr->bb_xul = (short) (fx1 * factor_x);
        tmplt_ptr->bb_yul = (short) ((100.0 - fy1) * factor_y);

        wwidth = fx2 - fx1;
        wheight = fy1 - fy2;

        tmplt_ptr->bb_width = (short) (wwidth * factor_x);
        tmplt_ptr->bb_height = (short) (wheight * factor_y);

/*
 *      Calculate the dimensions of the drawing area and the visible
 *      drawing area.
 */

/* RLK 10/12/90 Correct to properly adjust for the width of the scroll bars
                (remove these hard-coded constants) */

        tmplt_ptr->drw_width = tmplt_ptr->bb_width;
        tmplt_ptr->drw_height = tmplt_ptr->bb_height;

/* for scrolled window
        tmplt_ptr->drw_width = tmplt_ptr->bb_width - 15;
        tmplt_ptr->drw_height = tmplt_ptr->bb_height - 15;
*/

/* This code will double the size of the drawing area
        tmplt_ptr->drw_width = 2 * (tmplt_ptr->bb_width - 15);
        tmplt_ptr->drw_height = 2 * (tmplt_ptr->bb_height - 15);
*/

/*
 *      Calculate world coordinate transformation factors with respect
 *      to the pixel size of the drawing area.  Notes:  The 1.0 constant
 *      subtracted to prevent transformation factors from being calculated
 *      for a drawing area 1 pixel larger which could result from truncation.
 */
```

```
        tmplt_ptr->org_factor_x = tmplt_ptr->factor_x = (double)
                                  ((wwidth * factor_x - 1.0) / 100.0);
        tmplt_ptr->org_factor_y = tmplt_ptr->factor_y = (double)
                                  ((wheight * factor_y - 1.0) / 100.0);

/*
 *      Set zoom focus point offset values to zero.
 */

        tmplt_ptr->offset_x = 0;
        tmplt_ptr->offset_y = 0;

/* RLK 10/12/90 Correct to properly adjust for the width of the scroll bars
                (remove these hard-coded constants) */

/* for scrolled window
        tmplt_ptr->factor_x = (double)
                                  ((wwidth * factor_x - 15.0) / 100.0);
        tmplt_ptr->factor_y = (double)
                                  ((wheight * factor_y - 15.0) / 100.0);
*/

/* RLK 10/17/90 This code will adjust the transformation factors to
                utilize a doubled drawing area.
        tmplt_ptr->factor_x = (double)
                                  ((2.0 * (wwidth * factor_x - 15.0)) / 100.0);
        tmplt_ptr->factor_y = (double)
                                  ((2.0 * (wheight * factor_y - 15.0)) / 100.0);
*/

/*
 *      Increment the template record pointer
 */

        tmplt_ptr++;
    }

/*
 *  Position pointer at the beginning of History tab structure and read into
 *  memory htab structure.
 */

    Htab = ( struct hist_tab * ) ( Tmplt + tmplt_num );
    htab_ptr = Htab;

printf("  NUMBER OF HISTORY TABS IS %d\n",htab_num);

    for ( i = 0; i < htab_num; i++ ) {
        fscanf ( ddf_ffp, "%ld", &htab_ptr->htab_entr );
        fscanf ( ddf_ffp, "%d", &htab_ptr->time_cntr );
        fscanf ( ddf_ffp, "%s", temp );
        htab_ptr->llimit_flag = temp[0];
        fscanf ( ddf_ffp, "%s", temp );
        htab_ptr->ulimit_flag = temp[0];
        fscanf ( ddf_ffp, "%s", htab_ptr->file_name );
        htab_ptr->value = NULL;
        htab_ptr->next_ptr = htab_ptr + 1;

        htab_ptr++;
    }

    if ( htab_num > 0 )
        ( Htab + htab_num - 1 )->next_ptr = NULL;
```

```
/*
 *  Position pointer at the beginning of Mtext structure and read into
 *  memory the mtext structure.
 */

    Mtext = ( struct mtext_ent  * ) ( Htab + htab_num );
    mtext_ptr = Mtext;

    for ( i = 0; i < mltxt_num; i++ ) {
        fscanf ( ddf_ffp, "%d", & ( mtext_ptr->Mult_Entr ) );
        fscanf ( ddf_ffp, "%hd", & ( mtext_ptr->Num_Values ) );

        k = 0;
        getc ( ddf_ffp );

        do {
            local_text[k] = getc ( ddf_ffp );
            k++;
        } while ( k < 6 );

        strncpy ( mtext_ptr->Def_Text, local_text, 6 );

        text_ptr = ( struct val_txt * )
                        calloc(mtext_ptr->Num_Values, sizeof(struct val_txt));

        if ( text_ptr == NULL ) {
            tui_msg(M_YELLOW, "Error %d on calloc of text information ", errno);
            fclose ( ddf_ffp );
            return ( -1 );
        }

        mtext_ptr->text_ptr = text_ptr;

        for ( j = 0; j < mtext_ptr->Num_Values; j++ ) {
            fscanf ( ddf_ffp, "%d", & ( text_ptr->Value ) );
            k = 0;
            getc ( ddf_ffp );

            do {
                local_text[k] = getc ( ddf_ffp );
                k++;
            } while ( k < 6 );

            strncpy ( text_ptr->Text, local_text, 6 );
            text_ptr++;
        }

        mtext_ptr++;
    }

/*
 *  Position pointer at the beginning of Pbi structure and read into
 *  memory the Pbi structure.
 */

    Pbi = ( struct pbi_ent  * ) ( Mtext + mltxt_num );
    pbi_ptr = Pbi;

    for ( i = 0; i < pbi_num; i++ ) {
        fscanf ( ddf_ffp, "%*19c" );
        fscanf ( ddf_ffp, "%d", & ( pbi_ptr[i].ddd_indx ) );
        fscanf ( ddf_ffp, "%*25c" );
        fscanf ( ddf_ffp, "%d", & ( pbi_ptr[i].grph_indx ) );
```

```
        fscanf ( ddf_ffp, "%d", & ( pbi_ptr[i].grph_color ) );
        fscanf ( ddf_ffp, "%*29c" );
        fscanf ( ddf_ffp, "%d", & ( pbi_ptr[i].num_labels ) );

        pbi_label_ptr = ( struct label_indices  * )
                calloc(pbi_ptr[i].num_labels, sizeof(struct label_indices));

        pbi_ptr[i].grph_color += COLOR_OFFSET;

        if ( pbi_label_ptr == NULL ) {
            tui_msg( M_YELLOW, "Error %d on calloc of pbi", errno );
            fclose ( ddf_ffp );
            return ( -1 );
        }

        pbi_ptr[i].label_ptr = pbi_label_ptr;

    /*
     *      Read in the label indices
     */

        for ( j = 0; j < pbi_ptr[i].num_labels; j++ ) {
            fscanf ( ddf_ffp, "%hd", & ( pbi_label_ptr->index ) );
            pbi_label_ptr++;
        }

    /*
     *      Read in the length of the message
     */

        fscanf ( ddf_ffp, "%d", &mesg_len );
        fscanf ( ddf_ffp, "%*c", 1 );

    /*
     *      Skip past the message
     */

        for ( j = 0; j < mesg_len; j++ )
            fscanf ( ddf_ffp, "%*c", 1 );

    /*
     *      Read in the number of dependent MSIDs
     */

        fscanf ( ddf_ffp, "%d", &dep_msids );
        fscanf ( ddf_ffp, "%*c", 1 );

    /*
     *      Skip past all of the dependent MSIDs
     */

        if ( dep_msids > 0 ) {

            for ( j = 0; j < dep_msids; j++ ) {

                for ( k = 0; k < MSID_LENGTH; k++ )
                    fscanf ( ddf_ffp, "%*c", 1 );

                fscanf ( ddf_ffp, "%*c", 1 );
            }
        }

    /*
     *      Read the new line character
```

```
 */

        fscanf ( ddf_ffp, "%*c", 1 );

    } /* end reading PBIs */

/*
 *  Call read_fgr to read in all the foreground
 *  graphical records from the fg DDF file.
 */

    Fg_rec.graph_num = icon_num;
    Fg_rec.graph_rec = ( struct fgr_record  * ) ( Pbi + pbi_num );

    if (Fg_rec.graph_num > 0)
        error = read_fgr ( disp_num, ddf_ffp );

    if ( error == -1 ) {
        fclose ( ddf_ffp );
        return ( -1 );
    }

/*
 *  Position pointer at the beginning of ddd structure and read into
 *  memory ddd structure.
 */

    Ddd = ( struct ddd_ent  * ) ( Fg_rec.graph_rec + icon_num );
    ddd_ptr = Ddd;

    for ( i = 0; i < ddd_num; i++ ) {
        fscanf ( ddf_ffp, "%hd", & ( ddd_ptr->ddd_entr ) );
        fscanf ( ddf_ffp, "%d", &color);
        ddd_ptr->zero_val_cor = Pixels[color];
        fscanf ( ddf_ffp, "%d", &color);
        ddd_ptr->one_val_cor = Pixels[color];
        fscanf ( ddf_ffp, "%hd", & ( ddd_ptr->zero_locked ) );
        fscanf ( ddf_ffp, "%hd", & ( ddd_ptr->one_locked ) );
        fscanf ( ddf_ffp, "%d", & ( ddd_ptr->ddd_msids ) );

        ddd_ptr->ddd_app_ptr = (short *)
                                    calloc(ddd_ptr->ddd_msids, sizeof(short));

        if ( ddd_ptr->ddd_app_ptr == NULL ) {
            tui_msg( M_YELLOW, "Error %d on ddd append records calloc", errno );
            fclose ( ddf_ffp );
            return ( -1 );
        }

        loc_ddd_aptr = ddd_ptr->ddd_app_ptr;

        for ( j = 0; j < ddd_ptr->ddd_msids; j++ ) {
            fscanf ( ddf_ffp, "%hd", loc_ddd_aptr );
            loc_ddd_aptr++;
        }

        ddd_ptr++;
    }

/*
 *  Position pointer at the beginning of scale structure and read into
 *  memory scale structure.
 */
```

```
    Scale = ( struct scale_ent  * ) ( Ddd + ddd_num );
    scale_ptr = Scale;

    for ( i = 0; i < scale_num; i++ ) {
        fscanf ( ddf_ffp, "%hd", & ( scale_ptr->scale_entr ) );
        fscanf ( ddf_ffp, "%s", temp );
        scale_ptr->axis_type = temp[0];
        fscanf ( ddf_ffp, "%s", temp );
        scale_ptr->scale_type = temp[0];
        fscanf ( ddf_ffp, "%lf", & ( scale_ptr->low_scale ) );
        fscanf ( ddf_ffp, "%lf", & ( scale_ptr->high_scale ) );
        fscanf ( ddf_ffp, "%f", & ( scale_ptr->low_scale_x ) );
        fscanf ( ddf_ffp, "%f", & ( scale_ptr->low_scale_y ) );
        fscanf ( ddf_ffp, "%f", &high_scale_x );
        fscanf ( ddf_ffp, "%f", &high_scale_y );

        scale_ptr->scale_x_diff = high_scale_x - scale_ptr->low_scale_x;
        scale_ptr->scale_y_diff = high_scale_y - scale_ptr->low_scale_y;
        scale_ptr->msid_scale_range = scale_ptr->high_scale -
                                                scale_ptr->low_scale;

        scale_ptr++;
    }

/*
 *  Position pointer at the beginning of labels structure and read into
 *  memory labels structure.
 */

    Lab = ( struct label_ent * ) ( Scale + scale_num );
    label_ptr = Lab;

    for ( i = 0; i < label_num; i++ ) {
        fscanf ( ddf_ffp, "%d", & ( label_ptr->lbl_entr ) );
        fscanf ( ddf_ffp, "%s", font_style );
        fscanf ( ddf_ffp, "%d", &horz_size );
        fscanf ( ddf_ffp, "%d", &vert_size );

        label_ptr->font_num =
                        font_num(disp_num, font_style, horz_size, vert_size);

        fscanf ( ddf_ffp, "%d", &color);
        label_ptr->lbl_color = (long) Pixels[color];
        fscanf ( ddf_ffp, "%f", &fx1);
        fscanf ( ddf_ffp, "%f", &fy1);
        fscanf ( ddf_ffp, "%d", & ( label_ptr->label_len ) );

        label_ptr->strt_x_pos = (short) (fx1 * factor_x);
        label_ptr->strt_y_pos = (short) ((100.0 - fy1) * factor_y);

        label_ptr->label = calloc((label_ptr->label_len + 1), sizeof(char));

        if ( label_ptr->label == NULL ) {
            tui_msg(M_YELLOW,
                        "Error %d on calloc of label information ", errno);
            fclose ( ddf_ffp );
            return ( -1 );
        }

        loc_lbl_ptr = label_ptr->label;
        *loc_lbl_ptr = NULL;
        fscanf ( ddf_ffp, "%c", loc_lbl_ptr );

        for ( j = 0; j < label_ptr->label_len; j++ ) {
```

```
            fscanf ( ddf_ffp, "%c", loc_lbl_ptr );
            loc_lbl_ptr++;
        }

        *loc_lbl_ptr = NULL;

        label_ptr++;
    }

/*
 * Close DDF file
 */

    fclose ( ddf_ffp );

/*
 *  Call the history tab initialization routine.
 */

    if ( htab_num > 0 )
        ht_init(entry_num, htab_num);

/*
 *  Allocate memory for plot file entries
 */

    Plot_info_ptr = ( struct plot_ptrs  * )
                        calloc ( tmplt_num, sizeof ( struct plot_ptrs ) );

    if ( Plot_info_ptr == NULL ) {
        tui_msg( M_YELLOW, "Error %d on calloc of plot ptr struct ", errno );
        return ( -1 );
    }

/*
 *  Read the plot files.
 */

    Nbr_of_plots = tmplt_num;

    for ( i = 0; i < tmplt_num; i++ ) {
        plot_ptr = Plot_info_ptr + i;
        plot_ptr->plot_pos = Tmplt + i;
        plot_ptr->redraw_flag = NO;

        error = read_plt (disp_num, plot_ptr);

        if (error != 0)
            strcpy ( plot_ptr->plot_name, "NOFILEZZ" );
    }

    D(printf("END readfg\n"));

    return(total_nbr_fgrecords);
}
```

```
/*****************************************************************************
 * MODULE NAME: redraw.c
 *
 *  This function is called to redraw the display in case of an
 *  Expose event (i.e. if the window is exposed after being wholly or
 *  partially hidden).
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Richard Romeo - Ford Aerospace Corporation/Houston
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Ronnie Killough - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <X11/Xlib.h>
#include <constants.h>
#include <disp.h>
#include <DDdisp.h>
#include <DDplot.h>
#include <wex/wex.h>


extern struct dm_shmemory   *Dm_Address;          /* Ptr to DM shared memory.     */
extern struct plot_ptrs     *Plot_info_ptr;       /* Ptr thru plot ptr files.     */

extern short                Nbr_of_plots;         /* Number of plots to display.  */


int redraw(disp_num, ulx, uly, lrx, lry)

    short   disp_num;                         /* display number of the exposed display*/

    short   ulx, uly, lrx, lry;               /* coordinates of the redraw box        */
{
    int i;                                    /* loop counter                         */


    D(printf("START redraw\n"));
/*
 *  Redraw the background
 *
 *  If the expose rectangle is over 75% of the full display window,
 *  redraw the entire display.  Else redraw the exposed box only.
 */

/*
    expose_area = (lrx - ulx) * (lry - uly);
    window_area = Dm_Address->display[disp_num].size_x
                            * Dm_Address->display[disp_num].size_y;

    if (expose_area >= (window_area * 0.75))
        updtbg(disp_num);
    else
*/
        redwbg(disp_num, ulx, uly, lrx, lry);
```

```
/*
 *  Redraw foreground text and history tabs
 */

    redwfg(disp_num, ulx, uly, lrx, lry);

/*
 *  Redraw foreground graphics
 */

    /*rdwfgr(disp_num);*/

/*
 *  If any overlays are present, redraw them.
 */

    for ( i = 0; i < Nbr_of_plots; i++ )
        if ( (Plot_info_ptr + i)->ovr_flg == YES )
            draw_ovl ( (Plot_info_ptr + i) );

    D(printf("END redraw\n"));

    return (0);
}
```

```
/*****************************************************************************
 * MODULE NAME: redwbg
 *
 *  This function is called to redraw the background primitives in case of an
 *  Expose event on the display window.
 *
 *
 * DEVELOPMENT NOTES:
 *
 *  o   Calls to the intersect routine (and its subordinates) have been
 *      minimized by including the code directly in this routine to eliminate
 *      the overhead of repetitive function calls.
 *  o   Ellipses have not been tested.
 *  o   Curves have been implemented with line segments.  See comment
 *      accompanying code.
 *  o   Vector text has been implemented as normal text strings.
 *  o   Arcs have not been fully implemented and should not be used with
 *      this version of DM/DD.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Richard Romeo - Ford Aerospace Corporation/Houston
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Ronnie Killough - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <X11/Xlib.h>
#include <constants.h>
#include <disp.h>
#include <DDdisp.h>
#include <wex/EXmsg.h>


extern struct dm_shmemory   *Dm_Address;     /* ptr to DM shared memory      */
extern double   Horz_Size[MAX_FONTS];        /* horizontal fixed font sizes  */
extern double   Vert_Size[MAX_FONTS];        /* vertical fixed font sizes    */
extern struct bg_recs   Bg_Rec;              /* background records           */


redwbg(disp_num, ulx, uly, lrx, lry)

    short   disp_num;           /* display number of exposed display window */

    short   ulx, uly, lrx, lry; /* coordinates of the exposed area          */
{
    XPoint      points[100];        /* pointer to coord. point structure    */
    XPoint      point;              /* pointer to coord. point structure    */
    Display     *xdisplay;          /* ptr to X display struct for display  */
    Window      xwindow;            /* XID of display window                */
    GC          gc;                 /* graphics context in shared memory    */
    XGCValues   *gc_val;            /* ptr to gc values struct in shd mem   */
    Font    font;                   /* X Font ID of vector text font        */

    struct graph_record     *bg_graph_ptr;  /* ptr thru bg graphics records */
    struct rec_header       *bg_text_ptr;   /* ptr thru bg text records     */
```

```
    struct vtext_record     *vtext_ptr;     /* ptr thru vector text records */
    struct graph_pts        *poly_pts_ptr;  /* ptr thru polygon vertices    */
    struct graph_pts        *curve_pts_ptr; /* ptr thru array of curve pts  */
    struct line_record      *line_ptr;      /* ptr thru line records        */
    struct rectangle_record *rect_ptr;      /* ptr thru rectangle records   */
    struct polygon_record   *poly_ptr;      /* ptr thru polygon records     */
    struct circle_record    *circle_ptr;    /* ptr thru circle records      */
    struct arc_record       *arc_ptr;       /* ptr thru arc records         */
    struct ellipse_record   *ellipse_ptr;   /* ptr thru ellipse records     */
    struct curve_record     *curve_ptr;     /* ptr thru curve records       */

    float   x_scale, y_scale,       /* hold the aspect ratio information    */
            smajor, sminor,         /* axes for ellipse and circle          */
            radius,                 /* temp holder for radius               */
            angle1, angle2;         /* temp holder for graphical angle      */

    unsigned long   gc_mask;        /* mask for GC changes                  */

    int     i, j, k, w,             /* loop count variables                 */
            mminor, mmajor,         /* maj/min axes for cir/arcs/ellipses   */
            redraw_flag,            /* set if redraw of graphic needed      */
            pt_cnt,                 /* loop counter                         */
            x_min, x_max,           /* used in calculating bounding box     */
            y_min, y_max;           /*      intersections.                  */

/*
 *  Setup local display and window variables.
 */

    xdisplay = Dm_Address->xdisplay[disp_num];
    xwindow = Dm_Address->window[disp_num];
    gc = Dm_Address->gc[disp_num];
    gc_val = &Dm_Address->gc_val[disp_num];

/*
 *  Loop through bg graphical records in memory.
 *  If the graphic intersects the exposed area, redraw
 *  it on the display window.
 */

    i = 0;
    bg_graph_ptr = Bg_Rec.graph_rec;

    while (i < Bg_Rec.graph_num) {

        switch (bg_graph_ptr->graph_typ) {

        case LINE:

            line_ptr = (struct line_record *) bg_graph_ptr->graph_ptr;

            redraw_flag = NO;

/*
 *          Check if either endpoint of the line lies in the expose region
 */

            if (ulx <= line_ptr->point1_x && lrx >= line_ptr->point1_x
                    && uly <= line_ptr->point1_y && lry >= line_ptr->point1_y)
                redraw_flag = YES;

            else if (ulx <= line_ptr->point2_x && lrx >= line_ptr->point2_x
                    && uly <= line_ptr->point2_y && lry >= line_ptr->point2_y)
                redraw_flag = YES;
```

```
/*
 *          Check if line is horizontal and crosses the expose area
 */

            else if (line_ptr->point1_y == line_ptr->point2_y) {

                if (line_ptr->point2_x > line_ptr->point1_x) {
                    x_min = max(line_ptr->point1_x, ulx);
                    x_max = min(line_ptr->point2_x, lrx);
                } else {
                    x_min = max(line_ptr->point2_x, ulx);
                    x_max = min(line_ptr->point1_x, lrx);
                }

                if (x_min <= x_max && line_ptr->point1_y >= uly
                            && line_ptr->point1_y < lry)
                    redraw_flag = YES;

/*
 *          Check if line is vertical and crosses the expose area
 */

            } else if (line_ptr->point1_x == line_ptr->point2_x) {

                if (line_ptr->point2_y > line_ptr->point1_y) {
                    y_min = max(line_ptr->point1_y, uly);
                    y_max = min(line_ptr->point2_y, lry);
                } else {
                    y_min = max(line_ptr->point2_y, uly);
                    y_max = min(line_ptr->point1_y, lry);
                }

                if (y_min <= y_max && line_ptr->point1_x >= ulx
                            && line_ptr->point1_x < lrx)
                    redraw_flag = YES;

/*
 *          Line is not verticle nor horizontal.  Check if the
 *          line intersects any edge of the exposed area.
 */

            } else {
                points[0].x = line_ptr->point1_x;
                points[0].y = line_ptr->point1_y;
                points[1].x = line_ptr->point2_x;
                points[1].y = line_ptr->point2_y;

                redraw_flag = int_ln(ulx, uly, lrx, lry, points, 2);
            }

/*
 *          If the redraw flag has been set then
 *          set up the line attributes and draw it.
 */

            if (redraw_flag == YES) {

                /* set up the graphics context for this line */

                if (gc_mask = set_gc(xdisplay, gc, gc_val, line_ptr->graph_col,
                                     line_ptr->line_type, line_ptr->line_wdth,
                                     NO_CHANGE, NO_CHANGE, NO_CHANGE, NO_CHANGE))
                    XChangeGC(xdisplay, gc, gc_mask, gc_val);
```

```
            /* draw the line */

            XDrawLine(xdisplay, xwindow, gc,
                        line_ptr->point1_x, line_ptr->point1_y,
                        line_ptr->point2_x, line_ptr->point2_y);

        }

        break;


    case RECTANGLE:

        rect_ptr = (struct rectangle_record *) bg_graph_ptr->graph_ptr;

/*
 *          Check if the rectangle intersects the exposed area.
 */

        x_min = max(rect_ptr->ul_x, ulx);
        x_max = min(rect_ptr->ul_x + rect_ptr->width - 1, lrx);

        y_min = max(rect_ptr->ul_y, uly);
        y_max = min(rect_ptr->ul_y + rect_ptr->height - 1, lry);

/*
 *          If the rectangle intersects the redraw window
 *          then set up the rectangle attributes and draw it.
 */

        if ( x_min <= x_max && y_min <= y_max ) {

            /* set up the graphics context for this rectangle */

            if (gc_mask = set_gc(xdisplay, gc, gc_val, rect_ptr->graph_col,
                                 rect_ptr->line_type, rect_ptr->line_wdth,
                                 rect_ptr->pat_type, rect_ptr->pat_sizex,
                                 rect_ptr->pat_sizey, NO_CHANGE))
                XChangeGC(xdisplay, gc, gc_mask, gc_val);

            /* draw rectangle even if have fill pattern, since
                XFillRectangle doesn't draw the complete path */

            XDrawRectangle(xdisplay, xwindow, gc,
                           rect_ptr->ul_x, rect_ptr->ul_y,
                           rect_ptr->width, rect_ptr->height);

            /* if pattern type indicates a fill pattern, fill rectangle */

            if (rect_ptr->pat_type)
                XFillRectangle(xdisplay, xwindow, gc,
                               rect_ptr->ul_x, rect_ptr->ul_y,
                               rect_ptr->width, rect_ptr->height);

        }   /* end of if redraw */

        break;


    case POLYGON:

        poly_ptr = (struct polygon_record *) bg_graph_ptr->graph_ptr;
```

```
            poly_pts_ptr = poly_ptr->poly_pts_ptr;

    /*
     *      Set up the points array.
     */

            for (w = 0; w < poly_ptr->nmbr_pts; w++) {
                points[w].x = poly_pts_ptr->point_x;
                points[w].y = poly_pts_ptr->point_y;

                poly_pts_ptr++;
            }

    /*
     *      Check if any of the points are inside the exposed area.
     */

            pt_cnt = 0;
            redraw_flag = NO;

            while (redraw_flag == NO && pt_cnt < poly_ptr->nmbr_pts) {

                if (points[pt_cnt].x > ulx && points[pt_cnt].x < lrx
                        && points[pt_cnt].y > lry && points[pt_cnt].y < uly)
                    redraw_flag = YES;

                pt_cnt++;
            }

    /*
     *      Check if any of the line segments intersect
     *      the expose region.
     */

            if (!redraw_flag)
                redraw_flag = int_ln(ulx, uly, lrx, lry, points,
                                                    poly_ptr->nmbr_pts);

    /*
     *      If the polygon intersects the redraw window
     *      then set up the polygon attributes and draw it.
     */

            if (redraw_flag) {

                /* set up the graphics context for this polygon */

                if (gc_mask = set_gc(xdisplay, gc, gc_val, poly_ptr->graph_col,
                                poly_ptr->line_type, poly_ptr->line_wdth,
                                poly_ptr->pat_type, poly_ptr->pat_sizex,
                                poly_ptr->pat_sizey, NO_CHANGE))
                    XChangeGC(xdisplay, gc, gc_mask, gc_val);

    /* RLK 9/10/90  Assuming all points are relative to origin (depends on
                    how the Display Builder generates a polygon record.  This
                    polygon code was tested on hand-generated data files, so
                    this may not be a correct assumption */

                /* draw the polygon */

                XDrawLines(xdisplay, xwindow, gc, points,
                                poly_ptr->nmbr_pts, CoordModeOrigin);

    /* RLK 9/10/90  Assuming the polygon is non-complex so will use faster fill
```

```
                algorithm.  May be a bad assumption. */

                /* if pattern type indicates a fill pattern, fill polygon */

                if (poly_ptr->pat_type)
                    XFillPolygon(xdisplay, xwindow, gc, points,
                                 poly_ptr->nmbr_pts, Nonconvex,
                                 CoordModeOrigin);

            }   /* end of if redraw */

            break;


        case CIRCLE:

            /* setup local pointer to circle record */

            circle_ptr = (struct circle_record *) bg_graph_ptr->graph_ptr;

/*
 *          Check if circle's bounding box intersects the expose
 *          rectangle.  (This isn't the most accurate method of
 *          determining intersection of a circle and a rectangle,
 *          but it is fast.)
 */

            /* calculate the major and minor axes of the circle
                (width and height of the bounding box) */

/* RLK 9/10/90  May need to adjust the major/minor axes for ratio distortion
                using ratio of size of screen in millimeters/size in pixels */

            mmajor = mminor = (int) (2.0 * circle_ptr->radius);

            x_min = max(circle_ptr->bb_x, ulx);
            x_max = min(circle_ptr->bb_x + mmajor - 1, lrx);

            y_min = max(circle_ptr->bb_y, uly);
            y_max = min(circle_ptr->bb_y + mminor - 1, lry);

/*
 *          If the circle's bounding box intersects the redraw window
 *          then set up the circle attributes and draw it.
 */

            if ( x_min <= x_max && y_min <= y_max ) {

                /* setup graphics context for this circle */

                if (gc_mask = set_gc(xdisplay, gc, gc_val,
                                circle_ptr->graph_col, circle_ptr->line_type,
                                circle_ptr->line_wdth, circle_ptr->pat_type,
                                circle_ptr->pat_sizex, circle_ptr->pat_sizey,
                                NO_CHANGE))
                    XChangeGC(xdisplay, gc, gc_mask, gc_val);

                /* draw circle */

                XDrawArc(xdisplay, xwindow, gc,
                            circle_ptr->bb_x, circle_ptr->bb_y, mmajor, mminor,
                            START_CIRCLE, FULL_CIRCLE);

                /* if pattern type indicates a fill pattern, fill the circle */
```

```
            if (circle_ptr->pat_type)
                XFillArc(xdisplay, xwindow, gc, circle_ptr->bb_x,
                            circle_ptr->bb_y, mmajor, mminor,
                            START_CIRCLE, FULL_CIRCLE);

        }

        break;


    case ARC:

        /* setup local pointer to arc record */

        arc_ptr = (struct arc_record *) bg_graph_ptr->graph_ptr;

            /* setup graphics context for this arc */

            if (gc_mask = set_gc(xdisplay, gc, gc_val, arc_ptr->graph_col,
                                arc_ptr->line_type, arc_ptr->line_wdth,
                                arc_ptr->pat_type, arc_ptr->pat_sizex,
                                arc_ptr->pat_sizey, NO_CHANGE))
                XChangeGC(xdisplay, gc, gc_mask, gc_val);

/* RLK 10/22/90 The major and minor axes may need to be adjusted and the
                angles need to be converted from radians to degrees.  This
                should be done in readbg(). */

                /* draw arc */

                XDrawArc(xdisplay, xwindow, gc, arc_ptr->bb_x, arc_ptr->bb_y,
                            arc_ptr->maj_axis, arc_ptr->min_axis,
                            arc_ptr->angle1, arc_ptr->angle2);

                /* if pattern type indicates a fill pattern, fill arc */

/* RLK 9/11/90  Assuming arc fill mode is ArcChord...see gc assignment above */

                if (arc_ptr->pat_type)
                    XFillArc(xdisplay, xwindow, gc,
                                arc_ptr->bb_x, arc_ptr->bb_y,
                                arc_ptr->maj_axis, arc_ptr->min_axis,
                                arc_ptr->angle1, arc_ptr->angle2);

                break;

    case ELLIPSE:

        /* setup local pointer to ellipse record */

        ellipse_ptr = (struct ellipse_record *)bg_graph_ptr->graph_ptr;

/*
 *          Check if ellipse's bounding box intersects the expose
 *          rectangle.  (This isn't the most accurate method of
 *          determining intersection of an ellipse and a rectangle,
 *          but it is fast.)
 */

        x_min = max(ellipse_ptr->bb_x, ulx);
        x_max = min(ellipse_ptr->bb_x + ellipse_ptr->maj_axis - 1, lrx);

        y_min = max(ellipse_ptr->bb_y, uly);
```

```
            y_max = min(ellipse_ptr->bb_y + ellipse_ptr->min_axis - 1, lry);

/*
 *          If the ellipse's bounding box intersects the redraw window
 *          then set up the circle attributes and draw it.
 */

            if ( x_min <= x_max && y_min <= y_max ) {

                /* setup graphics context for this ellipse */

                if (gc_mask = set_gc(xdisplay, gc, gc_val,
                                ellipse_ptr->graph_col,
                                ellipse_ptr->line_type, ellipse_ptr->line_wdth,
                                ellipse_ptr->pat_type, ellipse_ptr->pat_sizex,
                                ellipse_ptr->pat_sizey, NO_CHANGE))
                    XChangeGC(xdisplay, gc, gc_mask, gc_val);

                /* draw ellipse */

                XDrawArc(xdisplay, xwindow, gc,
                            ellipse_ptr->bb_x, ellipse_ptr->bb_y,
                            ellipse_ptr->maj_axis, ellipse_ptr->min_axis,
                            START_CIRCLE, FULL_CIRCLE);

                /* if pattern type indicates a fill pattern, fill ellipse */

                if (ellipse_ptr->pat_type)
                    XFillArc(xdisplay, xwindow, gc,
                            ellipse_ptr->bb_x, ellipse_ptr->bb_y,
                            ellipse_ptr->maj_axis, ellipse_ptr->min_axis,
                            START_CIRCLE, FULL_CIRCLE);
            }

                break;

        case CURVE:

/* RLK 9/10/90  X10 had a command called XDraw which drew curves using a
                set of vertices and creating the curved surface with a
                spline algorithm.  X11 has no such command...will need to
                manually implement this algorithm. */

            /* setup local pointer to curve record */

            curve_ptr = (struct curve_record *) bg_graph_ptr->graph_ptr;

/*
 *          Set up the points array.
 */

            curve_pts_ptr = curve_ptr->curve_pts_ptr;

            for (k = 0; k < curve_ptr->nmbr_pts; k++) {
                points[k].x = curve_pts_ptr->point_x;
                points[k].y = curve_pts_ptr->point_y;
                curve_pts_ptr++;
            }

/*
 *          Check if any of the points are inside the exposed area.
 */

            pt_cnt = 0;
```

```
            redraw_flag = NO;

            while (redraw_flag == NO && pt_cnt < curve_ptr->nmbr_pts) {

                if (points[pt_cnt].x > ulx && points[pt_cnt].x < lrx
                        && points[pt_cnt].y > lry && points[pt_cnt].y < uly)
                    redraw_flag = YES;

                pt_cnt++;
            }

    /*
     *          Check if any of the line segments intersect
     *          the expose region.
     */

            if (!redraw_flag)
                redraw_flag = int_ln(ulx, uly, lrx, lry, points,
                                                    curve_ptr->nmbr_pts);

    /*
     *          If the polygon intersects the redraw window
     *          then set up the polygon attributes and draw it.
     */

            if (redraw_flag) {

                /* setup graphics context for this curve */

                if (gc_mask = set_gc(xdisplay, gc, gc_val, curve_ptr->graph_col,
                                curve_ptr->line_type, curve_ptr->line_wdth,
                                NO_CHANGE, NO_CHANGE, NO_CHANGE, NO_CHANGE))
                    XChangeGC(xdisplay, gc, gc_mask, gc_val);

                /* draw curve */

                XDrawLines(xdisplay, xwindow, gc, points,
                                curve_ptr->nmbr_pts, CoordModeOrigin);
            }

            break;

        case VECT_TXT:

            /* setup local pointer to vector text record */

            vtext_ptr = (struct vtext_record *) bg_graph_ptr->graph_ptr;

    /*
     *          Check if the text string bounding box intersects the exposed area.
     */

    /* RLK 10/23/90 Need to properly determine the text height & text extent
                and use in determining the text string bounding box */

            x_min = max(vtext_ptr->x_position, ulx);
            x_max = min(vtext_ptr->x_position + (vtext_ptr->char_len * 9), lrx);

            y_min = max(vtext_ptr->y_position - 15, uly);
            y_max = min(vtext_ptr->y_position, lry);

    /*
     *          If the text string bounding box intersects the exposed area
     *          then set up the text attributes and draw it.
```

```
 */

            if ( x_min <= x_max && y_min <= y_max ) {

/*
 *              Set text color and font
 */

                gc_mask = 0;

                if (gc_val->foreground != vtext_ptr->graph_col) {
                    gc_mask |= GCForeground;
                    gc_val->foreground = vtext_ptr->graph_col;
                }

                if (gc_val->font != vtext_ptr->font_num) {
                    gc_mask |= GCFont;
                    gc_val->font = vtext_ptr->font_num;
                }

                if (gc_mask)
                    XChangeGC(xdisplay, gc, gc_mask, gc_val);

                /* draw string to screen */

                XDrawString(xdisplay, xwindow, gc,
                        vtext_ptr->x_position, vtext_ptr->y_position,
                        vtext_ptr->record_item, vtext_ptr->char_len);
            }

        default:
            break;

        }   /* End of switch (graph type) */

        bg_graph_ptr++;

        i++;

    }   /* End of loop thru graphical records */

/*
 *  Loop thru text records and see if any lie in
 *  exposed area.  If so, redraw them.
 */

    bg_text_ptr = Bg_Rec.record;

    for (i = 0; i < Bg_Rec.char_num; i++) {

/*
 *      Check if the text string bounding box intersects the exposed area
 */

/* RLK 10/23/90 Need to properly determine the text height & text extent
                and use in determining the text string bounding box */

        x_min = max(bg_text_ptr->x_position, ulx);
        x_max = min(bg_text_ptr->x_position
                                + (bg_text_ptr->char_len * 9), lrx);

        y_min = max(bg_text_ptr->y_position - 15, uly);
        y_max = min(bg_text_ptr->y_position, lry);
```

```
/*
 *      If the text string bounding box intersects the exposed area
 *      then set up the text attributes and draw it.
 */

        if ( x_min <= x_max && y_min <= y_max ) {


/*
 *          Set text color and font
 */

            gc_mask = 0;

            if (gc_val->foreground != bg_text_ptr->color) {
                gc_mask |= GCForeground;
                gc_val->foreground = bg_text_ptr->color;
            }

            if (gc_val->font != bg_text_ptr->font_num) {
                gc_mask |= GCFont;
                gc_val->font = bg_text_ptr->font_num;
            }

            if (gc_mask)
                    XChangeGC(xdisplay, gc, gc_mask, gc_val);

            /* draw string on screen */

            XDrawString(xdisplay, xwindow, gc,
                    bg_text_ptr->x_position, bg_text_ptr->y_position,
                    bg_text_ptr->record_item, bg_text_ptr->char_len);

        }

        bg_text_ptr++;
    }
    return (0);
}
```

```
/*****************************************************************************
 * MODULE NAME: redwfg.c
 *
 *  This function refreshes the foreground tabular information on an
 *  expose event.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Tod Milam       - Ford Aerospace Corporation/Houston
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Ronnie Killough - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <wex/EXmsg.h>
#include <X11/Xlib.h>
#include <constants.h>
#include <DDdisp.h>


extern struct dm_shmemory   *Dm_Address;     /* ptr to DM shared memory       */
extern struct data_info     *Dh_Address;     /* ptr to DH shared memory       */
extern struct msid_ent      *Msid;           /* Msid entry table ptr          */
extern struct tabular_ent   *Tab;            /* Tabular entry table ptr       */
extern struct hist_tab      *Htab;           /* ptr to history tab info       */
extern double               Horz_Size[MAX_FONTS];
extern double               Vert_Size[MAX_FONTS];
extern unsigned char        Old_Data[60000];/* Old Data Array                 */


redwfg(disp_num, ulx, uly, lrx, lry)

    short   disp_num;               /* display # containing the exposed area */

    short   ulx, uly, lrx, lry;     /* coordinates of the exposed area       */

{
    struct msid_ent *msid_info;         /* ptr thru msid table               */
    struct tabular_ent *tab_info;       /* ptr thru tabular table            */
    struct hist_tab *htab;              /* ptr thru history tab array        */
    struct shm_decom    *decom_entry;   /* ptr to decom entry for msid       */
    struct shm_decom    *decom_buffer;  /* ptr to start of decom buffer      */

    long    status;                 /* status variable                       */

    int     index,                  /* index into the DH decom buffer        */
            i,                      /* loop counter                          */
            x_min, x_max,           /* used to det. expose area intersection */
            y_min, y_max;

/*
 *  Setup pointer to decom buffer in DH shared memory.
 *  Check if the Data Handler is updating the decom buffer.
 *  If so, exit this routine.
 */
```

```
        if (Dh_Address->need_decom == YES)
            return(0);

        Dh_Address->decom_in_use[disp_num] = YES;

        decom_buffer = (struct shm_decom *) ((char *) Dh_Address
                                                + Dh_Address->decom_buf);

    /*
     *  Loop through all of the msid records checking for redraw
     */

        msid_info = Msid;

        for (i = 0; i < Dh_Address->nbr_msids[disp_num]; i++) {

    /*
     *      If the msid has a tabular record then check
     *      if it lies within the expose area.
     */

            if (msid_info->Tab_Index > 0) {
                tab_info = Tab + msid_info->Tab_Index - 1;

    /*
     *          Check if the text string bounding box
     *          intersects the exposed area.
     */

    /* RLK 10/23/90 Need to properly determine the text height & text extent
                    and use in determining the text string bounding box */

                x_min = max(tab_info->X_XC, ulx);
                x_max = min(tab_info->X_XC + (tab_info->Data_Width * 9), lrx);

                y_min = max(tab_info->Y_XC - 15, uly);
                y_max = min(tab_info->Y_XC, lry);

    /*
     *          If the text string bounding box intersects the exposed area
     *          then set up the text attributes and draw it.
     */

                if ( x_min <= x_max && y_min <= y_max ) {

    /*
     *              Redraw the tabular entry if it is not a history tab field
     */

                    if (msid_info->hist_ind <= 0) {

    /*
     *                  Setup index into decom buffer.
     */

                        index = Dh_Address->msid_index[disp_num][i];

    /*
     *                  If the index is non-zero, extract the msid
     *                  value and display it to the screen.
     */

                        if (index >= 0) {
                            decom_entry = decom_buffer + index;
```

```
                        status = extract(&Old_Data[msid_info->data_ind],
                                                        decom_entry);
                        updtfg(disp_num, decom_entry, msid_info,
                                                        tab_info, status);
                    }

/*
 *                  Redraw the tabular entry if it is a history tab field
 */

                } else if (msid_info->hist_ind > 0) {

/*
 *                  Setup local pointer to the history tab entry.
 *                  If the history tab entry contains a value,
 *                  extract the status and value and update the display.
 */

                    htab = Htab + msid_info->hist_ind;

                    if (htab->value) {
                        status = extract(htab->value, &htab->decom_ent);
                        updtfg(disp_num, &htab->decom_ent, msid_info,
                                                    tab_info, status);
                    }
                }

            }   /* end of if redraw */

        }

        msid_info++;
    }
    return (0);
}
```

```
/*****************************************************************************
 * MODULE NAME: sel_disp.c
 *
 *  This function allows the user to select a display.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  S. Lee      - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <constants.h>
#include <disp.h>
#include <pf_key.h>
#include <wex/EXmsg.h>


extern Widget   Top;
extern struct file_info         *Disp_Info;     /* ptr to file information          */
extern struct pfkey_defs        Current_Com;    /* Current command structure        */

extern char                     *malloc();


int sel_disp ( )
{
    register int                i,
                                flag;

    static char                 **list;

    static int                  num_disps = 0;

    struct file_info            *d_info_ptr;

    char                        *ptr,
                                filename[50];

    D(printf("START sel_disp\n"));
/*
 *  Call the read_disp routine to read the directory of displays.
 */

    if ( num_disps == 0 ) {
        num_disps = read_disp ( );
        if ( num_disps == ERROR ) {
            num_disps = 0;
            return ( ERROR );
        }

/*
 *      Format the display names into a list of character strings.
 */
```

```
        d_info_ptr = Disp_Info;
        list = (char **)malloc ( num_disps * sizeof ( char * ) );
        for ( i = 0; i < num_disps; i++ ) {
            *(list+i) = malloc ( 100 );
            strcpy ( *(list+i), d_info_ptr->name );
            strcat ( *(list+i), d_info_ptr->desc );
            d_info_ptr++;
        }
        free ( (char *)Disp_Info );
    }

/*
 *  Present the list of names to the user and wait for a response.
 */

    flag = tui_get_list ( Top, list, num_disps, filename, "Select Display",
                          "Display Files", 0, -1, NULL, 0 );

/*
 *  If no display was selected, set the command to invalid.
 */

    if ( flag == 0 )
        Current_Com.func_no = INVALID;
    else {
        Current_Com.func_no = START_PDISPLAY;
        if ( strcmp ( filename, "DTE DISPLAY" ) == 0 )
            strcpy ( Current_Com.disp_name, filename );
        else {
            Current_Com.disp_name[8] = '\0';
            strncpy ( Current_Com.disp_name, filename, 8 );
            if ( ptr = index ( Current_Com.disp_name, ' ' ) )
                *ptr = '\0';
            else
                Current_Com.disp_name[8] = '\0';
            printf("%s\n",Current_Com.disp_name);
        }
    }

    D(printf("END sel_disp\n"));
    return ( flag );
}
```

```
/*****************************************************************************
 * MODULE NAME: set_cmap.c
 *
 *  This function sets the color map for a shell window. This is normally
 *  done after the shell is realized. This step is necessary to cause the
 *  correct colors to be displayed in the window when the pointer is moved
 *  into the window.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <X11/Intrinsic.h>
#include <wex/EXmsg.h>


extern Colormap Main_cmap;      /* The main colormap used by Display Manager          */


int set_cmap ( widget )

    Widget  widget;             /* The widget containing the window to set colormap on. */
{
    D(printf("START set_cmap\n"));
/*
 *  Assign main color map to widget window.
 */

    XSetWindowColormap ( XtDisplay ( widget ), XtWindow ( widget ), Main_cmap );

/*
 *  Normal return.
 */

    D(printf("END set_cmap\n"));
    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: set_gc.c
 *
 *  To set up the X graphics context (gc) and other attributes
 *  necessary to facilitate drawing the current graphic.
 *
 * Accepts:     a graphic context id (X GC)
 *              pointer to an XGCValues struct
 *              graph color (colormap index)
 *              line type (1-solid, 2-dashed, 3-dotted, 4-dotted/dashed)
 *              line width scale factor (float)
 *              pattern type (0-hollow, 1-solid, 2-hatch, 3..12-pattern)
 *              pattern size (width, height)
 * Returns:     a non-zero gc mask (if change needed in gc)
 *              or 0 (if no change needed)
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Ronnie Killough - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <X11/Xlib.h>
#include <constants.h>
#include <disp.h>
#include <DDdisp.h>
#include <wex/EXmsg.h>

unsigned long set_gc(xdisplay, gc, gc_val, graph_col, line_type, line_wdth,
                     pat_type, pat_sizex, pat_sizey, font)

    Display *xdisplay;              /* ptr to X display structure              */

    GC      gc;                     /* effective graphics context Xid          */

    XGCValues   *gc_val;            /* ptr to graphics context values structure */

    short   graph_col;              /* desired color                           */

    short   line_type;              /* desired line type                       */

    float   line_wdth;              /* desired line width                      */

    short   pat_type;               /* desired pattern type                    */

    short   pat_sizex, pat_sizey;   /* desired pattern size                    */

    Font    font;                   /* desired font                            */
{
    static int lline_type = -1;
    static int lpat_type = -1;

    static unsigned char dashed[2] = {6, 6};
    static unsigned char dotted[2] = {1, 6};
    static unsigned char dot_dashed[4] = {6, 4, 1, 4};

    unsigned long gc_mask = 0;      /* local gc mask                           */
    int dash_offset = 0;            /* local dash offset for XSetDashes        */

/*
```

```
 *  Check for foreground color change
 */

    if (graph_col != NO_CHANGE && gc_val->foreground != graph_col) {
        gc_val->foreground = graph_col;
        gc_mask |= GCForeground;
    }

/*
 *  Check for line style change
 */

    if (line_type != NO_CHANGE && lline_type != line_type) {

        lline_type = line_type;

        switch (line_type) {
            case 1:     /* solid line */
                gc_val->line_style = LineSolid;
                break;
            case 2:     /* dashed line */
                gc_val->line_style = LineOnOffDash;
                XSetDashes(xdisplay, gc, dash_offset, dashed, 2);
                break;
            case 3:     /* dotted line */
                gc_val->line_style = LineOnOffDash;
                XSetDashes(xdisplay, gc, dash_offset, dotted, 2);
                break;
            case 4:     /* dot/dash line */
                gc_val->line_style = LineOnOffDash;
                XSetDashes(xdisplay, gc, dash_offset, dot_dashed, 4);
                break;
            default:
                gc_val->line_style = LineSolid;
                break;
        }

        gc_mask |= GCLineStyle;
    }

/*
 *  Check for change in line width
 */

/* RLK 9/7/90   the line width in the bg file is a float intended for
                use as a GKS line_wdth scale factor.  Will just truncate
                to an X pixel width integer for now...prob OK */

    if (line_wdth != -1.0
            && gc_val->line_width != (int) line_wdth) {
        gc_val->line_width = (int) line_wdth;
        gc_mask |= GCLineWidth;
    }

/*
 *  Check for change in fill style

    if ((pat_type != NO_CHANGE) && (lpat_type != pat_type)) {

        lpat_type = pat_type;

        switch (pat_type) {
            case 0:     /* no fill
                break;
```

```
            case 1:
                gc_val->fill_style = FillSolid;
                gc_mask |= GCFillStyle;
                break;
            default:

                if (pat_type >= 3 && pat_type <= 12) {
                    gc_val->fill_style = FillTiled;
                    gc_mask |= GCFillStyle;

/* RLK 9/7/90   use pat_sizex & pat_sizey to get size of map somehow...maybe
                convert to a best-size.  Use pat_type as index into pixmap
                array, and assign this as the tile pixmap.  Will mess with
                this later *

                }

                /* else no action...assume no fill *
        }
    }
*/

/*
 *  Check for change in font
 */

    if ((font != NO_CHANGE) && (gc_val->font != font)) {
        gc_val->font = font;
        gc_mask |= GCFont;
    }

    return(gc_mask);
}
```

```
/*****************************************************************************
 * MODULE NAME: set_label.c
 *
 *  This function updates the label in a pushbutton widget. This is normally
 *  used for menu selections which are toggled back and forth between enabled
 *  and disabled states.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <X11/Intrinsic.h>
#include <Xm/PushB.h>
#include <constants.h>
#include <wex/EXmsg.h>


int set_label ( widget, label )

    Widget  widget;                 /* The pushbutton widget upon which to set the
                                     * label.
                                     */

    char    *label;                 /* The label to set on the pushbutton widget.
                                     */
{
    register int    i;

    XmString        string;

    Arg             args[10];

    D(printf("START set_label\n"));
/*
 *  Convert the label to a compound string and save in the argument list.
 */

    i = 0;
    string = XmStringLtoRCreate ( label, XmSTRING_DEFAULT_CHARSET );
    XtSetArg ( args[i], XmNlabelString, string ); i++;

/*
 *  Update the widget.
 */

    XtSetValues ( widget, args, i );

/*
 *  Free the compound string and return.
 */

    XmStringFree ( string );

    D(printf("END set_label\n"));
    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: set_timer.c
 *
 *  This function sets up a timer to cause the display to be updated.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <X11/Intrinsic.h>
#include <constants.h>
#include <disp.h>
#include <wex/EXmsg.h>


extern struct dm_shmemory   *Dm_Address;     /* ptr to DM shared memory      */

extern XtTimerCallbackProc  tmr_update();    /* time-out callback procedure  */


int set_timer (disp_num)

    short   disp_num;                        /* effective display number     */
{

/*
 *  Initialize the timeout.
 */

    XtAddTimeOut ( Dm_Address->display[disp_num].update_rate, tmr_update,
                   (caddr_t)disp_num );

    return (0);
}
```

```
/*****************************************************************************
 * MODULE NAME: shm_creat.c
 *
 *  The Shared Memory Create routine deletes and creates the Display Manager
 *  shared memory for this workstation.  Then the task is attached to the shared
 *  memory.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  K. Noonan       - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <fcntl.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <constants.h>
#include <disp.h>
#include <wex/EXmsg.h>


extern struct dm_shmemory   *Dm_Address;/* address of shared memory            */
extern int                  errno,      /* error return value                  */
                            Dm_Id;      /* Display Manager Shared memory ID    */


int shm_creat ( )
{
    D(printf("START shm_creat\n"));
/*
 * Create the shared memory segment.
 */

    if ( ( Dm_Id = shmget ( DM_SHM_KEY, sizeof ( struct dm_shmemory ),
                IPC_CREAT | 0666 ) ) == -1 ) {
        tui_msg ( M_YELLOW, "Error %d on  shared memory create", errno );
        return ( -1 );
    }

/*
 * Attach to the Display Manager shared memory.
 */

    if ( ( Dm_Address = ( struct dm_shmemory * ) shmat ( Dm_Id, 0, 0 ) ) == NULL ) {
        tui_msg ( M_YELLOW, "Error %d on  shared memory attach", errno );
        return ( -1 );
    }

    D(printf("END shm_creat\n"));
    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: sort_msid.c
 *
 *  After copying the LOCAL_TIME msids to the end of the list, this function
 *  uses a  bubble sort to put MSID records in alphabetical order by MSID name.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  K. Noonan       - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <constants.h>
#include <disp.h>
#include <wex/EXmsg.h>


int sort_msid ( msid_list, nbr_msids, nbr_recs )

    struct msid_record  *msid_list;     /* pointer to the msid list to sort        */

    short               nbr_msids,      /* nbr msid to sort                        */
                        nbr_recs;       /* nbr msid records to sort                */
{
    struct msid_record  temp_rec,       /* temporary holder of an msid record      */
                        tmsid_list[PLOT_MSIDS];

    short               swapped = YES,  /* flag to tell if records were swapped    */
                        time,           /* index for LOCAL_TIME records            */
                        msid;           /* index for msid records                  */

    int                 i = 0,          /* loop counter                            */
                        j,              /* loop counter                            */
                        bytes;          /* number of bytes in msid record to copy  */
    D(printf("START sort_msid\n"));
/*
 *  Get the size of the msid records.
 */

    bytes = sizeof ( struct msid_record );

/*
 *  Move the the new msid list into a temporary list.  All the actual msids
 *  will be at the top of the list and the LOCAL_TIME records will be at the
 *  bottom of the list.  Then move the newly sorted msid list back into the
 *  original list.
 */

    time = nbr_msids;
    msid = 0;
    for ( i = 0; i < nbr_recs; i++ ) {
        if ( ( strcmp ( ( msid_list + i ) ->msid_name, "LOCAL_TIME" ) ) == 0 ) {
            memcpy ( (char *)( &tmsid_list[time] ), (char *)( msid_list + i ), bytes );
            time++;
        } else {
```

```
            memcpy ( (char *)( &tmsid_list[msid] ), (char *)( msid_list + i ), bytes );
            msid++;
        }
    }

    memcpy ( ( char * ) msid_list, ( char * ) tmsid_list, bytes * nbr_recs );


/*
 *  Loop through all MSID records, until they are alphabetized ( no swaps )
 */

    i = 0;
    while ( ( i < nbr_msids ) && ( swapped == YES ) ) {

/*
 *      Riffle the last alphabetic MSID record to the bottom of the list
 */
        swapped = NO;
        for ( j = 1; j < nbr_msids - i; j++ ) {

/*
 *          Are these two MSID's out of order?
 */

            if ( strcmp ( (msid_list+j-1)->msid_name, (msid_list+j)->msid_name ) > 0 ) {

/*
 *              Yes. Swap the MSID Records.
 */

                swapped = YES;
                memcpy ( &temp_rec, msid_list + j - 1, bytes );
                memcpy ( msid_list + j - 1, msid_list + j, bytes );
                memcpy ( msid_list + j, &temp_rec, bytes );

            }
        }
        i++;
    }

    D(printf("END sort_msid\n"));
    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: stat_col.c
 *
 *  To determine which color a data item will be displayed in.
 *
 *  Accepts:        a status word
 *                  an msid table entry ptr
 *  Returns:        a status color
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Richard Romeo - Ford Aerospace Corporation/Houston
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Ronnie Killough - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <sys/types.h>
#include <sys/timeb.h>
#include <X11/Xlib.h>
#include <constants.h>
#include <DDdisp.h>
#include <wex/EXmsg.h>


extern struct limit_ent *Limit;     /* ptr to limit entries                  */


int stat_col(status, msid_info)

    long    status;                 /* status of data                        */
    struct msid_ent *msid_info;     /* ptr to msid entry                     */
{
    long    color,                  /* temp variable for color               */
            limit_ind;              /* local copy of msid limit index        */

    limit_ind = msid_info->Limit_Ind;

    if (status & DEAD_DATA)                         /* Dead Data */
        color = msid_info->Dead_Color;
    else if (status & MISSING_DATA)                 /* Missing */
        color = msid_info->Sta_Color;
    else if (status & STATIC_DATA)                  /* Static */
        color = msid_info->Sta_Color;
    else if (limit_ind > 0) {

        if (status & OFF_SCALE_HIGH)                /* Off high scale */
            color = (Limit + limit_ind - 1)->Cr_Hcolor;
        else if (status & OFF_SCALE_LOW)            /* Off low scale */
            color = (Limit + limit_ind - 1)->Cr_Lcolor;
        else if (status & CRITICAL_HIGH)            /* Critical high  */
            color = (Limit + limit_ind - 1)->Cr_Hcolor;
        else if (status & CRITICAL_LOW)             /* Critical low  */
            color = (Limit + limit_ind - 1)->Cr_Lcolor;
        else if (status & LIMIT_HIGH)               /* Out of limits high */
            color = (Limit + limit_ind - 1)->Hi_Color;
```

```
        else if (status & LIMIT_LOW)            /* Out of limits low  */
            color = (Limit + limit_ind - 1)->Lo_Color;
    }

/*
 *  If an unknown status occurs display a nominal color
 */

    else
        color = msid_info->Nom_Color;

    return (color);
}
```

```
/*****************************************************************************
 * MODULE NAME: tick_mk.c
 *
 *  This function draws a tick mark or grid line.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Richard Romeo - Ford Aerospace Corporation/Houston
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Ronnie Killough - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <X11/Xlib.h>
#include <constants.h>
#include <disp.h>
#include <DDplot.h>


extern struct dm_shmemory   *Dm_Address;    /* ptr to DM shared memory      */


void tick_mk(disp_num, plot_ptr, gc, xpos, ypos, length, xory)

    short   disp_num;                   /* effective display number         */
    struct plot_ptrs    *plot_ptr;      /* ptr to plot record               */
    GC      gc;                         /* Id of GC in DM shared memory     */
    short   xpos, ypos;                 /* start position of tick mark      */
    short   length;                     /* length of tick mark              */
    char    xory;                       /* X or Y axis tick mark            */

{
    XPoint points[2];                   /* endpoints of tick mark           */

/*
 *  Set the origin point
 */

    points[0].x = xpos;
    points[0].y = ypos;
/*
    points[0].x = (short) (xpos * plot_ptr->plot_pos->factor_x);
    points[0].y = (short) ((100.0 - ypos) * plot_ptr->plot_pos->factor_y);
*/

/*
    points[0].x = (short) (xpos * plot_ptr->plot_pos->factor_x)
                                        + plot_ptr->plot_pos->offset_x;
    points[0].y = (short) ((100.0 - ypos) * plot_ptr->plot_pos->factor_y)
                                        + plot_ptr->plot_pos->offset_y;
*/

/*
 *  Set the end point
 */

    if (xory == 'X') {
```

```
        points[1].x = points[0].x;
        points[1].y = ypos + length;
    } else if (xory == 'Y') {
        points[1].x = xpos + length;
        points[1].y = points[0].y;
    }

/*
 *  Draw the line
 */

    XDrawLine(Dm_Address->xdisplay[disp_num], XtWindow(plot_ptr->draw_win),
                    gc, points[0].x, points[0].y, points[1].x, points[1].y);

    return;
}
```

```
/*****************************************************************************
 * MODULE NAME: time_val.c
 *
 *  This function validates a time value.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  K. Noonan       - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <ctype.h>
#include <constants.h>
#include <wex/EXmsg.h>


int time_val ( char_str )

    char            *char_str;      /* integer or decimal character string */
{
    short            i,             /* loop counter                        */
                     nbr_colons,    /* number of colons in string          */
                     colon_chk,     /* check for a colon                   */
                     colon_char,    /* char nbr to check for a colon       */
                     valid,         /* set to YES if string is valid       */
                     length;        /* length of character string          */

    D(printf("START time_val\n"));
/*
 *  Get the length of the character string
 */

    length = strlen ( char_str );

/*
 *  Starting from the end, validate each character.  If a colon is not found
 *  at the first possible colon position, then the time input has to be in
 *  total seconds, so only digits are searched for.  If a colon is found at
 *  the first possible colon possible, then colons are searched for every
 *  third position until the third colon is found.  Then no more colons are
 *  needed.  The format possiblilities are seconds or ddd:hh:mm:ss, or a
 *  subset of the last format.
 */

    valid = YES;
    i = length - 1;
    nbr_colons = 0;
    colon_char = length - 3;
    colon_chk = YES;
    while ( i >= 0 && valid == YES ) {
        if ( i == colon_char && colon_chk == YES ) {
            if ( * ( char_str + i ) != ':' ) {
                if ( nbr_colons == 0 ) {
                    colon_chk = NO;
                    if ( ( isdigit ( * ( char_str + i ) ) ) == 0 )
```

```
                        valid = NO;
                    } else
                        valid = NO;
                } else {
                    colon_char -= 3;
                    i--;
                    nbr_colons++;
                    if ( nbr_colons >= 3 )
                        colon_chk = NO;
                }
            } else {
                if ( ( isdigit ( * ( char_str + i ) ) ) != 0 ) {
                    i--;
                } else {
                    valid = NO;
                }
            }
        }

        D(printf("END time_val\n"));
        return ( valid );
    }
```

```
/*****************************************************************************
 * MODULE NAME: tmr_update.c
 *
 *  This is the callback function which is called at specified intervals
 *  to update the dynamic text and graphics on the display.  If the
 *  display has not been paused by the user, this function also resets
 *  the timer to start the next update timer countdown.  If the display
 *  has not been paused, the timer is not reset to prevent further update
 *  to the display.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <X11/Xlib.h>
#include <X11/Intrinsic.h>
#include <constants.h>
#include <disp.h>
#include <wex/EXmsg.h>


extern  struct dm_shmemory  *Dm_Address;    /* Ptr to DM shared memory.          */


XtTimerCallbackProc tmr_update ( args, tid )

    char            *args;                  /* Contains the display # (disp_num).  */

    XtIntervalId    *tid;                   /* Timer id.                           */
{
    short           disp_num;               /* Display number to be updated.       */

/*
 *  Extract display number from arg list
 *  and call update.
 */

    disp_num = (int)args;

    update ( disp_num );

/*
 *  If display has not been paused, reset
 *  the timer for the next callback.
 */

    if ( Dm_Address->display[disp_num].disp_pause == NO )
        set_timer ( disp_num );

    return;
}
```

```
/*****************************************************************************
 * MODULE NAME: ui_init.c
 *
 *  This function initializes the main user interface.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Shell.h>
#include <Xm/Xm.h>
#include <Xm/RowColumn.h>
#include <user_inter.h>
#include <constants.h>
#include <disp.h>
#include <pf_key.h>
#include <wex/EXmsg.h>

extern Colormap             Main_cmap;

extern Widget               Verytop, Top, Pb_Alarm, Pb_Pbi, Pb_Log, Pb_Log_A, Pb_Msg,
                            Pb_Pf;

extern struct dm_shmemory   *Dm_Address;
extern struct pfkey_defs    Act_Pfkeys[],
                            Current_Com;

extern short                Disp_Num,
                            Msg_Popup_Flag;

extern char                 *Func_Desc[];


int ui_init ( argc, argv )

    int     argc;

    char    **argv;
{
    register int            i, n, len_accel;

    static char             accel[20] = "     <Key>F";

    typedef struct _iv {
        String  disp;
    } IV_REC, *IV;
    static IV_REC   iv;

    static XtResource resources[] = {
        { "df", "Df", XtRString, sizeof ( String ), XtOffset ( IV, disp ),
          XtRString, "" }
    };

    static XrmOptionDescRec options[] = {
        { "-df", "Df", XrmoptionSepArg, NULL }
```

```
    };

    Widget                  mb_main,
                            mp_file, mp_disp, mp_hist, mp_util, mp_limits,
                            mp_plot, mp_zoom, mp_keys, mp_help,
                            widget;

    Display                 *display;

    Arg                     args[10];

    XtCallbackProc          cb_help(),
                            cb_cmd ();

    XColor                  color;

    char                    name[100],
                            **old_font_paths;

    int                     flag,
                            screen,
                            x,
                            num_paths;

    unsigned long           planes[MAX_COLORS],
                            pixels[MAX_COLORS];

    D(printf("START ui_init\n"));

/*
 * Initialize the top level widget.
 */

    Verytop = XtInitialize ( argv[0], "Display_Manager", options, XtNumber ( options ),
                             &argc, argv );

/*
 *  Retrieve any application-specific resources.
 */

    XtGetApplicationResources ( Verytop, &iv, resources, XtNumber ( resources ), NULL, 0 )
;

/*
 *  Save the display pointer and name.
 */

    display = Dm_Address->xdisplay[Disp_Num] = XtDisplay ( Verytop );
    strcpy ( Dm_Address->display_name[Disp_Num], XDisplayString ( display ) );
    screen = DefaultScreen ( display );

/*
 *  Get search path for X fonts.
 */

/*
#ifdef DEBUG
*/

    old_font_paths = XGetFontPath(display, &num_paths);

    for (x=0; x<num_paths; x++)
        printf("%s\n", *(old_font_paths + x));
```

```
/*
#endif
*/

/*
 *  Create the main color map. The main color map is used by Termap to init-
 *  ialize widget colors. Its contents are copied into the first few pixels
 *  of each image color map.
 */

    Main_cmap = XCreateColormap ( display, DefaultRootWindow ( display ),
                                  DefaultVisual ( display, screen ), AllocNone );

/*
 *  The Display Manager will work best if the window manager is run in monochrome mode.
 *  In this case, it will use a black and a white color in the first two cells
 *  of the color map. Therefore allocate a black and a white color in the first
 *  two cells in the positions normally used on the systems.
 */

    color.flags = DoRed | DoGreen | DoBlue;

#ifdef SUN
    color.red = color.green = color.blue = 0xffff;
    XAllocColor ( display, Main_cmap, &color );

    color.red = color.green = color.blue = 0x0000;
    XAllocColor ( display, Main_cmap, &color );
#else
    color.red = color.green = color.blue = 0x0000;
    XAllocColor ( display, Main_cmap, &color );

    color.red = color.green = color.blue = 0xffff;
    XAllocColor ( display, Main_cmap, &color );
#endif

/*
 *  Create a new shell widget. This is necessary because you cannot set
 *  the color map except upon creation and you can't specify arguments for
 *  the top level widget.
 */

    i = 0;
    XtSetArg ( args[i], XmNcolormap, Main_cmap ); i++;
    Top = XtAppCreateShell ( "Display Manager Control Panel", "Display_Manager",
                applicationShellWidgetClass, display, args, i );

/*
 * Create the menu bar, and the form which will contain main fields.
 */

    i = 0;
    XtManageChild ( mb_main = XmCreateMenuBar ( Top, "menubar", args, i ) );

/*
 * Create pulldown menu for File commands.
 */

    i = 0;
    mp_file = XmCreatePulldownMenu ( mb_main, "mp_file",        args, i );
              tui_create_cascade   ( mb_main, "File", mp_file, args, i );

    Pb_Msg =
    tui_create_pushbutton ( mp_file, "Enable Message",  cb_cmd, MSG_ON,          args, i );
```

```
    tui_create_pushbutton ( mp_file, "Set Flight/Data", cb_cmd, SET_FLIGHT,     args, i );
    tui_create_pushbutton ( mp_file, "Screen Dump",     cb_cmd, SCRN_DUMP,      args, i );
    tui_create_pushbutton ( mp_file, "Edit Colors",     cb_cmd, EDIT_COLORS,    args, i )
    tui_create_pushbutton ( mp_file, "Exit",            cb_cmd, HALT_DISPLAY,   args, i ),

/*
 * Create pulldown menu for Display commands.
 */

    i = 0;
    mp_disp = XmCreatePulldownMenu ( mb_main, "mp_disp",           args, i );
              tui_create_cascade   ( mb_main, "Display", mp_disp, args, i );

    i = 0;
    tui_create_pushbutton ( mp_disp, "Select Display",  cb_cmd, START_DISPLAY,  args, i );
    tui_create_pushbutton ( mp_disp, "Remove Display",  cb_cmd, CLEAR_DISPLAY,  args, i );
    Pb_Pf =
    tui_create_pushbutton ( mp_disp, "Freeze Display",  cb_cmd, FREEZE_DISPLAY, args, i );


/*
 * Create pulldown menu for Utilities commands.
 */

    i = 0;
    mp_util = XmCreatePulldownMenu ( mb_main, "mp_util",           args, i );
              tui_create_cascade   ( mb_main, "Utilities", mp_util, args, i );

    i = 0;
    tui_create_pushbutton ( mp_util, "Change Update Rate", cb_cmd, UPD_RATE,     args, i );
    tui_create_pushbutton ( mp_util, "Unlatch DDD MSID",   cb_cmd, DDD_UNLATCH, args, i );
    tui_create_pushbutton ( mp_util, "Unlatch ALL DDD's",  cb_cmd, DDD_UNL_ALL, args, i )
    tui_create_pushbutton ( mp_util, "Change GDR",         cb_cmd, GDR_CHG,     args, i );

/*
 *  Create the commands which change state based on enable/disable.
 */

    Pb_Alarm = tui_create_pushbutton ( mp_util, "Enable Alarms",      cb_cmd,
                                       POS_ALARM,         args, i );
    Pb_Pbi   = tui_create_pushbutton ( mp_util, "Enable PBIs",        cb_cmd,
                                       PBI_ENABLE,        args, i );
    Pb_Log   = tui_create_pushbutton ( mp_util, "Enable Logging",     cb_cmd,
                                       LOGENABLE_DISPLAY, args, i );
    Pb_Log_A = tui_create_pushbutton ( mp_util, "Enable All Logging", cb_cmd,
                                       LOGENABLE_ALL,     args, i );

/*
 * Create pulldown for History Table commands.
 */

    i = 0;
    mp_hist = XmCreatePulldownMenu ( mb_main, "mp_hist",               args, i );
              tui_create_cascade   ( mb_main, "Hist/Table", mp_hist, args, i );

    i = 0;
    tui_create_pushbutton ( mp_hist, "History Tables", cb_cmd, HIST_TAB, args, i );

/*
 * Create pulldown for Limits commands.
 */

    i = 0;
    mp_limits = XmCreatePulldownMenu ( mb_main, "mp_limits",         args, i );
```

```
                tui_create_cascade   ( mb_main, "Limits", mp_limits, args, i );

    i = 0;
    tui_create_pushbutton ( mp_limits, "List Limits",   cb_cmd, LIM_LIST, args, i);
    tui_create_pushbutton ( mp_limits, "Change Limits", cb_cmd, LIM_MENU, args, i);

/*
 * Create pulldown for Plot commands.
 */

    i = 0;
    mp_plot = XmCreatePulldownMenu ( mb_main, "mp_plot",        args, i );
              tui_create_cascade   ( mb_main, "Plots", mp_plot, args, i );

    i = 0;
    tui_create_pushbutton ( mp_plot, "List Plots",             cb_cmd, PLOT_LIST,   args, i
);
    tui_create_pushbutton ( mp_plot, "Display Overlay",        cb_cmd, PLOT_OVRLAY, args, i
);
    tui_create_pushbutton ( mp_plot, "Save Overlay",           cb_cmd, SAVE_OVRLAY, args, i
);
    tui_create_pushbutton ( mp_plot, "Define Universal Plot", cb_cmd, PLOT_UNV,    args, i
);

/*
 * Create pulldown for zoom commands.
 */

    i = 0;
    mp_zoom = XmCreatePulldownMenu ( mb_main, "mp_zoom",       args, i );
              tui_create_cascade   ( mb_main, "Zoom", mp_zoom, args, i );

    i = 0;
    tui_create_pushbutton ( mp_zoom, "Zoom",                cb_cmd, ZOOM_DIS, args, i );
    tui_create_pushbutton ( mp_zoom, "Reset Zoom",          cb_cmd, ZOOM_RES, args, i );
    tui_create_pushbutton ( mp_zoom, "Change Zoom Factor", cb_cmd, ZOOM_FAC, args, i );

/*
 *  Create a dummy pulldown for the function keys. This pulldown is never managed, but
 *  provides an easy way to associate function key accelerators with commands.
 */

    i = 0;
    mp_keys = XmCreatePulldownMenu ( mb_main, "mp_keys",       args, i );
              tui_create_cascade   ( mb_main, "Keys", mp_keys, args, i );

/*
 *  Create the actual commands which correspond to the function keys. Note that the
 *  code sets the accelerator to either normal or shifted for the two sets of function
 *  keys.
 */

    len_accel = strlen ( accel );

    for ( n = 0; n < PFKEY_COUNT; n++ ) {
        flag = ( Act_Pfkeys[n].valid_flag == 0 && Act_Pfkeys[n].defined );

/*
 *      Build the name of the function key. This will appear on the menu and serve as
 *      the "Show PF Keys" Function.
 */

        sprintf ( name, "%5s F%02d - %s",
            ( n < PFKEY_COUNT/2 ) ? "Shift"                          : " Ctrl",
```

```
            ( n<PFKEY_COUNT/2    ) ? n+1                                    : n-PFKEY_COUNT/2 + 1
,
            ( flag               ) ? Func_Desc[Act_Pfkeys[n].func_no] : " " );

/*
 *      If the function key is defined and valid, build the accelerator string and
 *      save as an argument.
 */

        if ( flag ) {
            sprintf ( &accel[len_accel], "%d",
                (n<PFKEY_COUNT/2) ? n+1 : n-PFKEY_COUNT/2 + 1 );
            if ( n < PFKEY_COUNT/2 )
                strncpy ( accel, "Shift", 5 );
            else
                strncpy ( accel, " Ctrl", 5 );
            XtSetArg ( args[0], XmNaccelerator, accel );

        }

/*
 *      Create the widget.
 */

        tui_create_pushbutton ( mp_keys, name, ( flag ) ? cb_cmd : NULL, -n, args, 1 );
    }

/*
 *  Initialize default menu label states.
 */

    Msg_Popup_Flag = ON;
    tui_msg_control ( Msg_Popup_Flag );

    init_label ( );

/*
 * Create pulldown for Help.
 */

    i = 0;
    mp_help = XmCreatePulldownMenu ( mb_main, "",                args, i );
    widget  = tui_create_cascade   ( mb_main, "Help", mp_help, args, i );
    XtSetArg ( args[0], XmNmenuHelpWidget, widget );
    XtSetValues ( mb_main, args, 1 );

    tui_create_pushbutton ( mp_help, "Enable/Disable Messages",     cb_help,  0, args, i )
;
    tui_create_pushbutton ( mp_help, "Set Flight ID/Datatype",      cb_help,  1, args, i )
;
    tui_create_pushbutton ( mp_help, "Screen Dump",                 cb_help,  2, args, i )
;
    tui_create_pushbutton ( mp_help, "Edit Colors",                 cb_help,  3, args, i )
;
    tui_create_pushbutton ( mp_help, "Exit",                        cb_help,  4, args, i )
;
    tui_create_pushbutton ( mp_help, "Select Display",              cb_help,  5, args, i )
;
    tui_create_pushbutton ( mp_help, "Remove Display",              cb_help,  6, args, i
;
    tui_create_pushbutton ( mp_help, "Freeze Display",              cb_help,  7, args, i )
;
    tui_create_pushbutton ( mp_help, "Change Update Rate",          cb_help,  8, args, i )
;
```

```
    tui_create_pushbutton ( mp_help, "Unlatch DDD MSID",              cb_help,  9, args, i )
;
    tui_create_pushbutton ( mp_help, "Unlatch All DDD's",             cb_help, 10, args, i )
;
    tui_create_pushbutton ( mp_help, "Change GDR",                    cb_help, 11, args, i )
;
    tui_create_pushbutton ( mp_help, "History Tables",                cb_help, -1, args, i )
;
    tui_create_pushbutton ( mp_help, "Enable/Disable Alarms",         cb_help, 12, args, i )
;
    tui_create_pushbutton ( mp_help, "Enable/Disable PBI's",          cb_help, 13, args, i )
;
    tui_create_pushbutton ( mp_help, "Enable/Disable Logging",        cb_help, 14, args, i )
;
    tui_create_pushbutton ( mp_help, "Enable/Disable All Logging",    cb_help, 15, args, i )
;
    tui_create_pushbutton ( mp_help, "List Limits",                   cb_help, 16, args, i )
;
    tui_create_pushbutton ( mp_help, "Change Limits",                 cb_help, 17, args, i )
;
    tui_create_pushbutton ( mp_help, "List Plots",                    cb_help, 18, args, i )
;
    tui_create_pushbutton ( mp_help, "Display Overlay",               cb_help, 19, args, i )
;
    tui_create_pushbutton ( mp_help, "Save Overlay",                  cb_help, 20, args, i )
;
    tui_create_pushbutton ( mp_help, "Define Universal Plot",         cb_help, 21, args, i )
;
    tui_create_pushbutton ( mp_help, "Zoom",                          cb_help, 22, args, i )
;
    tui_create_pushbutton ( mp_help, "Reset Zoom Factor",             cb_help, 23, args, i )
;
    tui_create_pushbutton ( mp_help, "Set Zoom Factor",               cb_help, 24, args, i )
;
    tui_create_pushbutton ( mp_help, "Show PF Keys",                  cb_help, 25, args, i )
;

/*
 *  Allocate additional colors which will be required by popups created later in the
 *  application.
 */

    XAllocNamedColor ( display, Main_cmap, "lightblue", &color, &color );
    XAllocNamedColor ( display, Main_cmap, "skyblue",   &color, &color );

/*
 *  Allocate the remaining number of color cells which were not taken by the
 *  Motif colors. Note that the number of colors used by Motif is not constant
 *  and color matching takes place, so it is not safe to simply allocate
 *  MAX_COLORS-NUM_MOTIF_COLORS, because this might leave a few colors at the
 *  end of the color map unallocated. This would result in an X error when a
 *  later attempt to perform an XStoreColor is attempted.
 */

    for ( i = 0; i < NUM_MOTIF_COLORS; i++ )
        if ( XAllocColorCells ( display, Main_cmap, True, planes, 0,
                                pixels, MAX_COLORS-i ) )
            break;

    if ( i == NUM_MOTIF_COLORS ) {
        XFreeColormap ( display, Main_cmap  );
        tui_msg ( M_YELLOW, "Could not allocate colors in colormap" );
        return ( -1 );
    }
```

```
/*
 *  Initialize the colors used by the display functions.
 */

    colors ( );

/*
 *  Realize the top level widget.
 */

    XtRealizeWidget ( Top );

    XSetWindowColormap ( display, XtWindow ( Top ), Main_cmap );

/*
 * Check to see if a display name was input as an intialization argument.
 */

    if ( *iv.disp ) {
        strcpy ( Current_Com.disp_name, iv.disp );
        if ( val_fn ( Current_Com.disp_name, YES ) ) {
            Current_Com.func_no = START_PDISPLAY;
            command ( NO );
        } else
            tui_msg ( M_YELLOW, "Display %s not started", Current_Com.disp_name );
    }

    D(printf("END ui_init\n"));
    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: unlatch.c
 *
 *  This routine unlatches one or all msids depending on the unlatch action
 *  flag in shared memory.  If the action is all msids, then all the latch
 *  flags in the msid records are set to NO.  If the action is for an msid,
 *  then the msid is searched for in the msid records and the latch flag for
 *  all occurrances of that msid are set to NO.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  C. Davis        - Ford Aerospace Corporation
 *
 *
 *  MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <X11/Xlib.h>
#include <stdio.h>
#include <constants.h>
#include <disp.h>
#include <DDdisp.h>
#include <wex/EXmsg.h>


extern struct msid_ent                  *Msid;          /* msid structure pointer       */
extern struct dm_shmemory               *Dm_Address;    /* DM structure pointer         */
extern struct fg_file_header            *Ffile;
extern short                            Disp_Num;       /* display number               */


int unlatch ( )
{
    struct msid_ent *msid_ptr;                          /* msid local record pointer    */
    int             i,                                  /* loop count variable          */
                    retval;                             /* return value from strcmp     */
    short           match,                              /* YES, if a match              */
                    finished;                           /* loop control variable        */

    D(printf("START unlatch\n"));
/*
 *  If the action flag is to unlatch all msid's, then set the ddd latch flag
 *  to NO in each msid record.
 */

    msid_ptr = Msid;

    if ( Dm_Address->display[Disp_Num].action == ALL ) {
        for ( i = 0; i < Ffile->Entry_Num; i++ ) {
            msid_ptr->ddd0_latch = NO;
            msid_ptr->ddd1_latch = NO;
            msid_ptr++;
        }
        tui_msg ( M_BLUE, "Msids unlatched" );

/*
 *  The action flag is to unlatch all occurrances of the MSID on the display.
```

```
 *  Search the msid records for a match on the msid and the source. If a
 *  match is found, then clear the latch flag.  Search the list until the
 *  msid name is less alphabetically, then the msid in the msid records list.
 */

    } else {
        i = 0;
        finished = match = NO;

        while ( ( i < Ffile->Entry_Num ) && ( finished == NO ) ) {
            retval = strcmp ( msid_ptr->MSID, Dm_Address->display[Disp_Num].msid_name );
            if ( retval == 0 ) {
                if ( strcmp ( msid_ptr->Data_Src,
                              Dm_Address->display[Disp_Num].src ) == 0 ) {
                    match = YES;
                    msid_ptr->ddd0_latch = msid_ptr->ddd1_latch = NO;
                }
            } else if ( retval > 0 )
                finished = YES;

            msid_ptr++;
            i++;
        }

/*
 *  If no match was found in the msid records list, then advise.
 */

        if ( match == NO )
            tui_msg ( M_YELLOW,"Msid %s source %s is not on this display",
                        Dm_Address->display[Disp_Num].msid_name,
                        Dm_Address->display[Disp_Num].src );
    }
    D(printf("START unlatch\n"));
    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: unv_plot.c
 *
 *  This function allows the user to define a universal plot.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  K. Noonan       - Ford Aerospace Corporation
 *
 *
 * INTERNAL FUNCTIONS:
 *
 *      o   unv_menu        -   Displays the universal plot menu.
 *
 *      o   cb_unv          -   Processes all callbacks from the menu.
 *
 *      o   process_plot    -   Processes entry of a plot file name.
 *
 *      o   display_xy      -   Processes selection of the X/Y button.
 *
 *      o   display_msid    -   Processes selection of the MSID button.
 *
 *      o   save_xy         -   Saves X/Y values.
 *
 *      o   save_msid       -   Saves MSID values.
 *
 *      o   process_ok      -   Processes selection of the OK button.
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <X11/Intrinsic.h>
#include <X11/Shell.h>
#include <Xm/Xm.h>
#include <Xm/Text.h>
#include <user_inter.h>
#include <constants.h>
#include <disp.h>
#include <pf_key.h>
#include <wex/EXmsg.h>


struct disp_info            *display;
static struct msid_record   msid_rec[PLOT_MSIDS];

static Widget       f_msid, t_plot, t_xy_id, t_xlow, t_xhigh, t_ylow, t_yhigh,
                    t_msid_id, t_msid, t_src, r_sample, s_axis_no, r_xory,
                    t_msid_p, t_src_p, r_sample_p, s_axis_no_p;

static char         *samples[] = { "A", "L" },
                    *x_and_y[] = { "X", "Y" },
                    *list    [] = { "1", "2", "3", "4", "5", "6", "7", "8", "9", "10" },
```

```
                plot_name    [DNAME_LEN + 5],    /* plot file name                */
                unv_name     [DNAME_LEN + 5],    /* universal plot file name      */
                sav_unv_name[DNAME_LEN + 5],
                low_scale    [TOTAL_AXES][16],   /* low scale values              */
                high_scale   [TOTAL_AXES][16],   /* high scale values             */
                scale_type   [TOTAL_AXES][2];    /* type of scale Time or Number  */

static int      flag,
                version,
                ptr_axis,
                ptr_msid,
                plot_indx,
                time_x,
                time_y,
                nbr_x_axis,
                nbr_y_axis,
                nbr_y,
                nbr_x,
                active,
                nbr_msid_rec,                    /* number of msid records        */
                nbr_plot_msid,                   /* number of msids to plot       */
                univ_file;                       /* YES, if universal file exists */

FILE            *fopen ( ),
                *fp = 0;

extern Widget                Top;
extern struct dm_shmemory    *Dm_Address;        /* display manager shm           */
extern short                 Disp_Num;           /* display manager number        */
extern int                   errno;
extern char                  Disp_Path[DNAME_LEN],   /* default display path      *
                             Plot_Path[DNAME_LEN];   /* default plot path         */


int unv_plot ( )
{

    D(printf("START unv_plot\n"));

/*
 *  Save display pointer.
 */

    display = &Dm_Address->display[Disp_Num];

    display->disp_pause = YES;

/*
 *  Remain in a loop receiving the plot name until either ESCAPE is selected
 */

    unv_menu ( );

    D(printf("END unv_plot\n"));
    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: unv_menu
 *
 *  This function presents the menu which allows the universal plot to be
 *  defined.
 *****************************************************************************/

static int unv_menu ( )
{
    register int    i;

    Widget          shell, form, f_plot, f_xy, f_cmd;

    Arg             args[10];

    XtCallbackProc  cb_unv();

    XEvent          event;

    D(printf("START unv_menu\n"));

/*
 *  Create the shell widget.
 */

    D(printf("  Creating shell\n"));
    i = 0;
    shell = tui_create_trans_shell ( "Define Universal Plot", args, i );

/*
 *  Create the main form.
 */

    D(printf("  Creating forms\n"));
    i = 0;
    form   = tui_create_form ( shell, "form",   TRUE,  args, i );
    f_plot = tui_create_form ( form,  "f_plot", FALSE, args, i );
    f_xy   = tui_create_form ( form,  "f_xy",   FALSE, args, i );
    f_msid = tui_create_form ( form,  "f_msid", FALSE, args, i );
    f_cmd  = tui_create_form ( form,  "f_cmd",  FALSE, args, i );

/*
 *  Create all widgets.
 */

    i = 0;
    D(printf("  Creating plot file widgets\n"));
    tui_create_label ( f_plot, "l_plot", "Plot File", args, i );

    t_plot = tui_create_text  ( f_plot, "t_plot", "", DNAME_LEN-1, XmSINGLE_LINE_EDIT,
                                TRUE, args, i );

    i = 0;
    D(printf("  Creating xy widgets\n"));
    tui_create_label ( f_xy, "l_xlow",  " Low X Scale", args, i );
    tui_create_label ( f_xy, "l_xhigh", "High X Scale", args, i );
    tui_create_label ( f_xy, "l_ylow",  " Low Y Scale", args, i );
    tui_create_label ( f_xy, "l_yhigh", "High Y Scale", args, i );

    t_xy_id = tui_create_text ( f_xy, "t_xy_id", "", 0,  XmSINGLE_LINE_EDIT, FALSE,
                                args, i );
    t_xlow  = tui_create_text ( f_xy, "t_xlow",  "", 14, XmSINGLE_LINE_EDIT, TRUE,
                                args, i );
```

```
    t_xhigh = tui_create_text ( f_xy, "t_xhigh", "", 14, XmSINGLE_LINE_EDIT, TRUE,
                                args, i );
    t_ylow  = tui_create_text ( f_xy, "t_ylow",  "", 14, XmSINGLE_LINE_EDIT, TRUE,
                                args, i );
    t_yhigh = tui_create_text ( f_xy, "t_yhigh", "", 14, XmSINGLE_LINE_EDIT, TRUE,
                                args, i );

    i = 0;
    D(printf("  Creating MSID widgets\n"));
    tui_create_label ( f_msid, "l_msid",      "MSID",        args, i );
    tui_create_label ( f_msid, "l_src",       "Source",      args, i );
    tui_create_label ( f_msid, "l_sample",    "Sample",      args, i );
    tui_create_label ( f_msid, "l_axis_no",   "Axis #",      args, i );
    tui_create_label ( f_msid, "l_xory",      "X or Y",      args, i );
    tui_create_label ( f_msid, "l_msid_p",    "Pair MSID",   args, i );
    tui_create_label ( f_msid, "l_src_p",     "Pair Source", args, i );
    tui_create_label ( f_msid, "l_sample_p",  "Pair Sample", args, i );
    tui_create_label ( f_msid, "l_axis_no_p", "Pair Axis #", args, i );


    i = 0;
    t_msid_id = tui_create_text ( f_msid, "t_msid_id", "", 0,
                                  XmSINGLE_LINE_EDIT, FALSE, args, i );
    t_msid    = tui_create_text ( f_msid, "t_msid",    "", MSID_LENGTH,
                                  XmSINGLE_LINE_EDIT, TRUE,  args, i );
    t_src     = tui_create_text ( f_msid, "t_src",     "", 3,
                                  XmSINGLE_LINE_EDIT, TRUE,  args, i );
    t_msid_p  = tui_create_text ( f_msid, "t_msid_p",  "", MSID_LENGTH,
                                  XmSINGLE_LINE_EDIT, TRUE,  args, i );
    t_src_p   = tui_create_text ( f_msid, "t_src_p",   "", 3,
                                  XmSINGLE_LINE_EDIT, TRUE,  args, i );

    r_sample   = tui_create_rb ( f_msid, "r_sample",   samples, 2, samples[0], args, i );
    r_sample_p = tui_create_rb ( f_msid, "r_sample_p", samples, 2, samples[0], args, i );
    r_xory     = tui_create_rb ( f_msid, "r_xory",     x_and_y, 2, x_and_y[0], args, i );

    D(printf("  Creating Selection widgets\n"));
    s_axis_no   = tui_create_sel ( f_msid, "s_axis_no",   list, 0, "Axis #s", args, i );
    s_axis_no_p = tui_create_sel ( f_msid, "s_axis_no_p", list, 0, "Axis #s", args, i );

    i = 0;
    D(printf("  Creating separators\n"));
    XtManageChild ( XmCreateSeparator ( form, "sep0", args, i ) );
    XtManageChild ( XmCreateSeparator ( form, "sep1", args, i ) );
    XtManageChild ( XmCreateSeparator ( form, "sep2", args, i ) );

    i = 0;
    D(printf("  Creating commands\n"));
    tui_create_pushbutton ( f_cmd, "OK",     cb_unv, (caddr_t)1, args, i );
    tui_create_pushbutton ( f_cmd, "Plot",   cb_unv, (caddr_t)2, args, i );
    tui_create_pushbutton ( f_cmd, "Axis",   cb_unv, (caddr_t)3, args, i );
    tui_create_pushbutton ( f_cmd, "MSID",   cb_unv, (caddr_t)4, args, i );
    tui_create_pushbutton ( f_cmd, "Cancel", cb_unv, (caddr_t)0, args, i );
    tui_create_pushbutton ( f_cmd, "Help",   cb_unv, (caddr_t)5, args, i );


/*
 *  Put all input widgets in a tab group.
 */

    XmAddTabGroup ( t_plot      );
    XmAddTabGroup ( t_xlow      );
    XmAddTabGroup ( t_xhigh     );
    XmAddTabGroup ( t_ylow      );
```

```
    XmAddTabGroup ( t_yhigh     );
    XmAddTabGroup ( t_msid      );
    XmAddTabGroup ( t_src       );
/*  XmAddTabGroup ( r_sample    );*/
/*  XmAddTabGroup ( s_axis_no   );*/
/*  XmAddTabGroup ( r_xory      );*/
    XmAddTabGroup ( t_msid_p    );
    XmAddTabGroup ( t_src_p     );
/*  XmAddTabGroup ( r_sample_p  );*/
/*  XmAddTabGroup ( s_axis_no_p );*/

/*
 *  Realize and popup the shell.
 */

    XtRealizeWidget ( shell );
    XtPopup ( shell, None );
    set_cmap ( shell );

/*
 *  Wait until the user finishes with the popup.
 */

    flag = -1;
    while ( flag == -1 ) {
        XtNextEvent     ( &event );
        XtDispatchEvent ( &event );
    }

    XtDestroyWidget ( shell );

/*
 *  Return the value selected by the user (0 is for not verified, 1 is for
 *  verified.
 */

    return ( flag );
}
```

```
/*****************************************************************************
 * MODULE NAME: cb_unv
 *
 *  This function processes all menu callbacks.
 *****************************************************************************/

/* ARGSUSED */
static XtCallbackProc cb_unv ( w, closure, calldata )

    Widget      w;              /* Set to widget which in which callback originated.    */

    caddr_t     closure,        /* Indicates selected command.                          */
                *calldata;      /* Widget-specific information.                         */
{
    D(printf("START cb_unv\n"));
/*
 *  Process OK button.
 */

    if ( (int)closure == 1 ) {
        D(printf("  OK\n"));

        flag = (int)closure;
        D(printf("  FP is %d\n",fp));
        if ( fp && save_xy ( ) == 0 && save_msid ( ) == 0 )
            process_ok ( );

/*
 *  Process PLOT button.
 */

    } else if ( (int)closure == 2 ) {
        D(printf("  PLOT\n"));
        if ( process_plot ( ) == 0 ) {
            ptr_axis = ptr_msid = 0;
            display_xy   ( );
            display_msid ( );
        }

/*
 *  Process X/Y button.
 */

    } else if ( (int)closure == 3 ) {
        D(printf("  X/Y\n"));
        if ( save_xy ( ) == 0 ) {
            ptr_axis++;
            if ( ptr_axis == nbr_x )
                ptr_axis = 0;
            display_xy   ( );
        }

/*
 *  Process MSID button.
 */

    } else if ( (int)closure == 4 ) {
        D(printf("  MSID\n"));
        if ( save_msid ( ) == 0 ) {
            ptr_msid++;
            if ( ptr_msid == nbr_msid_rec )
                ptr_msid = 0;
            display_msid ( );
```

```
        }

/*
 *  Process CANCEL button.
 */

    } else if ( (int)closure == 0 ) {
        flag = (int)closure;

/*
 *  If help button was selected, display appropriate help text.
 */

    } else if ( (int)closure == 3 )
        cb_help ( 0, 21, 0 );

    D(printf("END cb_unv\n"));
    return;
}
```

```
/*****************************************************************************
 * MODULE NAME: process_plot
 *
 *  This function processes entry of a plot file name.
 *****************************************************************************/

static process_plot ( )
{
    register int    i, j;

    int             acc,
                    x_cnt,
                    y_cnt,
                    match,
                    restricted;

    char            s[80],
                    pl_name[DNAME_LEN + 5];

    Arg             args[10];

    D(printf("START process_plot\n"));
/*
 *  Check if a file is already open. If so, verify that the user wants to try to open
 *  a new file.
 */

    if ( fp && tui_display_question ( Top, "Define Universal Plot",
                        "Plot file already open - Are you sure you want to open another?",
                        -1, NULL, 0 ) == 0 )
        return ( -1 );

/*
 *  Retrieve the entered filename and validate it.
 */

    strcpy ( pl_name, XmTextGetString ( t_plot ) );
    if ( val_fn ( pl_name, YES ) == 0 )
        return ( -1 );

/*
 *  Rules for plot name validation are if no directory is specified, then the length
 *  must be less than or equal to NO_PATH_DISP.  If a directory is specified and the
 *  WEX mode is OPERATIONAL then the plot must reside under the "/WEX" directory.
 *  If the user does not type in a plot name and the ESCAPE key was not selected,
 *  an advisory come out stating that the plot name is invalid.
 */

    if ( pl_name[0] != '/' ) {
        strcpy ( unv_name, Plot_Path );
        strcat ( unv_name, pl_name );
        strcpy ( plot_name, Disp_Path );
        strcat ( plot_name, pl_name );
    } else {
        if ( strncmp ( pl_name, "/WEX/", 5 ) == 0 ) {
            get_fn ( pl_name, plot_name );
            strcpy ( unv_name, Plot_Path );
            strcat ( unv_name, plot_name );
            strcpy ( plot_name, pl_name );
        } else {
            strcpy ( unv_name, pl_name );
            strcpy ( plot_name, pl_name );
        }
```

```
    }

/*
 *  Search the active plot file list.  If the plot is active, then the user
 *  will only be allowed to update the scale values for the axes.
 */

    active = match  = NO;
    i      = 0;
    while ( ( i < MAX_PLOTS ) && ( match == NO ) ) {
        if ( ( strcmp ( Dm_Address->plots.act_plots[i], plot_name ) ) == 0 ) {
            match = active = YES;
            tui_msg ( M_YELLOW, "Plot is active - no msid information updates allowed" );
        } else
            i++;
    }

/*
 *   First attempt to read from an already existing universal plot file.  If one
 *   does not exist, then read in the plot definition file.
 */

    univ_file = YES;
    strcpy ( sav_unv_name, unv_name );
    strncat ( unv_name, ".unv\0", 5 );
    fp = fopen ( unv_name, "r+" );
    if ( fp == NULL ) {
        univ_file = NO;
        strncat ( plot_name, ".plt\0", 5 );
        fp = fopen ( plot_name, "r" );
        if ( fp == NULL ) {
            tui_msg ( M_YELLOW, "Error %d on opening plot file %s", errno, plot_name );
            return ( -1 );
        }
    }

/*
 *  Check the version.
 */

    fscanf ( fp, "%d", &version );
    if ( version > VERSION ) {
        tui_msg ( M_YELLOW, "Incompatible versions - file version %d  s/w version %d",
                            version, VERSION );
        fclose ( fp );
        return ( -1 ) ;
    }

    fscanf ( fp, "%*51c" );                     /* skip to number of x axis */
    fscanf ( fp, " %d", &nbr_x );               /* number of x axes         */
    D(printf("  NUMBER X AXIS IS %d\n",nbr_x));
    fscanf ( fp, "%d", &nbr_y );                /* number of y axes         */
    fscanf ( fp, "%d", &nbr_plot_msid );        /* number of actual  msids  */
    fscanf ( fp, "%d", &nbr_msid_rec );     /* number of msid records   */
    fscanf ( fp, "%*10c" );                     /* skip to access code      */
    fscanf ( fp, "%d", &acc );
    fscanf ( fp, "%*31c" );                     /* skip to the msid info.   */

/*
 *  Check the access restriction code to see if the plot is either a Medical
 *  or Payload restricted plot.  If the plot is access restricted and the
 *  position Id does not match the access restriction, then exit out of this
 *  routine.
 */
```

```
    restricted = chk_res ( acc, display->pos_id );
    if ( restricted == YES ) {
        fclose ( fp );
        return ( -1 );
    }

    D(printf("  Past restriction check\n"));

/*
 *  Read in the msid name, sample number, the source, the axis type, the axis
 *  number, and the plot msid, if the plot is not active.
 */

    if ( active == NO ) {
        D(printf("  Plot inactive...about to read msid records\n"));
        for ( i = 0; i < nbr_msid_rec; i++ ) {
            fscanf ( fp, "%*3c" );              /* skip to msid name        */
            fscanf ( fp, "%s", msid_rec[i].msid_name );
            D(printf("    %s\n", msid_rec[i].msid_name));
            fscanf ( fp, "%s", msid_rec[i].sample );
            fscanf ( fp, "%s", msid_rec[i].source );
            if ( version >= 3 )
                fscanf ( fp, "%*10c" );     /* skip ppl information     */
            fscanf ( fp, "%s", msid_rec[i].axis );
            fscanf ( fp, "%hd", & ( msid_rec[i].axis_nbr ) );
            fscanf ( fp, "%s", msid_rec[i].plot_msid );
            fscanf ( fp, "%s", s );     /* read to skip correctly   */
            fscanf ( fp, "%*77c" );         /* skip to next msid record */
            msid_rec[i].plot_indx = INVALID;
            D(printf("    %s %s %s %s %d %s\n", msid_rec[i].msid_name, msid_rec[i].sample
                      msid_rec[i].source, msid_rec[i].axis, msid_rec[i].axis_nbr,
                      msid_rec[i].plot_msid));
        }

    D(printf("  Past msid read\n"));

/*
 *      Search through the msids for msid pairs.
 */

        for ( i = 0; i < nbr_msid_rec; i++ ) {
            if ( msid_rec[i].plot_indx == INVALID ) {
                match = NO;
                j = i + 1;
                while ( match == NO && j < nbr_msid_rec ) {
                    if ( msid_rec[j].plot_indx == INVALID ) {
                        if ( ( strcmp ( msid_rec[i].msid_name,
                                        msid_rec[j].plot_msid ) == 0 ) &&
                             ( strcmp ( msid_rec[i].plot_msid,
                                        msid_rec[j].msid_name ) ) == 0 ) {
                            msid_rec[i].plot_indx = j;
                            msid_rec[j].plot_indx = i;
                            match = YES;
                        } else {
                            j++;
                        }
                    } else {
                        j++;
                    }
                }
            }
        }
```

```
    D(printf("  Past msid pairs search\n"));

/*
 *  Plot is active so skip over the msid records.
 */

    } else {
        D(printf("  Plot active...skipping msid records\n"));
        if ( nbr_msid_rec > 0 )
            if ( version < 3 )
                fseek ( fp, nbr_msid_rec  * 115, SEEK_CUR );
            else
                fseek ( fp, nbr_msid_rec  * 125, SEEK_CUR );
    }

/*
 *  Read in the high and low scale values for each X and Y axis.
 */

    x_cnt = y_cnt = time_x = time_y = nbr_x_axis = nbr_y_axis = 0;

    for  ( i=0; i < nbr_x + nbr_y; i++  )  {
        fscanf ( fp,"%s", s ) ;
        fscanf ( fp,"%*5c" ) ;   /* skip to scale type */
        if  ( s[0] == 'X' )  {
            fscanf ( fp,"%s",scale_type[x_cnt] ) ;
            if  ( scale_type[x_cnt][0] == 'T' )
                time_x = YES;
            else
                nbr_x_axis = YES;
            fscanf ( fp,"%*12c" ) ;   /* skip to scale values */
            fscanf ( fp,"%s",low_scale[x_cnt] ) ;
            fscanf ( fp,"%s",high_scale[x_cnt] ) ;
            x_cnt++;
        } else {
            fscanf ( fp,"%s",scale_type[MAX_AXES + y_cnt] ) ;
            if  ( scale_type[MAX_AXES + y_cnt][0] == 'T' )
                time_y = YES;
            else
                nbr_y_axis = YES;
            fscanf ( fp,"%*12c" ) ;   /* skip to scale values */
            fscanf ( fp,"%s",low_scale[MAX_AXES + y_cnt] ) ;
            fscanf ( fp,"%s",high_scale[MAX_AXES + y_cnt] ) ;
            y_cnt++;
        }
        fscanf ( fp,"%s", s ) ;  /* read to skip */
        fscanf ( fp,"%*24c" ) ;   /* skip to next record*/
    }
    fclose ( fp ) ;

    D(printf("  Past scale read..closed file\n"));

/*
 *  Create new select widgets.
 */

    XtDestroyWidget ( s_axis_no   );
    XtDestroyWidget ( s_axis_no_p );
    s_axis_no   = tui_create_sel ( f_msid, "s_axis_no",   list, nbr_x, "Axis #s", args, 0)
;
    s_axis_no_p = tui_create_sel ( f_msid, "s_axis_no_p", list, nbr_y, "Axis #s", args, 0)
;

    D(printf("  NUMBER X AXIS IS %d\n",nbr_x));
```

```
    D(printf("END process_plot"));
    return ( 0 ) ;
}
```

```
/*****************************************************************************
 * MODULE NAME: display_xy
 *
 *  This function processes selection of the X/Y button.
 *****************************************************************************/

static int display_xy ( )
{
    char        s[80];

    D(printf("  START DISPLAY_XY\n"));

    XmTextSetString ( t_xy_id, "test" );
    XmTextSetString ( t_xlow,  low_scale [ptr_axis           ] );
    XmTextSetString ( t_xhigh, high_scale[ptr_axis           ] );
    XmTextSetString ( t_ylow,  low_scale [ptr_axis + MAX_AXES] );
    XmTextSetString ( t_yhigh, high_scale[ptr_axis + MAX_AXES] );

    sprintf ( s, "Axis %d of %d", ptr_axis+1, nbr_x );
    XmTextSetString ( t_xy_id, s );

    D(printf("  END DISPLAY_XY\n"));
    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: display_msid
 *
 *  This function processes selection of the MSID button.
 *****************************************************************************/

static int display_msid ( )
{
    char            s[80];

    short           local_time;

    D(printf("  START DISPLAY_MSID\n"));
/*
 * If the msid is undefined ( "ZZZZZZZZZZ" ), then don't display the msid.  If
 * the msid is a local time msid, then set a flag.  For msids that are local
 * time, nothing is displayed for the source or the sample number.  The
 * default source is set to MTM and the sample number is set to Last.
 *
 */

    local_time = NO;
    if ( strcmp ( msid_rec[ptr_msid].msid_name, "ZZZZZZZZZZ" ) == 0 )
        XmTextSetString ( t_msid, " " );
    else {
        XmTextSetString ( t_msid, msid_rec[ptr_msid].msid_name );
        if ( strcmp ( msid_rec[ptr_msid].msid_name, "LOCAL_TIME" ) == 0 )
            local_time = YES;
    }

    if ( local_time )
        XmTextSetString ( t_src, "MTM");
    else
        XmTextSetString ( t_src, msid_rec[ptr_msid].source );

    if ( local_time )
        tui_radio_set_value ( r_sample, samples[1] );
    else
        tui_radio_set_value ( r_sample, msid_rec[ptr_msid].sample );

    tui_radio_set_value ( r_xory, msid_rec[ptr_msid].axis );

    sprintf ( s, "%d", msid_rec[ptr_msid].axis_nbr );
    XmTextSetString ( s_axis_no, s );

/*
 * If the plot msid is undefined ( "ZZZZZZZZZZ" ), then don't display the msid.
 * If the plot msid is a local time msid, then set a flag.  For msids that are
 * local time, nothing is displayed for the source or the sample number.  The
 * default source is set to MTM and the sample number is set to Last.
 */

    local_time = NO;
    plot_indx = msid_rec[ptr_msid].plot_indx;
    if ( ( strcmp ( msid_rec[plot_indx].msid_name, "ZZZZZZZZZZ" ) ) == 0 )
        XmTextSetString ( t_msid_p, " " );
    else
        XmTextSetString ( t_msid_p, msid_rec[plot_indx].msid_name );

    if ( strcmp ( msid_rec[plot_indx].msid_name, "LOCAL_TIME" ) == 0 )
        local_time = YES;

    if ( local_time )
```

```
        XmTextSetString ( t_src_p, "MTM");
    else
        XmTextSetString ( t_src_p, msid_rec[plot_indx].source );

    if ( local_time )
        XmTextSetString ( t_src_p, "MTM");

    if ( local_time )
        tui_radio_set_value ( r_sample_p, samples[1] );
    else
        tui_radio_set_value ( r_sample_p, msid_rec[plot_indx].sample );

    sprintf ( s, "%d", msid_rec[plot_indx].axis_nbr );
    XmTextSetString ( s_axis_no_p, s );

    sprintf ( s, "MSID %d of %d", ptr_msid+1, nbr_msid_rec );
    XmTextSetString ( t_msid_id, s );

    D(printf("  END DISPLAY_MSID\n"));
    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: save_xy
 *
 *  This function saves X/Y values.
 *****************************************************************************/

static int save_xy ( )
{
    register char   *p;

    int             i,
                    offset = 0;

    char            s[80];

    D(printf("  START SAVE_XY\n"));
/*
 *  Validate the low scale value.
 */

    for ( i = 0; i < 2; i++ ) {
        strcpy ( s, p = XmTextGetString ( ( offset == 0 ) ? t_xlow : t_ylow ) );
        free ( p );
        if ( strlen ( s ) <= 0 ) {
            tui_msg ( M_YELLOW, "Invalid low scale value" );
            return ( -1 );
        }
        if ( scale_type[offset + ptr_axis][0] == 'T' ) {
            if ( time_val ( s ) == NO ) {
                tui_msg ( M_YELLOW,
                    "Invalid time - use total seconds, ddd:hh:mm:ss, or a subset" );
                return ( -1 );
            }
        } else if ( limit_val ( s ) == NO && dec_val ( s ) == NO ) {
                tui_msg ( M_YELLOW,
                    "Invalid numeric - use scientific, floating point or int" );
                return ( -1 );
        }
        strcpy ( low_scale[offset + ptr_axis], s );

/*
 *  Validate the high scale value.
 */

        strcpy ( s, p = XmTextGetString ( ( offset == 0 ) ? t_xhigh : t_yhigh ) );
        free ( p );
        if ( strlen ( s ) <= 0 ) {
            tui_msg ( M_YELLOW, "Invalid high scale value" );
            return ( -1 );
        }
        if ( scale_type[offset + ptr_axis][0] == 'T' ) {
            if ( time_val ( s ) == NO ) {
                tui_msg ( M_YELLOW,
                    "Invalid time - use total seconds, ddd:hh:mm:ss, or a subset" );
                return ( -1 );
            }
        } else if ( limit_val ( s ) == NO && dec_val ( s ) == NO ) {
                tui_msg ( M_YELLOW,
                    "Invalid numeric - use scientific, floating point or int" );
                return ( -1 );
        }
        strcpy ( high_scale[offset + ptr_axis], s );
```

```
            offset = MAX_AXES;
        }

        D(printf("  END SAVE_XY\n"));
        return ( 0 );
    }
```

```
/*****************************************************************************
 * MODULE NAME: save_msid
 *
 *  This function saves MSID values.
 *****************************************************************************/

static int save_msid ( )
{
    register char   *p;

    int             local_time,
                    offset,
                    cnt,
                    new_axis_nbr;

    char            s[80],
                    real_src[4];

    D(printf("START SAVE_MSID\n"));

    plot_indx = msid_rec[ptr_msid].plot_indx;
/*
 * Receive the user input for the MSID to plot.  If the user does not input
 * anything for the MSID, then set the MSID to "ZZZZZZZZZZ" ( undefined ) .  If
 * the user inputs "LOCAL_TIME", then check to see if the plot msid is
 * LOCAL_TIME.  If the plot msid is, then reject the input.  If no axis types
 * are defined for the type of msid defined, then the msid is rejected.
 */

    D(printf("  Checking MSID\n"));
    local_time = NO;

    strcpy ( s, p = XmTextGetString ( t_msid ) );
    free ( p );

    if ( strlen ( s ) <= 0 )
        strcpy ( s, "ZZZZZZZZZZ" );

    if ( strcmp ( s, "LOCAL_TIME" ) == 0 ) {
        if ( strcmp ( msid_rec[plot_indx].msid_name, "LOCAL_TIME" ) == 0 ) {
            tui_msg ( M_YELLOW, "Cannot plot time against time - input an msid" );
            return ( -1 );
        } else if ( time_x == NO && time_y == NO ) {
            tui_msg ( M_YELLOW, "Invalid MSID selection - no time axes defined" );
            return ( -1 );
        }
    } else if ( nbr_x_axis == NO && nbr_y_axis == NO ) {
        tui_msg ( M_YELLOW, "Invalid MSID selection - no number axes defined" );
        return ( -1 );

    }

    strcpy ( msid_rec[ptr_msid].msid_name,  s );
    strcpy ( msid_rec[plot_indx].plot_msid, s );

/*
 *  If the user input LOCAL_TIME, clear out the sample and source fields.
 *  Set a flag so the user will not tab to the next two fields.
 */

    D(printf("  Setting Source and Sample for LOCAL TIME\n"));
    if ( strcmp ( s, "LOCAL_TIME" ) == 0 ) {
        local_time = YES;
```

```
        XmTextSetString ( t_src, "MTM" );
        strcpy ( msid_rec[ptr_msid].source, "MTM" );
        tui_radio_set_value ( r_sample, "L" );
        strcpy ( msid_rec[ptr_msid].sample, "L" );
    }

/*
 *  Source.
 */

    D(printf("  Checking Source\n"));
    if ( local_time == NO ) {
        strcpy ( s, p = XmTextGetString ( t_src ) );
        free ( p );
        if ( val_src ( s, real_src ) == NO ) {
            tui_msg ( M_YELLOW, "Invalid source name %s", s );
            return ( -1 );
        }
        strcpy ( msid_rec[ptr_msid].source, s );
    }

/*
 *  If not local time, retrieve the selection.
 */

    D(printf("  Checking Sample\n"));
    if ( local_time == NO )
        strcpy ( msid_rec[ptr_msid].sample, tui_radio_get_value ( r_sample ) );

/*
 *  Receive the user input for X or Y axis.  Match the axis type to the msid
 *  type.  If no compatible axis types are defined for the msid and axis pair
 *  then reject the axis type input.
 */

    D(printf("  Checking X/Y Axis\n"));
    strcpy ( msid_rec[plot_indx].axis, tui_radio_get_value ( r_xory ) );
    if ( msid_rec[ptr_msid].axis[0] == 'X' ) {
        if ( local_time == YES ) {
            if ( time_x == YES ) {
                strcpy ( msid_rec[plot_indx].axis, "Y" );
                offset = 0;
            } else {
                tui_msg ( M_YELLOW, "No X axes have been defined as time scale" );
                return ( -1 );
            }
        } else {
            if ( nbr_x_axis == YES ) {
                strcpy ( msid_rec[plot_indx].axis, "Y" );
                offset = 0;
            } else {
                tui_msg ( M_YELLOW, "No X axes have been defined as number scale" );
                return ( -1 );
            }
        }

/*
 *  Y axis.
 */

    } else {
        if ( local_time == YES ) {
            if ( time_y == YES ) {
                strcpy ( msid_rec[plot_indx].axis, "X" );
```

```
                offset = MAX_AXES;
            } else {
                tui_msg ( M_YELLOW, "No Y axes have been defined as time scale" );
                return ( -1 );
            }
        } else {
            if ( nbr_y_axis == YES ) {
                strcpy ( msid_rec[plot_indx].axis, "X" );
                offset = MAX_AXES;
            } else {
                tui_msg ( M_YELLOW, "No Y axes have been defined as number scale" );
                return ( -1 );
            }
        }
    }

/*
 *  Receive the user input for the axis number.   Validate that the axis number
 *  is within range for the number of x or y axis.   Validate also that the
 *  axis type ( number or time ) is compatible with the msid type.
 */

    D(printf("  Checking Axis Number\n"));
    if ( msid_rec[ptr_msid].axis[0] == 'X' )
        cnt = nbr_x;
    else
        cnt = nbr_y;

    strcpy ( s, p = XmTextGetString ( s_axis_no ) );
    free ( p );

    if ( strlen ( s ) > 0 ) {
        new_axis_nbr = atoi ( s );
        if ( new_axis_nbr > 0 && new_axis_nbr <= cnt ) {
            new_axis_nbr = atoi ( s );
            if ( local_time == YES ) {
                if ( scale_type[offset + new_axis_nbr - 1][0] == 'T' ) {
                    msid_rec[ptr_msid].axis_nbr = new_axis_nbr;
                } else {
                    tui_msg ( M_YELLOW, "Invalid axis - select a time scale axis" );
                    return ( -1 );
                }
            } else {
                if ( scale_type[offset + new_axis_nbr - 1][0] == 'N' ) {
                    msid_rec[ptr_msid].axis_nbr = new_axis_nbr;
                } else {
                    tui_msg ( M_YELLOW, "Invalid axis - select a number scale axis" );
                    return ( -1 );
                }
            }
        } else {
            tui_msg ( M_YELLOW, "Invalid axis # specified - input between 1 and %d", cnt);
            return ( -1 );
        }
    } else {
        tui_msg ( M_YELLOW, "Invalid axis # specified - input between 1 and %d", cnt );
        return ( -1 );
    }

/*
 *  Receive the user input for the plot MSID.  If the user does not input
 *  anything for the MSID, then set the MSID to "ZZZZZZZZZZ" ( undefined ) .  If
 *  the user inputs "LOCAL_TIME", then check to see if the msid to plot against
 *  is LOCAL_TIME.  If the msid is, then reject the input.  If no axis types
```

```
*  are defined for the type of msid defined, then the msid is rejected.
*/

    D(printf("  Checking Plot MSID\n"));
    local_time = NO;
    strcpy ( s, p = XmTextGetString ( t_msid_p ) );
    free ( p );

    if ( strlen ( s ) <= 0 )
        strcpy ( s, "ZZZZZZZZZZ" );

    if ( strcmp ( s, "LOCAL_TIME" ) == 0 ) {
        if ( strcmp ( msid_rec[ptr_msid].msid_name, "LOCAL_TIME" ) == 0 ) {
            tui_msg ( M_YELLOW, "Cannot plot time against time - input an msid" );
            return ( -1 );
        } else {
            if ( msid_rec[plot_indx].axis[0] == 'X' ) {
                if ( time_x == NO ) {
                    tui_msg ( M_YELLOW, "No X time axes defined - select an MSID" );
                    return ( -1 );
                }
            } else {
                if ( time_y == NO ) {
                    tui_msg ( M_YELLOW, "No Y time axes defined - select an MSID" );
                    return ( -1 );
                }
            }
        }
    } else {
        if ( msid_rec[plot_indx].axis[0] == 'X' ) {
            if ( nbr_x_axis == NO ) {
                tui_msg ( M_YELLOW, "No X number axes defined - select LOCAL_TIME" );
                return ( -1 );
            }
        } else {
            if ( nbr_y_axis == NO ) {
                tui_msg ( M_YELLOW, "No Y number axes defined - select LOCAL_TIME" );
                return ( -1 );
            }
        }
    }
    strcpy ( msid_rec[ptr_msid].plot_msid,  s );
    strcpy ( msid_rec[plot_indx].msid_name, s );

/*
 * If the user input LOCAL_TIME, clear out the sample and source fields.
 * Set a flag so the user will not tab to the next two fields.
 */

    D(printf("  Setting Source and Sample for LOCAL TIME\n"));
    if ( strcmp ( s, "LOCAL_TIME" ) == 0 ) {
        local_time = YES;
        XmTextSetString ( t_src_p, "MTM" );
        strcpy ( msid_rec[plot_indx].source, "MTM" );
        tui_radio_set_value ( r_sample_p, "L" );
        strcpy ( msid_rec[plot_indx].sample, "L" );
    }

/*
 * Receive the user input for the axis number of the plot msid.  The axis type
 * X or Y was determined when the axis type was input for the main msid.
 * Validate that the axis number is within range for the number of x or y
 * axis.   Validate also that the axis type ( number or time ) is compatible
 * with the msid type.
```

```
   */

      D(printf("  Checking Plot Axis Number\n"));
      if ( msid_rec[plot_indx].axis[0] == 'X' ) {
          cnt = nbr_x;
          offset = 0;
      } else {
          cnt = nbr_y;
          offset = MAX_AXES;
      }

      strcpy ( s, p = XmTextGetString ( s_axis_no_p ) );
      free ( p );
      if ( strlen ( s ) > 0 ) {
          new_axis_nbr = atoi ( s );
          if ( new_axis_nbr > 0 && new_axis_nbr <= cnt ) {
              if ( local_time == YES ) {
                  if ( scale_type[offset + new_axis_nbr - 1][0] == 'T' ) {
                      msid_rec[plot_indx].axis_nbr = new_axis_nbr;
                  } else {
                      tui_msg ( M_YELLOW, "Invalid axis - select a time scale axis" );
                      return ( -1 );
                  }
              } else {
                  if ( scale_type[offset + new_axis_nbr - 1][0] == 'N' ) {
                      msid_rec[plot_indx].axis_nbr = new_axis_nbr;
                  } else {
                      tui_msg ( M_YELLOW, "Invalid axis - select a number scale axis" );
                      return ( -1 );
                  }
              }
          } else {
              tui_msg ( M_YELLOW, "Invalid axis # specified - input between 1 and %d", cnt )
;
              return ( -1 );
          }
      } else {
          tui_msg ( M_YELLOW, "Invalid axis # specified - input between 1 and %d", cnt );
          return ( -1 );
      }

  /*
   *  Source user input for plot msid if msid is not local time.
   */

      D(printf("  Checking Source\n"));
      if ( local_time == NO ) {
          strcpy ( s, p = XmTextGetString ( t_src_p ) );
          free ( p );
          if ( val_src ( p, s ) == NO ) {
              tui_msg ( M_YELLOW, "Invalid source name %s", p );
              return ( -1 );
          }
          strcpy ( msid_rec[plot_indx].source, p );
      }

  /*
   *  Sample number input for plot msid.
   */

      D(printf(" Checking Sample\n"));
      strcpy ( msid_rec[ptr_msid].plot_sample, tui_radio_get_value ( r_sample_p ) );

      D(printf("END SAVE_MSID\n"));
```

```
        return ( 0 );
    }
```

```
/*****************************************************************************
 * MODULE NAME: process_ok
 *
 *  This function processes selection of the OK button.
 *****************************************************************************/

static int process_ok ( )
{
    register int    i;

    int             x_cnt,
                    y_cnt;

    char            s[80];

    D(printf("START PROCESS_OK\n"));
/*
 *  If no error has occurred and the user has not selected ESCAPE, then if the
 *  universal plot file does not exist, then copy the .plt to .unv.  Open the
 *  file.
 */

    if ( univ_file != YES ) {
        sprintf ( s, "cp %s %s", plot_name, unv_name );
        if ( system ( s ) != 0 ) {
            tui_msg ( M_YELLOW, "Error on creating the universal plot file" );
            return ( -1 );
        }
    }
    fp = fopen ( unv_name, "r+" );
    if ( fp == NULL ) {
        tui_msg ( M_YELLOW, "Error %d on opening the universal plot file" );
        return ( -1 );
    }

/*
 *  Write the user defined information into the universal file.  Skip over the
 *  plot file header and write the axis information and then the msid
 *  information.
 */

    if ( active == YES ) {
        if ( version < 3 )
            fseek ( fp, 105 + ( nbr_msid_rec  * 115 ), SEEK_CUR );
        else
            fseek ( fp, 105 + ( nbr_msid_rec  * 125 ), SEEK_CUR );
    } else {

/*
 *      Count the number of plot msids and call routine to alphabetically sort the
 *      MSID's and then write the alphabetized MSID's out to the universal file.
 */

        nbr_plot_msid = 0;
        for ( i = 0; i < nbr_msid_rec; i++ ) {
            if ( strcmp ( msid_rec[i].msid_name, "LOCAL_TIME" ) != 0 )
                nbr_plot_msid++;
        }
        sprintf ( s, "%2.2d", nbr_plot_msid );
        fseek ( fp, 57, SEEK_CUR );/* skip to start of msid * records */
        fprintf ( fp, "%-2s", s );

        fseek ( fp, 46, SEEK_CUR );/* skip to start of msid * records */
```

```
        sort_msid ( msid_rec, nbr_plot_msid, nbr_msid_rec );

        for ( i = 0; i < nbr_msid_rec; i++ ) {
            fseek ( fp, 3, SEEK_CUR );/* skip to msid name */
            fprintf ( fp, "%-10s ", msid_rec[i].msid_name );
            fprintf ( fp, "%-3s ", msid_rec[i].sample );
            fprintf ( fp, "%-3s ", msid_rec[i].source );
            if ( version >= 3 )
                fseek ( fp, 10, SEEK_CUR );/* skip ppl information */
            fprintf ( fp, "%-1s ", msid_rec[i].axis );
            sprintf ( s, "%d", msid_rec[i].axis_nbr );
            fprintf ( fp, "%-1s ", s );
            fprintf ( fp, "%-10s ", msid_rec[i].plot_msid );
            fseek ( fp, 78, SEEK_CUR );/* skip to next msid * record */
        }
    }

/*
 *  Write the X and Y axis scale information.
 */

    x_cnt = 0;
    y_cnt = 0;

    for ( i = 0; i < nbr_x + nbr_y; i++ ) {
        fscanf ( fp, "%s", s );
        fseek ( fp, 19, SEEK_CUR );/* skip to low scale value */
        if ( s[0] == 'X' ) {
            fprintf ( fp, "%-14s ", low_scale[x_cnt] );
            fprintf ( fp, "%-14s ", high_scale[x_cnt] );
            x_cnt++;
        } else {
            fprintf ( fp, "%-14s ", low_scale[MAX_AXES + y_cnt] );
            fprintf ( fp, "%-14s ", high_scale[MAX_AXES + y_cnt] );
            y_cnt++;
        }
        fseek ( fp, 26, SEEK_CUR );/* skip to next record */
    }
    fclose ( fp );
    strcpy ( Dm_Address->plots.unv_plot, sav_unv_name );
    for ( i = 0; i < MAX_DISP; i++ ) {
        if ( Dm_Address->display[i].disp_init == YES ) {
            if ( active == YES )
                Dm_Address->display[i].read_plot = NEW_SCALE;
            else
                Dm_Address->display[i].read_plot = READ_PLOT;
        }
    }

    D(printf("END PROCESS_OK\n"));
    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: upd_rate.c
 *
 *  This function allows the user to set the display update rate. It presents
 *  a form with a scale widget which allows selection of values in the range
 *  of 1 to 60 seconds.
 *
 *
 * INTERNAL FUNCTIONS:
 *
 *  o   cb_upd_rate     -   Callback function which processes all callbacks
 *                          from the form.
 *
 *  o   upd_rate_menu   -   This function displays the popup and waits for the
 *                          user to enter the new update rate.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  K. Noonan       - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <X11/Intrinsic.h>
#include <Xm/Scale.h>
#include <constants.h>
#include <disp.h>
#include <pf_key.h>
#include <user_inter.h>
#include <wex/EXmsg.h>

static Widget               scale;
static int                  flag;
static char                 *labels[] = { "1", "60" };
#define NUM_LABELS          2


extern struct dm_shmemory   *Dm_Address; /* Shared memory area                  */
extern short                Disp_Num;    /* Display Manager number              */
extern Widget               Top;


int upd_rate ( )
{
    int         update_rate;

    D(printf("START upd_rate\n"));
/*
 *  Save the current update rate.
 */

    update_rate = Dm_Address->display[Disp_Num].update_rate / 1000;

/*
 *  Call the menu function to allow the user to update the rate.
 */
```

```
    upd_rate_menu ( update_rate );

/*
 *  Return the status of the popup.
 */

    D(printf("END upd_rate\n"));
    return ( flag );
}
```

```
/*****************************************************************************
 * MODULE NAME: upd_rate_menu
 *
 *  This function displays the form and waits for the user to selecte the new
 *  update rate.
 *****************************************************************************/

static int upd_rate_menu ( update_rate )

    int             update_rate;
{
    register int    i;

    Arg             args[10];

    Widget          shell, form, f_data, f_cmd;

    XtCallbackProc  cb_upd_rate();

    XEvent          event;

    D(printf("START upd_rate_menu\n"));
/*
 *  Create the shell widget.
 */

    i = 0;
    shell = tui_create_trans_shell ( "Change Update Rate", args, i );

/*
 *  Create the main form.
 */

    i = 0;
    form   = tui_create_form ( shell, "form",   TRUE,  args, i );
    f_data = tui_create_form ( form,  "f_data", FALSE, args, i );
    f_cmd  = tui_create_form ( form,  "f_cmd",  FALSE, args, i );

/*
 *  Create all widgets.
 */

    i = 0;
    tui_create_label ( f_data, "label", "Update Rate (In seconds)", args, i );

    i = 0;
    scale = tui_create_scale ( f_data, "scale", 1, 60, update_rate, labels, 2, args, i );

    i = 0;
    XtManageChild ( XmCreateSeparator ( form, "sep0", args, i ) );

    i = 0;
    tui_create_pushbutton ( f_cmd, "Cancel", cb_upd_rate, (caddr_t)0, args, i );
    tui_create_pushbutton ( f_cmd, "OK",     cb_upd_rate, (caddr_t)1, args, i );
    tui_create_pushbutton ( f_cmd, "Help",   cb_upd_rate, (caddr_t)2, args, i );

/*
 *  Realize and popup the shell.
 */

    XtRealizeWidget ( shell );
    XtPopup ( shell, None );
    set_cmap ( shell );
```

```
/*
 *  Wait until the user finishes with the popup.
 */

    flag = -1;
    while ( flag == -1 ) {
        XtNextEvent     ( &event );
        XtDispatchEvent ( &event );
    }

    XtDestroyWidget ( shell );

/*
 *  Return the value selected by the user (0 is for not verified, 1 is for
 *  verified.
 */

    D(printf("END upd_rate_menu\n"));
    return ( flag );
}
```

```
/*****************************************************************************
 * MODULE NAME: cb_upd_rate
 *
 *  This callback function is called when the user selects one of the buttons
 *  on the form.
 *
 *****************************************************************************/

/* ARGSUSED */
static XtCallbackProc cb_upd_rate ( w, closure, calldata )

    Widget      w;              /* Set to widget which in which callback originated.    */

    caddr_t     closure,        /* Indicates selected command.                          */
                *calldata;      /* Widget-specific information.                         */
{
    int     update_rate;

    D(printf("START cb_upd_rate\n"));
/*
 *  Process OK button.
 */

    if ( (int)closure == 1 ) {
        XmScaleGetValue ( scale, &update_rate );
        Dm_Address->display[Disp_Num].update_rate = update_rate * 1000;
        flag = (int)closure;

/*
 *  Process CANCEL button.
 */

    } else if ( (int)closure == 0 ) {
        flag = (int)closure;

/*
 *  If help button was selected, display appropriate help text.
 */

    } else if ( (int)closure == 2 )
        cb_help ( 0, 8, 0 );

   D(printf("END cb_upd_rate\n"));
   return;
}
```

```
/*****************************************************************************
 * MODULE NAME: update.c
 *
 *  This routine calls a routine to extract data from the data buffer
 *  using the information in the decom buffer and converts that data
 *  into the proper type format as defined at display build time.  Each
 *  dynamic display item is then updated via a call to the appropriate
 *  routine.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Richard Romeo - Ford Aerospace Corporation/Houston
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Nancy Martin     - Software Engineering Section
 *                     Data Systems Department
 *                     Automation and Data Systems Division
 *                     Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/timeb.h>
#include <X11/Xlib.h>
#include <constants.h>
#include <disp.h>
#include <DDfg_graph.h>
#include <DDdisp.h>
#include <DDplot.h>
#include <wex/EXlog.h>
#include <wex/EXmsg.h>


extern struct dm_shmemory   *Dm_Address;     /* ptr to DM shared memory      */
extern struct data_info     *Dh_Address;     /* ptr to DH shared memory      */
extern struct plot_ptrs     *Plot_info_ptr;  /* ptr to plot records          */
extern struct msid_ent      *Msid;           /* ptr to Msid entries          */
extern struct tabular_ent   *Tab;            /* ptr to tabular entries       */
extern struct timeb         Last_Update;     /* last ds_getparms call time   */
extern struct fg_recs       Fg_rec;          /* foreground record structure  */

extern int      errno;                       /* system return error value    */
extern int      Year;                        /* year to be displayed         */
extern int      Year_Cat;                    /* entire year to be displayed  */
extern int      Offset;                      /* offset into data array       */
extern int      Log_File_Id;                 /* WEX logging identifier       */
extern int      Log_Pid;                     /* PID of this process          */
extern int      Logging_On;                  /* logging enabled flag         */

extern short    Nbr_of_plots;                /* # of plots for this display  */
extern short    Ovrlay_Drawn;                /* overlay change flag          */

extern unsigned char    New_Data[60000];     /* New Data Array               */
extern unsigned char    Old_Data[60000];     /* Old Data Array               */


int update (disp_num)

    short   disp_num;               /* number of display to be updated      */
```

```
{
    register char       *ldata_buffer;  /* local ptr to data buffer          */
    register struct shm_decom
                        *decom_buffer;  /* local ptr to decom buffer         */
    register struct msid_ent
                        *msid_info;     /* local ptr to msid entries         */
    register struct tabular_ent
                        *tab_info;      /* local ptr to tabular entries      */

    static int      change = 0;         /* change flag for buffer switch     */
    static short    year_flag = YES;    /* convert year                      */

    struct shm_decom    *decom_entry;   /* ptr to decom entry for msid       */
    struct timeb        current_time;   /* current system time               */
    struct plot_ptrs    *plot_ptr;      /* ptr thru plot records             */

    double  current_year,               /* holder for the current ftime      */
            before_time,                /* holder for the conv before ftime  */
            after_time;                 /* holder for the conv after ftime   */

    long    status;                     /* status of the msid value          */

    int     i, j,                       /* loop counters                     */
            year,                       /* local variable for year           */
            sample_size,                /* sample size of one sample         */
            retval,                     /* return value of memcmp            */
            offset,                     /* offset into the old data array    */
            skip_amt,                   /* # of bytes to skip in data buf    */
            error,                      /* error return value                */
            update,                     /* update flag for buffer switch     */
            bytes_logged;               /* # bytes logged by EXlogwrite      */

    short   first_pass,                 /* pass count through MSIDs          */
            index;                      /* index into decom buffer           */

    unsigned char   *start_of_sample;   /* start of sample in data buffer    */


/*  D(printf ("START update\n")); */

/*
 *  Get the current system time.
 */

    ftime(&current_time);

/*
 *  Conversion factor for the year to be displayed.
 */

    if (year_flag == YES) {
        current_year = current_time.time * SEC_YR_CONV;

        if (current_year < YEAR_DIFF)
            Year = BASE_YEAR + current_year;
        else
            Year = current_year - YEAR_DIFF;

        Year_Cat = current_year + COMP_BASE_YEAR;
        year_flag = NO;
    }

/*
```

```
 *  Check to see if it is time to update the msids.  If the current
 *  system time minus the last update time of the data is greater than
 *  update rate, update the msids.
 */

    before_time = (Last_Update.time * 1000) + Last_Update.millitm;
    after_time = (current_time.time * 1000) + current_time.millitm;

    if ((after_time - before_time) >=
                    (Dm_Address->display[disp_num].update_rate - 500.0)) {
        Last_Update.time = current_time.time;
        Last_Update.millitm = current_time.millitm;

/*
 *      If need_decom flag is set, Data Handler is updating the
 *      decom buffer so return. Otherwise set decom-in-use flag.
 */

        if (Dh_Address->need_decom == YES) {
#ifdef SUN
            usleep ( 100000 );
#else
            astpause (0, 100);
#endif
            return (0);
        }

        Dh_Address->decom_in_use[disp_num] = YES;

/*
 *      Set up the local pointers. The decom buffer will be
 *      used to access information from the updated data buffer.
 */

        decom_buffer = (struct shm_decom *) ((char *) Dh_Address +
                                                Dh_Address->decom_buf);
        ldata_buffer = (char *) ((char *) Dh_Address +
                                    Dh_Address->buffer[Dh_Address->buf_ready]);

/*
 *      On first pass, copy the data from the data buffer in Data
 *      Handler shared memory.  On the second pass, check for changes
 *      in the data and update the msids if changes exist.
 */

        for (first_pass = 0; first_pass < 2; first_pass++) {

/*
 *          Update foreground dynamic primitives.
 */

/*nlm
            Offset = 0;

            for (i = 0; i < Fg_rec.graph_num; i++) {
                error = DDupdfgr(decom_buffer, ldata_buffer, first_pass);
                if (error == -1)
                    return (-1);
            }
*/

/*
 *          Update each tabular msid.
 */
```

```
            msid_info = Msid;
            offset = 0;

            for (i = 0; i < Dh_Address->nbr_msids[disp_num]; i++) {

                if (msid_info->Tab_Index > 0 && (msid_info->hist_ind <= 0)) {
                    tab_info = Tab + msid_info->Tab_Index - 1;

/*
 *                  Get index into decom buffer.  If index is
 *                  valid, setup ptr to this msid's decom entry.
 */

                    index = Dh_Address->msid_index[disp_num][i];

                    if (index >= 0) {
                        decom_entry = decom_buffer + index;
/*
 *                      Check for decom error in local decom buffer and
 *                      calculate number of bytes that have to be skipped
 *                      per sample.
 */

                        if ((decom_entry->error == NULL)
                             && (decom_entry->num_samps > 0)) {
                            sample_size = decom_entry->sample_size;

                            if (msid_info->Sample == -1)
                                skip_amt = (decom_entry->num_samps - 1)
                                                        * sample_size;
                            else {

                                if (msid_info->Sample >
                                                    decom_entry->num_samps) {
                                    tui_msg (M_WHITE, "Sample requested exceeds number ret
rieved");

                                    decom_entry->error = YES;
                                } else
                                    skip_amt = (msid_info->Sample - 1)
                                                        * sample_size;
                            }

                            if (decom_entry->error == NULL) {
                                start_of_sample = (unsigned char *)
                                        (ldata_buffer + decom_entry->offset
                                                            + skip_amt);
/*
 *                              If first pass through msids, copy data from
 *                              DH shared memory data buffer to the local
 *                              incoming data buffer.
 */

                                if (first_pass == 0) {
                                    memcpy(&New_Data[offset],
                                                start_of_sample, sample_size);
                                    offset += sample_size;

/*
 *                                  If  not the first pass through the msids,
 *                                  compare the incoming data buffer with the
 *                                  old data buffer.  If there are changes,
 *                                  save the new data in the old data buffer
 *                                  and set the change flag.
```

```
 */

                                } else {
                                    retval = memcmp(&New_Data[offset],
                                                    &Old_Data[offset], sample_size);
                                    if (retval != 0) {
                                        memcpy (&Old_Data[offset],
                                                &New_Data[offset], sample_size);
                                        change = 1;

/*
 *
 *
 */
                                        Save the offset into the data buffer
                                        for this msid in shared memory.

                                        msid_info->data_ind = offset;

/*
 *
 *
 *
 */
                                        Extract the data value into the type
                                        union and update the value on the
                                        display.

                                        status = extract(&New_Data[offset],
                                                                    decom_entry);
                                        msid_info->Wid_Ind = i;
                                        error = updtfg(disp_num, decom_entry,
                                                msid_info, tab_info, status);
                                    }

/*
 *
 *
 */
                                        Increment offset into data buffer
                                        by the number of samples of this msid.

                                        offset += sample_size;
                                }
                            }   /* end of decom error check */
                        }   /* end of check for decom error and # samples */
                    }   /* end of index validation */
                }   /* end of non-historical tabular msid check */

                msid_info++;
            }   /* end of loop through msids */
        }   /* end of first_pass loop */

/*
 *      If the log enable flag is set, log the data.
 */

        if (Logging_On == YES) {
            bytes_logged = EXlogwrite(Log_File_Id, "DATA BUFFER   ",
                                                    11, Log_Pid, LOG_BINARY);
            bytes_logged = EXlogwrite(Log_File_Id, Old_Data, offset,
                                                    Log_Pid, LOG_BINARY);

            if (bytes_logged <= 0)
                tui_msg (M_YELLOW, "Displayer logging error %d", errno);
        }

/*
 *      Update all active plots.
 */
```

```
        for (j = 0; j < Nbr_of_plots; j++) {
            plot_ptr = Plot_info_ptr + j;

            if (plot_ptr->act_flg == YES) {
                update = proc_plt(disp_num, plot_ptr);
                if (update == YES)
                    change = YES;
            }
        }

/*
 *      Check overlay flag for buffer change
 */
        if (Ovrlay_Drawn == YES) {
            Ovrlay_Drawn = NO;
            change = YES;
        }

/*
 *      Reset the decom-in-use flag.
 *      Reset before time to current time
 */

        Dh_Address->decom_in_use[disp_num] = NO;
        before_time = after_time;

/*
 *  If it is not time to update the buffer,
 *  pause for 100 milliseconds.
 */

    } else if ((after_time - before_time) < 1000) {
#ifdef SUN
        usleep ( 1000 *(int)(500 - (after_time - before_time)));
#else
        astpause (0, (int)(500 - (after_time - before_time)));
#endif
    } else {
#ifdef SUN
        usleep    ( 1000000 );
#else
        astpause ( 0, 1000 );
#endif
    }

/*
 *  If an update to history tabs is needed,
 *  update history tabs.
 */

    if (Dm_Address->display[disp_num].dd_htab == YES) {
        retval = updtht();

        if (retval < 0)
            change = YES;

        Dm_Address->display[disp_num].dd_htab = NO;
    }

/*  D(printf ("END update\n")); */

    return (0);
}
```

```
/*****************************************************************************
 * MODULE NAME: updtbg.c
 *
 *  This routine displays background text and graphics to the display.
 *  Only necessary during initialization, reinitialization, and as a
 *  response to X expose events on the entire display window.
 *
 * DEVELOPMENT NOTES:
 *
 *  o   Only lines, rectangles, polygons, and circles have been fully
 *      converted.  Ellipses have not been tested.  Arcs and curves
 *      need more work.
 *  o   Although their is some code here to support it, background fill
 *      patterns have not been converted to X.
 *  o   Vector text records have been implemented as normal strings, since
 *      X does not directly support the concept.
 *  o   Further comments are embedded in the code.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Richard Romeo    - Ford Aerospace Corporation/Houston
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Ronnie Killough - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <X11/Xlib.h>
#include <constants.h>
#include <disp.h>
#include <DDdisp.h>
#include <wex/EXmsg.h>


extern struct bg_recs   Bg_Rec;             /* ptr to background records */
extern struct dm_shmemory  *Dm_Address;     /* ptr to DM shared mem */
extern float    Font65_height;              /* font height constants */
extern float    Font80_height;
extern float    Font100_height;


updtbg (disp_num)

    short   disp_num; /* effective display number */
{
    GC      gc;                     /* graphics context ID from DM shared memory    */
    XGCValues   *gc_val;            /* ptr to gc values struct in DM shared memory  */
    XPoint  points[100];            /* set of X points for polygon drawing          */
    Display *xdisplay;              /* ptr to X display structures for display      */
    Window  xwindow;                /* XID of effective display window              */
    Font    font;                   /* vector text font number                      */

    static unsigned char dashed[2] = {4, 4};
    static unsigned char dotted[2] = {3, 1};
    static unsigned char dot_dashed[4] = {3, 4, 3, 1};

    struct rec_header   *bg_text_ptr;       /* ptr thru bg text records             */
```

```
    struct graph_record *bg_graph_ptr;      /* ptr thru bg graphical recs      */
    struct vtext_record *vtext_ptr;         /* ptr to vector text record       */
    struct graph_pts    *poly_pts_ptr;      /* ptr thru array of polygon pts   */
    struct graph_pts    *curve_pts_ptr;     /* ptr thru array of curve pts     */
    struct line_record  *line_ptr;          /* ptr thru line records           */
    struct rectangle_record *rect_ptr;      /* ptr thru rectangle records      */
    struct polygon_record   *poly_ptr;      /* ptr thru polygon records        */
    struct circle_record    *circle_ptr;    /* ptr thru circle records         */
    struct arc_record       *arc_ptr;       /* ptr thru arc records            */
    struct ellipse_record   *ellipse_ptr;   /* ptr thru ellipse records        */
    struct ell_arc_record   *ell_arc_ptr;   /* ptr thru elliptical arc recs    */
    struct curve_record *curve_ptr;         /* ptr thru curve records          */

    float   radius,                         /* temp holder for radius          */
            angle1, angle2;                 /* temp holder for arc angles      */

    unsigned long gc_mask;                  /* mask for gc changes             */

    int     i, j, k, w,                     /* loop count variables            */
            bb_x, bb_y,                     /* coords of arc bounding box      */
            mmajor, mminor;                 /* arc maj/min axes (width/hght)   */


    D(printf("START updtbg\n"));
/*
 *  Setup local display and window variables
 */

    xdisplay = Dm_Address->xdisplay[disp_num];
    xwindow = Dm_Address->window[disp_num];
    gc = Dm_Address->gc[disp_num];
    gc_val = &Dm_Address->gc_val[disp_num];

/*
 *  Loop through graphical records in memory and display on screen
 */

    bg_graph_ptr = Bg_Rec.graph_rec;

    for (i = 0; i < Bg_Rec.graph_num; i++) {

        switch (bg_graph_ptr->graph_typ) {

            case LINE:

                line_ptr = (struct line_record *) bg_graph_ptr->graph_ptr;

                /* set up the graphics context for this line */

                if (gc_mask = set_gc(xdisplay, gc, gc_val, line_ptr->graph_col,
                                     line_ptr->line_type, line_ptr->line_wdth,
                                     NO_CHANGE, NO_CHANGE, NO_CHANGE, NO_CHANGE))
                    XChangeGC(xdisplay, gc, gc_mask, gc_val);

                /* draw the line */

                XDrawLine(xdisplay, xwindow, gc,
                            line_ptr->point1_x, line_ptr->point1_y,
                            line_ptr->point2_x, line_ptr->point2_y);

                break;

            case RECTANGLE:
```

```
            rect_ptr = (struct rectangle_record *) bg_graph_ptr->graph_ptr;

            /* set up the graphics context for this rectangle */

            if (gc_mask = set_gc(xdisplay, gc, gc_val, rect_ptr->graph_col,
                                 rect_ptr->line_type, rect_ptr->line_wdth,
                                 rect_ptr->pat_type, rect_ptr->pat_sizex,
                                 rect_ptr->pat_sizey, NO_CHANGE))
                XChangeGC(xdisplay, gc, gc_mask, gc_val);

            /* draw rectangle regardless even if have fill pattern, since
                XFillRectangle doesn't draw the complete path */

            XDrawRectangle(xdisplay, xwindow, gc,
                           rect_ptr->ul_x, rect_ptr->ul_y,
                           rect_ptr->width, rect_ptr->height);

            /* if pattern type indicates a fill pattern, fill rectangle */

            if (rect_ptr->pat_type)
                XFillRectangle(xdisplay, xwindow, gc,
                               rect_ptr->ul_x, rect_ptr->ul_y,
                               rect_ptr->width, rect_ptr->height);
            break;

        case POLYGON:

            poly_ptr = (struct polygon_record *) bg_graph_ptr->graph_ptr;

            /* set up the graphics context for this polygon */

            if (gc_mask = set_gc(xdisplay, gc, gc_val, poly_ptr->graph_col,
                                 poly_ptr->line_type, poly_ptr->line_wdth,
                                 poly_ptr->pat_type, poly_ptr->pat_sizex,
                                 poly_ptr->pat_sizey, NO_CHANGE))
                XChangeGC(xdisplay, gc, gc_mask, gc_val);

            /* copy polygon points into the XPoint structure */

            poly_pts_ptr = poly_ptr->poly_pts_ptr;

            for (w = 0; w < poly_ptr->nmbr_pts; w++) {
                points[w].x = poly_pts_ptr->point_x;
                points[w].y = poly_pts_ptr->point_y;
                poly_pts_ptr++;
            }

/* RLK 9/10/90  Assuming all points are relative to origin (depends on
            how the Display Builder generates a polygon record.  This
            polygon code was tested on hand-generated data files, so
            this may not be a correct assumption */

            /* draw the polygon */

            XDrawLines(xdisplay, xwindow, gc, points,
                           poly_ptr->nmbr_pts, CoordModeOrigin);

/* RLK 9/10/90  Assuming the polygon is non-complex so will use faster fill
            algorithm.  May be a bad assumption. */

            /* if pattern type indicates a fill pattern, fill polygon */

            if (poly_ptr->pat_type)
                XFillPolygon(xdisplay, xwindow, gc, points,
```

```
                                poly_ptr->nmbr_pts, Nonconvex,
                                CoordModeOrigin);

            break;

        case CIRCLE:

            /* setup local pointer to circle record */

            circle_ptr = (struct circle_record *) bg_graph_ptr->graph_ptr;

            /* setup graphics context for this circle */

            if (gc_mask = set_gc(xdisplay, gc, gc_val,
                            circle_ptr->graph_col, circle_ptr->line_type,
                            circle_ptr->line_wdth, circle_ptr->pat_type,
                            circle_ptr->pat_sizex, circle_ptr->pat_sizey,
                            NO_CHANGE))
                XChangeGC(xdisplay, gc, gc_mask, gc_val);

            /* calculate the major and minor axes of the circle
                (width and height of the bounding box) */

/* RLK 9/10/90  May need to adjust the major/minor axes for ratio distortion
                using ratio of size of screen in millimeters/size in pixels */

            mmajor = mminor = (int) (2.0 * circle_ptr->radius);

            /* draw circle */

            XDrawArc(xdisplay, xwindow, gc,
                        circle_ptr->bb_x, circle_ptr->bb_y, mmajor, mminor,
                        START_CIRCLE, FULL_CIRCLE);

            /* if pattern type indicates a fill pattern, fill the circle */

            if (circle_ptr->pat_type)
                XFillArc(xdisplay, xwindow, gc, circle_ptr->bb_x, circle_ptr->bb_y,
                            mmajor, mminor, START_CIRCLE, FULL_CIRCLE);

            break;

        case ARC:

            /* setup local pointer to arc record */

            arc_ptr = (struct arc_record *) bg_graph_ptr->graph_ptr;

            /* setup graphics context for this arc */

            if (gc_mask = set_gc(xdisplay, gc, gc_val, arc_ptr->graph_col,
                                arc_ptr->line_type, arc_ptr->line_wdth,
                                arc_ptr->pat_type, arc_ptr->pat_sizex,
                                arc_ptr->pat_sizey, NO_CHANGE))
                XChangeGC(xdisplay, gc, gc_mask, gc_val);

/* RLK 10/22/90 The major and minor axes may need to be adjusted and the
                angles need to be converted from radians to degrees.  This
                should be done in readbg(). */

            /* draw arc */

            XDrawArc(xdisplay, xwindow, gc, arc_ptr->bb_x, arc_ptr->bb_y,
                        arc_ptr->maj_axis, arc_ptr->min_axis,
```

```
                        arc_ptr->angle1, arc_ptr->angle2);

                /* if pattern type indicates a fill pattern, fill arc */

/* RLK 9/11/90  Assuming arc fill mode is ArcChord...see gc assignment above */

                if (arc_ptr->pat_type)
                    XFillArc(xdisplay, xwindow, gc,
                                arc_ptr->bb_x, arc_ptr->bb_y,
                                arc_ptr->maj_axis, arc_ptr->min_axis,
                                arc_ptr->angle1, arc_ptr->angle2);

                break;

            case ELLIPSE:

                /* setup local pointer to ellipse record */

                ellipse_ptr = (struct ellipse_record *)bg_graph_ptr->graph_ptr;

                /* setup graphics context for this ellipse */

                if (gc_mask = set_gc(xdisplay, gc, gc_val,
                                ellipse_ptr->graph_col,
                                ellipse_ptr->line_type, ellipse_ptr->line_wdth,
                                ellipse_ptr->pat_type, ellipse_ptr->pat_sizex,
                                ellipse_ptr->pat_sizey, NO_CHANGE))
                    XChangeGC(xdisplay, gc, gc_mask, gc_val);

                /* draw ellipse */

                XDrawArc(xdisplay, xwindow, gc,
                            ellipse_ptr->bb_x, ellipse_ptr->bb_y,
                            ellipse_ptr->maj_axis, ellipse_ptr->min_axis,
                            START_CIRCLE, FULL_CIRCLE);

                /* if pattern type indicates a fill pattern, fill ellipse */

                if (ellipse_ptr->pat_type)
                    XFillArc(xdisplay, xwindow, gc,
                            ellipse_ptr->bb_x, ellipse_ptr->bb_y,
                            ellipse_ptr->maj_axis, ellipse_ptr->min_axis,
                            START_CIRCLE, FULL_CIRCLE);

                break;

            case CURVE:

/* RLK 9/10/90  X10 had a command called XDraw which drew curves using a
                set of vertices and creating the curved surface with a
                spline algorithm.  X11 has no such command...will need to
                manually implement this algorithm. */

                /* setup local pointer to curve record */

                curve_ptr = (struct curve_record *) bg_graph_ptr->graph_ptr;

                /* setup graphics context for this curve */

                if (gc_mask = set_gc(xdisplay, gc, gc_val, curve_ptr->graph_col,
                                curve_ptr->line_type, curve_ptr->line_wdth,
                                NO_CHANGE, NO_CHANGE, NO_CHANGE, NO_CHANGE))
                    XChangeGC(xdisplay, gc, gc_mask, gc_val);
```

```
                /* copy curve vertices into XPoint structure */

                curve_pts_ptr = curve_ptr->curve_pts_ptr;

                for (k = 0; k < curve_ptr->nmbr_pts; k++) {
                    points[k].x = curve_pts_ptr->point_x;
                    points[k].y = curve_pts_ptr->point_y;
                    curve_pts_ptr++;
                }

                /* draw curve */

                XDrawLines(xdisplay, xwindow, gc, points,
                                curve_ptr->nmbr_pts, CoordModeOrigin);

                break;

            case VECT_TXT:

                /* setup local pointer to vector text record */

                vtext_ptr = (struct vtext_record *) bg_graph_ptr->graph_ptr;

/*
 *              Set text color and font
 */

                gc_mask = 0;

                if (gc_val->foreground != vtext_ptr->graph_col) {
                    gc_mask |= GCForeground;
                    gc_val->foreground = vtext_ptr->graph_col;
                }

                font = font_num(disp_num, vtext_ptr->font_style,
                                vtext_ptr->char_width, vtext_ptr->vert_size);

                if (gc_val->font != font) {
                    gc_mask |= GCFont;
                    gc_val->font = font;
                }

                if (gc_mask)
                    XChangeGC(xdisplay, gc, gc_mask, gc_val);

                /* draw string to screen */

                XDrawString(xdisplay, xwindow, gc,
                        vtext_ptr->x_position, vtext_ptr->y_position,
                        vtext_ptr->record_item, vtext_ptr->char_len);

            default:
                break;

        }   /* End of switch(graph... */

        bg_graph_ptr++;

    }   /* End of for (graphical records) */

/*
 *  Loop through text items in memory and display on screen
 */
```

```
    bg_text_ptr = Bg_Rec.record;

    /* set gc mask to GCForeground only, since that's all that might change */

/* RLK 9/11/90  The stmt above won't be true after the font stuff is fixed */


    /* display text records */

    for (i = 0; i < Bg_Rec.char_num; i++) {

/*
 *      Set text color and font
 */

        gc_mask = 0;

        if (gc_val->foreground != bg_text_ptr->color) {
            gc_mask |= GCForeground;
            gc_val->foreground = bg_text_ptr->color;
        }

        if (gc_val->font != bg_text_ptr->font_num) {
            gc_mask |= GCFont;
            gc_val->font = bg_text_ptr->font_num;
        }

        if (gc_mask)
            XChangeGC(xdisplay, gc, gc_mask, gc_val);

/*
 *      Draw string on screen
 */

        XDrawString(xdisplay, xwindow, gc,
                    bg_text_ptr->x_position, bg_text_ptr->y_position,
                    bg_text_ptr->record_item, bg_text_ptr->char_len);

        bg_text_ptr++;
    }

    D(printf("END updtbg\n"));

    return (0);
}
```

```
/*****************************************************************************
 * MODULE NAME: updtfg.c
 *
 * Accepts:     the local decom pointer, the msid entry table pointer,
 *              the tabular entry table pointer, and the data status.
 * Purpose:
 *      This routine converts the updated extracted data and formats
 *      that data to be displayed on the screen as dynamic data.  This
 *      routine also checks the limit status of each msid and displays
 *      the limit symbol and it's output status color.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Richard Romeo - Ford Aerospace Corporation/Houston
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Ronnie Killough - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <X11/Xlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/timeb.h>
#include <constants.h>
#include <disp.h>
#include <DDdisp.h>
#include <wex/EXmsg.h>


int updtfg ( disp_num, decom_ptr, lmsid, tab_info, status )

    short   disp_num;                   /* display number                   */
    struct shm_decom    *decom_ptr;     /* local decom pointer              */
    struct msid_ent     *lmsid;         /* local msid entry table pointer   */
    struct tabular_ent  *tab_info;      /* local tabular entry table ptr    */
    long    status;                     /* data status                      */
{
    char   *malloc ( );
    extern char     Cdata[256];             /* character data from shared memory */
    extern struct   dm_shmemory *Dm_Address;/* pointer to DM SHM                 */
    extern union    p_data      Data;       /* local pointer for union structure */
    extern struct   file_header *File1;     /* pointer for file header structure */
    extern struct   msid_ent    *Msid;      /* local pointer for msid structure  */
    extern struct   limit_ent   *Limit;     /* local pointer for limit structure */
    extern struct   limit_ent   *limit_info;/* local pointer for limit structure */
    extern struct   mtext_ent   *Mtext;     /* local pointer for mtext structure */
    struct mtext_ent            *mtext_ptr; /* local pointer for mtext structure */
    struct val_txt              *text_ptr;  /* local pointer for mtext structure */
    extern int      Year;                   /* current year variable             */
    extern int      Year_Cat;               /* current year variable             */
    extern long     S_color;                /* Last color used                   */
    extern struct   bg_recs     Bg_Rec;     /* addr for background records       */

    int     i;                              /* local increment variable          */
    int     k,
            l;                              /* local increment variable          */
    unsigned int    idata;                  /* division for binary conversion    */
```

```
    int     digit;                          /* number in binary conversion       */
    long    color;                          /* local variable for text color     */
    long    limit_ind;
    short   first_status;                   /* variable for status of msid       */
    short   match;                          /* variable for comparsion match     */
    short   truncate_flag = NO;             /* set to yes when truncated         */
    short   num_digits;                     /* max. nbr of binary digits         */
    short   pad;                            /* nbr of binary zeros for an even 4 */
    char    data_src[256];                  /* storage for screen output         */
    char    data_src2[256];                 /* storage for screen output         */
    char    temp_data_src[256];             /* storage for screen output         */
    char    stat_char[3];                   /* status character appended to out  */
    int     data_width;                     /* working data width                */

    Window xwindow;
    Display *xdisplay;
    int     screen;
    Font    fid;
    XGCValues   *gc_val;
    unsigned long   gc_mask;
    GC gc;
    long    x,
            y;                              /* true coordinates for text string  */

    unsigned long   days,                   /* number of days in sample          */
                    hours,                  /* number of hours in sample         */
                    minutes,                /* number of minutes in sample       */
                    seconds,                /* number of seconds in sample       */
                    milliseconds;           /* number of milliseconds in sample  */

    double  real_min;
    double  real_sec;
    double  real_hours;

/*
 * Determine what group of screen types to used
 */

    data_width = tab_info->Data_Width;
    if ( lmsid->Stat_Flag != 0 )
        data_width--;
    switch ( decom_ptr->attribute ) {
    case 'P':                       /*      Discrete Parent                  */
    case 'D':                       /*      Double Precision Real            */
    case 'L':                       /*      Natural ( Unsigned )             */
    case 06:                        /*      Discrete Parent                  */
    case 11:                        /*      BCD Time Variable                */
    case 13:                        /*      BCD Hex Time Variable            */
    case 15:                        /*      Bit Weighted Time Variable       */
    case 16:                        /*      Bit Weighted Clock Time          */
    case 17:                        /*      Bit Weighted Clock Time          */
    case 18:                        /*      Bit Weighted GMT/MET             */
    case 19:                        /*      Spacelab Floating Point          */
    case 20:                        /*      Experiment I/O GMT ( Type X )    */
    case 21:                        /*      Experiment I/O GMT ( Type H )    */

        switch ( lmsid->Scrn_Type ) {
        case 1:                             /*      Tabular Float            */

            if ( decom_ptr->length <= 32 ) {
                sprintf ( data_src, "%*.*f", tab_info->Data_Width,
                    tab_info->Dig_Right, Data.sfdata[0] );
                sprintf ( data_src2, "%.*f", tab_info->Dig_Right, Data.sfdata[0] );
            }
```

```
        else {
            sprintf ( data_src, "%*.*f", tab_info->Data_Width,
                tab_info->Dig_Right, Data.ddata );
            sprintf ( data_src2, "%.*f", tab_info->Dig_Right, Data.ddata );
        }
        if ( strlen ( data_src2 ) > tab_info->Data_Width )
            truncate_flag = YES;
        break;

    case 2:     /* Tabular Integer   */

        if ( ( decom_ptr->attribute == 'D' ) || ( decom_ptr->attribute == 19 ) ) {
            if ( ( Data.ddata < 2147483647.0 ) && ( Data.ddata > -2147483648.0 ) )
                digit = Data.ddata;
            else
                digit = 2147483647;
            sprintf ( data_src, "%*d", data_width, digit );
            sprintf ( data_src2, "%d", digit );
        }
        else {
            if ( decom_ptr->length <= 32 ) {
                sprintf ( data_src, "%*d", data_width, Data.sldata[0] );
            }
            else {
                sprintf ( data_src, "%*d", data_width, Data.ddata );
            }
            sprintf ( data_src2, "%d", Data.sldata[0] );
        }
        if ( strlen ( data_src2 ) > tab_info->Data_Width )
            truncate_flag = YES;
        break;

    case 21:    /* Tabular Unsigned Integer   */

        if ( ( decom_ptr->attribute == 'D' ) || ( decom_ptr->attribute == 19 ) ) {
            if ( ( Data.ddata < 2147483647.0 ) && ( Data.ddata > -2147483648.0 ) )
                idata = Data.ddata;
            else
                idata = 2147483647;
            sprintf ( data_src, "%*d", data_width, idata );
            sprintf ( data_src2, "%d", digit );
        }
        else {
            if ( decom_ptr->length <= 32 ) {
                sprintf ( data_src, "%*d", data_width, Data.uldata[0] );
            }
            else {
                sprintf ( data_src, "%*d", data_width, Data.ddata );
            }
            sprintf ( data_src2, "%d", Data.uldata[0] );
        }
        if ( strlen ( data_src2 ) > tab_info->Data_Width )
            truncate_flag = YES;
        break;

    case 3:     /* Tabular Scientific Notation   */

        if ( decom_ptr->length <= 32 ) {
            if ( lmsid->Stat_Flag != 0 ) {/* Display msid status     */
                sprintf ( data_src, "%*.*E", data_width,
                    tab_info->Dig_Right - 5, Data.sldata[0] );
            }
            else {
                sprintf ( data_src, "%*.*E", data_width,
```

```
                        tab_info->Dig_Right - 4, Data.sldata[0] );
                }
            }
            else {
                if ( lmsid->Stat_Flag != 0 ) {/* Display msid status     */
                    sprintf ( data_src, "%*.*E", data_width,
                         tab_info->Dig_Right - 5, Data.ddata );
                }
                else {
                    sprintf ( data_src, "%*.*E", data_width,
                        tab_info->Dig_Right - 4, Data.ddata );
                }
            }

            break;

        case 4:     /* Tabular Hexadecimal */

            sprintf ( data_src, "%*x", data_width, Data.ddata );
            sprintf ( data_src2, "%x", Data.ddata );
            if ( strlen ( data_src2 ) > tab_info->Data_Width )
                truncate_flag = YES;
            break;

        case 5:     /* Tabular Octal */
            sprintf ( data_src, "%*o", data_width, Data.ddata );
            sprintf ( data_src2, "%o", Data.ddata );
            if ( strlen ( data_src2 ) > tab_info->Data_Width )
                truncate_flag = YES;
            break;

        case 6:     /* Binary */

            if ( data_width <= 32 ) {
                num_digits = 1;
                idata = Data.ldata[0];
                temp_data_src[0] = 48;/* convert digit to character */
                while ( idata != 0 ) {
                    digit = idata % 2;
                    idata >>= 1;
                    temp_data_src[num_digits - 1] = digit + '0';
/*
 * convert digit to character
 */
                    if ( idata != 0 )
                    num_digits++;
                }
            }
            else {
                idata = Data.ldata[1];
                for ( k = 0; k < 32; k++ ) {
                    digit = idata % 2;
                    idata >> = 1;
                    temp_data_src[k] = digit + '0';/* convert digit to character */
                }
                idata = Data.ldata[0];
                for ( k = 32; k < data_width; k++ ) {
                    digit = idata % 2;
                    idata >> = 1;
                    temp_data_src[k] = digit + '0';/* convert digit to character */
                }
            }

            pad = 4 - ( num_digits % 4 );
```

```
        for ( k = 0; k < pad; k++ )
            temp_data_src[num_digits + k] = 48;

        num_digits += pad;

        if ( num_digits > tab_info->Data_Width )
            truncate_flag = YES;

        l = 0;
        for ( k = num_digits - 1; k >= 0; k-- ) {
            data_src[k] = temp_data_src[l];
            l++;
        }
        temp_data_src[num_digits] = NULL;
        data_src[data_width] = NULL;
        break;

    case 9:     /*  Multilevel Text */
        match = NO;
        mtext_ptr = Mtext + lmsid->Txt_Index - 1;
        if ( lmsid->Txt_Index > 0 ) {
            text_ptr = mtext_ptr->text_ptr;
            for ( i = 1; i <= mtext_ptr->Num_Values; i++ ) {
                if ( Data.sldata[0] == text_ptr->Value ) {
                    strcpy ( data_src, text_ptr->Text );
                    match = YES;
                    break;
                }
                text_ptr++;
            }
            if ( match == NO ) {
                strcpy ( data_src, mtext_ptr->Def_Text );
            }
        }
        else {
            data_src[0] = NULL;
        }
        break;

    case 10:    /* Tabular time 1 ( ddd:hh:mm:ss.sss )  */
    case 11:    /* Tabular time 1 ( ddd:hh:mm:ss.sss )  */
    case 12:    /* Tabular time 1 ( ddd:hh:mm:ss.sss )  */
    case 18:    /* Tabular time 1 ( ddd:hh:mm:ss.sss )  */
    case 19:    /* Tabular time 1 ( ddd:hh:mm:ss.sss )  */
    case 20:    /* Tabular time 1 ( ddd:hh:mm:ss.sss )  */

        if ( decom_ptr->attribute == 'D' ) {
            days = Data.ddata / 24.0;
            real_hours = Data.ddata - ( ( double ) days * 24.0 );
            hours = real_hours;
            real_min = ( real_hours - ( double ) hours ) * 60.0;
            minutes = real_min;
            real_sec = ( real_min - ( double ) minutes ) * 60.0;
            seconds = real_sec;
            milliseconds = ( real_sec - ( double ) seconds ) * 1000.0;
            if ( lmsid->Scrn_Type == 10 ) {
                sprintf ( data_src, "%03d:%02d:%02d:%02d.%03d",
                    days, hours, minutes, seconds, milliseconds );
            }
            else if ( lmsid->Scrn_Type == 11 ) {
                sprintf ( data_src, "%d:%03d:%02d:%02d:%02d.%03d",
                    Year_Cat, days, hours, minutes, seconds, milliseconds );
            }
```

```
            else if ( lmsid->Scrn_Type == 12 ) {
                sprintf ( data_src, "%d:%03d:%02d:%02d:%02d.%03d",
                    Year, days, hours, minutes, seconds, milliseconds );
            }
            else if ( lmsid->Scrn_Type == 18 ) {
                sprintf ( data_src, "%03d/%02d:%02d:%02d.%03d",
                    days, hours, minutes, seconds, milliseconds );
            }
            else if ( lmsid->Scrn_Type == 19 ) {
                sprintf ( data_src, "%d:%03d/%02d:%02d:%02d.%03d",
                    Year_Cat, days, hours, minutes, seconds, milliseconds );
            }
            else {
                sprintf ( data_src, "%d:%03d/%02d:%02d:%02d.%03d",
                        Year, days, hours, minutes, seconds, milliseconds );
            }

        }
        else {
            days = Data.usdata[0] >> 6;
            hours = Data.usdata[0] & 0x003F;
            minutes = ( Data.uldata[0] & 0x0000FE00 ) >> 9;
            seconds = ( Data.uldata[0] & 0x000001FF ) >> 2;
            milliseconds = ( Data.uldata[1] & 0x1FFF ) >> 3;
            if ( lmsid->Scrn_Type == 10 ) {
                sprintf ( data_src, "%03d:%02d:%02d:%02d.%03d",
                    days, hours, minutes, seconds, milliseconds );
            }
            else if ( lmsid->Scrn_Type == 11 ) {
                sprintf ( data_src, "%d:%03x:%02x:%02x:%02x.%03d",
                    Year_Cat, days, hours, minutes, seconds, milliseconds );
            }
            else if ( lmsid->Scrn_Type == 12 ) {
                sprintf ( data_src, "%d:%03x:%02x:%02x:%02x.%03d",
                    Year, days, hours, minutes, seconds, milliseconds );
            }
            else if ( lmsid->Scrn_Type == 18 ) {
                sprintf ( data_src, "%03x/%02x:%02x:%02x.%03d",
                    days, hours, minutes, seconds, milliseconds );
            }
            else if ( lmsid->Scrn_Type == 19 ) {
                sprintf ( data_src, "%d:%03x/%02x:%02x:%02x.%03d",
                    Year_Cat, days, hours, minutes, seconds, milliseconds );
            }
            else {
                sprintf ( data_src, "%03x:%02x:%02x:%02x.%03d",
                    days, hours, minutes, seconds, milliseconds );
            }
        }
        break;

    case 13:    /* Tabular time 4 ( hhh )  */

        hours = Data.usdata[0] & 0x003F;

        sprintf ( data_src, "%03x", hours );
        break;

    case 16:    /* Tabular time 4 ( hhh:mm:ss.sss )  */

        hours = ( Data.uldata[0] & 0x003F0000 ) >> 16;
        minutes = ( Data.uldata[0] & 0x0000FE00 ) >> 9;
        seconds = ( Data.uldata[0] & 0x000001FF ) >> 2;
        milliseconds = ( Data.uldata[1] & 0x1FFF ) >> 3;
```

```
            sprintf ( data_src, "%02x:%02x:%02x.%03d",
                hours, minutes, seconds, milliseconds );
            break;

        case 15:    /* Tabular time 5 ( mm:ss.sss )  */

            minutes = ( Data.uldata[0] & 0x0000FE00 ) >> 9;
            seconds = ( Data.uldata[0] & 0x000001FF ) >> 2;
            milliseconds = ( Data.uldata[1] & 0x1FFF ) >> 3;

            sprintf ( data_src, "%02x:%02x.%03d",
                minutes, seconds, milliseconds );
            break;

        case 17:    /* Tabular time 5 ( sssss.sss )  */

            days = ( Data.usdata[0] >> 6 ) & 0x000F;
            days += ( ( Data.usdata[0] >> 10 ) & 0x000F ) * 10;
            days += ( Data.usdata[0] >> 14 ) * 100;

            hours = Data.usdata[0] & 0x000F;
            hours += ( ( Data.usdata[0] >> 4 ) & 0x00000003 ) * 10;

            minutes = ( ( Data.uldata[0] >> 9 ) & 0x0000000F );
            minutes += ( ( Data.uldata[0] >> 13 ) & 0x00000007 ) * 10;

            seconds = ( Data.uldata[0] >> 2 ) & 0x0000000F;
            seconds += ( ( Data.uldata[0] >> 6 ) & 0x00000007 ) * 10;

            seconds += ( days * 86400 ) + ( hours * 3600 ) +
             ( minutes * 60 );

            milliseconds = ( Data.uldata[1] & 0x1FFF ) >> 3;

            sprintf ( data_src, "%*d.%03d", data_width - 4,
                seconds, milliseconds );
            break;

        default:
            break;
        }           /* End of screen type switch case */
        break;

    case 'E':       /*      Single Precision Real    */
    case 'F':       /*      Integer ( Signed )       */
    case 1:         /*      Real                     */
    case 2:         /*      Integer ( Signed )       */
    case 3:         /*      Integer ( No Complement )          */
    case 4:         /*      Integer ( No Complement/Overflow ) */
    case 5:         /*      Natural ( Unsigned )     */
    case 7:         /*      BCD ( Format X )         */
    case 8:         /*      BCD ( Format Y )         */
    case 9:         /*      BCD TACAN Range          */
    case 10:        /*      BCD TACAN GMT            */
    case 12:        /*      BCD Analog Variable      */
    case 14:        /*      BC Hex Analog Variable   */

        switch ( lmsid->Scrn_Type ) {
        case 1:     /* Tabular Float   */
            if ( decom_ptr->length <= 32 ) {
                sprintf ( data_src, "%*.*f", tab_info->Data_Width,
                    tab_info->Dig_Right, Data.sfdata[0] );
                sprintf ( data_src2, "%.*f", tab_info->Dig_Right, Data.sfdata[0] );
```

```
        }
        else {
            sprintf ( data_src, "%*.*f", tab_info->Data_Width,
                tab_info->Dig_Right, Data.ddata );
            sprintf ( data_src2, "%.*f", tab_info->Dig_Right, Data.ddata );
        }
        if ( strlen ( data_src2 ) > tab_info->Data_Width )
            truncate_flag = YES;
        break;

    case 2:     /* Tabular Integer   */

        if ( decom_ptr->attribute == 'E' ) {
            digit = Data.sfdata[0];
            sprintf ( data_src, "%*d", data_width, digit );
            sprintf ( data_src2, "%d", digit );
        }
        else {
            if ( decom_ptr->length <= 32 ) {
                sprintf ( data_src, "%*d", data_width, Data.sldata[0] );
            }
            else {
                sprintf ( data_src, "%*d", data_width, Data.ddata );
            }
            sprintf ( data_src2, "%d", Data.sldata[0] );
        }
        if ( strlen ( data_src2 ) > tab_info->Data_Width )
            truncate_flag = YES;
        break;

    case 21:    /* Tabular Unsigned Integer   */

        if ( decom_ptr->attribute == 'E' ) {
            if ( ( Data.ddata < 2147483647.0 ) && ( Data.ddata > -2147483648.0 ) )
                idata = Data.ddata;
            else
                idata = 2147483647;
            sprintf ( data_src, "%*d", data_width, idata );
            sprintf ( data_src2, "%d", digit );
        }
        else {
            if ( decom_ptr->length <= 32 ) {
                sprintf ( data_src, "%*d", data_width, Data.uldata[0] );
            }
            else {
                sprintf ( data_src, "%*d", data_width, Data.ddata );
            }
            sprintf ( data_src2, "%d", Data.uldata[0] );
        }
        if ( strlen ( data_src2 ) > tab_info->Data_Width )
            truncate_flag = YES;
        break;

    case 3:     /* Tabular Scientific Notation   */

        if ( decom_ptr->length <= 32 ) {
            if ( lmsid->Stat_Flag != 0 ) {/* Display msid status     */
                sprintf ( data_src, "%*.*E", data_width,
                    tab_info->Dig_Right - 5, Data.sldata[0] );
            }
            else {
                sprintf ( data_src, "%*.*E", data_width,
                    tab_info->Dig_Right - 4, Data.sldata[0] );
            }
```

```
        }
        else {
            if ( lmsid->Stat_Flag != 0 ) {/* Display msid status      */
                sprintf ( data_src, "%*.*E", data_width,
                    tab_info->Dig_Right - 5, Data.ddata );
            }
            else {
                sprintf ( data_src, "%*.*E", data_width,
                    tab_info->Dig_Right - 4, Data.ddata );
            }
        }
        break;

    case 4:     /* Tabular Hexadecimal */
        sprintf ( data_src, "%*x", data_width, Data.ddata );
        sprintf ( data_src2, "%x", Data.ddata );
        if ( strlen ( data_src2 ) > tab_info->Data_Width )
            truncate_flag = YES;
        break;

    case 5:     /* Tabular Octal */
        sprintf ( data_src, "%*o", data_width, Data.ddata );
        sprintf ( data_src2, "%o", Data.ddata );
        if ( strlen ( data_src2 ) > tab_info->Data_Width )
            truncate_flag = YES;
        break;

    case 6:     /* Binary */

        if ( data_width <= 32 ) {
            num_digits = 1;
            idata = Data.ldata[0];
            temp_data_src[0] = 48;/* convert digit to character */
            while ( idata != 0 ) {
                digit = idata % 2;
                idata >>= 1;
                temp_data_src[num_digits - 1] = digit + '0';
            /* convert digit to character */
                if ( idata != 0 )
                num_digits++;
            }
        }
        else {
            idata = Data.ldata[1];
            for ( k = 0; k < 32; k++ ) {
                digit = idata % 2;
                idata >> = 1;
                temp_data_src[k] = digit + '0';/* convert digit to character */
            }
            idata = Data.ldata[0];
            for ( k = 32; k < data_width; k++ ) {
                digit = idata % 2;
                idata >> = 1;
                temp_data_src[k] = digit + '0';/* convert digit to character */
            }
        }

        pad = 4 - ( num_digits % 4 );

        for ( k = 0; k < pad; k++ )
            temp_data_src[num_digits + k] = 48;

        num_digits += pad;
```

```
            if ( num_digits > tab_info->Data_Width )
                truncate_flag = YES;

            l = 0;
            for ( k = num_digits - 1; k >= 0; k-- ) {
                data_src[k] = temp_data_src[l];
                l++;
            }
            temp_data_src[num_digits] = NULL;
            data_src[data_width] = NULL;
            break;

        case 9:     /*  Multilevel Text */
            match = NO;
            mtext_ptr = Mtext + lmsid->Txt_Index - 1;
            if ( lmsid->Txt_Index > 0 ) {
                text_ptr = mtext_ptr->text_ptr;
                for ( i = 1; i <= mtext_ptr->Num_Values; i++ ) {
                    if ( Data.sldata[0] == text_ptr->Value ) {
                        strcpy ( data_src, text_ptr->Text );
                        match = YES;
                        break;
                    }
                    text_ptr++;
                }
                if ( match == NO ) {
                    strcpy ( data_src, mtext_ptr->Def_Text );
                }
            }
            else {
                data_src[0] = NULL;
            }
            break;

        default:
            break;

        }           /* End of screen type switch case */

        break;
    case 'B':       /*      Discrete                */
    case 24:        /*      Discrete                */

        match = NO;
        sprintf ( data_src, "%d", Data.sldata[0] );
        if ( lmsid->Scrn_Type == 9 ) {
            mtext_ptr = Mtext + lmsid->Txt_Index - 1;
            if ( lmsid->Txt_Index > 0 ) {
                text_ptr = mtext_ptr->text_ptr;
                for ( i = 1; i <= mtext_ptr->Num_Values; i++ ) {
                    if ( Data.sldata[0] == text_ptr->Value ) {
                        strcpy ( data_src, text_ptr->Text );
                        match = YES;
                        break;
                    }
                    text_ptr++;
                }
                if ( match == NO ) {
                    strcpy ( data_src, mtext_ptr->Def_Text );
                }
            }
            else {
                data_src[0] = NULL;
            }
```

```
        }
        break;

    case 'A':       /*      ASCII Character String      */
    case 22:        /*      EBCDIC Character String     */
    case 23:        /*      ASCII Character String      */

        if ( lmsid->Scrn_Type == 8 ) {
            strncpy ( data_src, Cdata, data_width );
            data_src[data_width] = '\0';
        }
        break;

    default:
        sprintf ( data_src, "%*x", data_width, Data.ddata );
        break;
    }               /* End of attribute switch case */

/*
 * Display updated value and status to the screen and return.
 */

/*
 * Get X info
 */

    xwindow = Dm_Address->window[disp_num];
    xdisplay = Dm_Address->xdisplay[disp_num];
    screen = DefaultScreen (xdisplay);
    gc = Dm_Address->gc[disp_num];
    gc_val = &Dm_Address->gc_val[disp_num];
    x = tab_info->X_XC;
    y = tab_info->Y_XC;

/*
 * Decide which status and color will be xdisplayed
 */

/*
 *  We can probably leave most of the color stuff in place, and then just
 *  before actually doing the XDrawImageString, select the GC based on
 *  the -color- variable.  If the color happens to be an uncommon one, i.e.
 *  no GC was sent to the server for it during init, then generate & send
 *  a GC for it.  Or better, create a static GC which is used for
 *  non-allocated colors.  ( Will that work? ) .
 */

    color = lmsid->Nom_Color;

    limit_ind = lmsid->Limit_Ind;
    first_status = NO;

    if ( status & DEAD_DATA ) { /* Dead Data                */
        color = lmsid->Dead_Color;
        stat_char[0] = 'D';
        strncpy ( data_src, "
                                                ", data_width );
        first_status = YES;
    }
    else if ( status & MISSING_DATA ) {/* Missing           */
        strncpy ( data_src, "
                                                ", data_width );
        color = lmsid->Sta_Color;
        stat_char[0] = 'M';
```

```
        first_status = YES;
    }
    else if ( status & STATIC_DATA ) {/* Static                */
        color = lmsid->Sta_Color;
        stat_char[0] = 'S';
        first_status = YES;
    }
    else if ( status & OFF_SCALE_HIGH ) {/* Out of crit. high  */
        if ( limit_ind > 0 )
            color = ( Limit + limit_ind - 1 )->Cr_Hcolor;
        stat_char[0] = 'H';
        first_status = YES;
    }
    else if ( status & OFF_SCALE_LOW ) {/* Out of crit. low   */
        if ( limit_ind > 0 )
            color = ( Limit + limit_ind - 1 )->Cr_Lcolor;
        stat_char[0] = 'L';
        first_status = YES;
    }
    else if ( status & CRITICAL_HIGH ) {/* Out of crit. high  */
        if ( limit_ind > 0 )
            color = ( Limit + limit_ind - 1 )->Cr_Hcolor;
        stat_char[0] = 'H';
        first_status = YES;
    }
    else if ( status & CRITICAL_LOW ) {/* Out of crit. low   */
        if ( limit_ind > 0 )
            color = ( Limit + limit_ind - 1 )->Cr_Lcolor;
        stat_char[0] = 'L';
        first_status = YES;
    }
    else if ( status & LIMIT_HIGH ) {/* Out of limits high */
        if ( limit_ind > 0 )
            color = ( Limit + limit_ind - 1 )->Hi_Color;
        stat_char[0] = 'H';
        first_status = YES;
    }
    else if ( status & LIMIT_LOW ) {/* Out of limits low   */
        if ( limit_ind > 0 )
            color = ( Limit + limit_ind - 1 )->Lo_Color;
        stat_char[0] = 'L';
        first_status = YES;
    }
    if ( truncate_flag == YES ) {    /* Truncation */
        stat_char[0] = 'T';
        truncate_flag = NO;
        first_status = YES;
    }

/*
 * If an unknown status occurs xdisplay a nominal color
 */

    if ( first_status == NO ) {
        color = lmsid->Nom_Color;
        stat_char[0] = ' ';
    }

    data_src[data_width] = NULL;
    stat_char[1] = NULL;

    if ( lmsid->Stat_Flag != 0 )
        strncat ( data_src, stat_char, 2 );
    else {
```

```
        stat_char[0] = ' ';
        strncat (data_src, stat_char, 2);
    }


    if (gc_mask = set_gc(xdisplay, gc, gc_val, (short) color, NO_CHANGE,
                -1.0, NO_CHANGE, NO_CHANGE, NO_CHANGE, tab_info->font_num)) {
        XChangeGC(xdisplay, gc, gc_mask, gc_val);
    }

    XDrawImageString(xdisplay, xwindow, gc, x, y, data_src, data_width+1);

    return(0);
}
```

```
/*****************************************************************************
 * MODULE NAME: updtht.c
 *
 *  This function updates the history table entries in the display.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Tod Milam - Ford Aerospace Corporation/Houston
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <constants.h>
#include <disp.h>
#include <DDdisp.h>
#include <wex/EXmsg.h>


extern struct dm_shmemory   *Dm_Address;    /* Display Manager shared mem.          */
extern struct ht_files      *Ht_files;      /* the array of file names and pointers */
extern struct hist_tab      *Htab;          /* the array of history tab information */
extern struct msid_ent      *Msid;          /* msid structure pointer               */
extern struct tabular_ent   *Tab;           /* Tabular entry table local ptr        */
extern short                Disp_Num;       /* display number                       */


int updtht ( )
{
    struct ht_files     *file_struct;       /* local pointer to file array          */
    struct hist_tab     *htab;              /* local pointer to history tab array   */
    struct msid_ent     *msid_ptr;          /* local msid pointer                   */
    struct tabular_ent  *tab_ptr;           /* local tabular pointer                */

    short               flag,               /* loop exit flag                       */
                        version,            /* version read from the history tab file */
                        access,             /* access rest code from the hist tab file */
                        i;                  /* loop counter                         */

    char                flight_id[5],       /* flight id read from the hist tab file */
                        strm_type[3],       /* stream type read from the hist tab file */
                        ht_file_name[50],   /* local history tab file name to open  */
                        *value,             /* value read from the hist tab file    */
                        *file,              /* holds the file name from shared memory */
                        msid[MSID_LENGTH],  /* msid name read from the hist tab file */
                        sample[5],          /* sample read from the hist tab file   */
                        source[3],          /* source read from the hist tab file   */
                        attribute,          /* attribute read from the hist tab file */
                        error;              /* error read from the hist tab file    */

    int                 size,               /* size read from the hist tab file     */
                        length,             /* length read from the hist tab file   */
                        num_samps;          /* number of samples read from the file */

    long                status;             /* status variable                      */
```

```
    double              lolimit,            /* low limit read from the hist tab file    */
                        hilimit;            /* high limit read from the hist tab file   */

    D(printf("START updtht\n"));
/*
 *  Set up the local variables.
 */

    file = Dm_Address->display[Disp_Num].htab_file;
    file_struct = Ht_files;
    htab = Htab;

/*
 *  If there aren't any history tabs on the display then exit.
 */

    if ( htab == NULL ) {
        tui_msg ( M_YELLOW, "No history tabs in this file" );
        return ( -1 );
    }

/*
 *  Find the first occurrance of file in the hist tab list.
 */

    while ( htab->next_ptr != NULL && strcmp ( htab->file_name, file ) != 0 )
        htab = htab->next_ptr;

/*
 *  If the file is not used then exit.
 */

    if ( strcmp ( htab->file_name, file ) != 0 ) {
        tui_msg ( M_YELLOW, "This htab file <%s> not used in this display", file );
        return ( 0 );
    }

/*
 *  If the list of open files is not empty.
 */

    flag = NO;
    if ( file_struct != NULL ) {

/*
 *      Loop through to see if the file is already open.
 */

        while ( file_struct->next_ptr != NULL && flag != YES ) {
            if ( strcmp ( file_struct->file_name, file ) == 0 )
                flag = YES;
            else
                file_struct = file_struct->next_ptr;
        }

/*
 *      See if exited the loop because found the file.
 */

        if ( strcmp ( file_struct->file_name, file ) == 0 )
            flag = YES;

/*
 *      If the file is not already in the open file list.
```

```
 */

        if ( flag == NO ) {
            file_struct->next_ptr =
                        (struct ht_files *)calloc ( 1, sizeof ( struct ht_files ) );
            if ( file_struct->next_ptr == NULL ) {
                tui_msg ( M_YELLOW, "Error allocating history tab file struct" );
                return ( -1 );
            }
            file_struct = file_struct->next_ptr;
            file_struct->ht_rec_ptr = htab;
        }
    }

/*
 *  There are no open files in the list.
 */

    else {
        Ht_files = ( struct ht_files * ) calloc ( 1, sizeof ( struct ht_files ) );
        file_struct = Ht_files;
        if ( file_struct == NULL ) {
            tui_msg ( M_YELLOW, "Error allocating history tab file struct" );
            return ( -1 );
        }
        file_struct->ht_rec_ptr = htab;
    }

/*
 *  If the file is not in the open list.
 */

    if ( flag == NO ) {

/*
 *      Build the file name to open.
 */

        strcpy ( file_struct->file_name, file );
        if ( file[0] != '/' ) {
            strcpy ( ht_file_name, Dm_Address->display[Disp_Num].plot_path );
            strcat ( ht_file_name, file );
        } else {
            strcpy ( ht_file_name, file );
        }
        strcat ( ht_file_name, ".htb" );

/*
 *      Open the file to read.
 */

        file_struct->file_ptr = fopen ( ht_file_name, "rb" );
        if ( file_struct->file_ptr == NULL ) {
            tui_msg ( M_YELLOW, "Error opening history tab file %s", ht_file_name );
            return ( -1 );
        }

/*
 *      Read the header of the file.
 */

        fread ( (void *) &version, 2, 1, file_struct->file_ptr );
        fread ( (void *) flight_id, 5, 1, file_struct->file_ptr );
        fread ( (void *) strm_type, 3, 1, file_struct->file_ptr );
```

```
        fread ( (void *) &file_struct->num_entries, 4, 1, file_struct->file_ptr );
        fread ( (void *) &access, 2, 1, file_struct->file_ptr );

/*
 *      Check the version of the file against the software version.
 */

        if ( version > VERSION ) {
            tui_msg ( M_YELLOW, "Version %hd of the history tab file is not supported",
                                version );
            return ( -1 );
        }

/*
 *      Check the flight id against the display flight id.
 */

    }

/*
 *  For each msid in the history tab data file.
 */

    for ( i = 0; i < file_struct->num_entries; i++ ) {

/*
 *      Read the msid, sample, and source.
 */

        fread ( (void *) msid, MSID_LENGTH, 1, file_struct->file_ptr );
        fread ( (void *) sample, 5, 1, file_struct->file_ptr );
        fread ( (void *) source, 3, 1, file_struct->file_ptr );

/*
 *      Read the decom information.
 */

        fread ( (void *) &size,      4, 1, file_struct->file_ptr );
        fread ( (void *) &length,    4, 1, file_struct->file_ptr );
        fread ( (void *) &num_samps, 4, 1, file_struct->file_ptr );
        fread ( (void *) &attribute, 1, 1, file_struct->file_ptr );
        fread ( (void *) &error,     1, 1, file_struct->file_ptr );

/*
 *      Read the limits and value.
 */

        fread ( (void *) &lolimit, 8, 1, file_struct->file_ptr );
        fread ( (void *) &hilimit, 8, 1, file_struct->file_ptr );
        value = (char *)malloc ( size );
        fread ( (void *) value, size, 1, file_struct->file_ptr );

/*
 *      Check to see if the msid is in the history tab list.
 */

        htab = file_struct->ht_rec_ptr;
        while ( htab != NULL && strcmp ( msid, (Msid + htab->msid_index - 1)->MSID ) != 0
&&
                ( strcmp ( htab->file_name, file ) == 0 ) )
            htab = htab->next_ptr;

/*
 *      If the msid is found in the history tab list.
```

```
 */

        if ( htab != NULL && ( strcmp ( htab->file_name, file ) == 0 ) ) {

/*
 *          Move values along the sequence of hist tab entries.
 */

            while ( htab->next_ptr != NULL &&
                    strcmp ( msid, (Msid+htab->next_ptr->msid_index-1)->MSID ) == 0 &&
                    htab->time_cntr > 1 &&
                    strcmp ( htab->file_name, file ) == 0 ) {

/*
 *              If the value exists then free up this memory.
 */

                if ( htab->value != NULL )
                    free ( htab->value );

/*
 *              If the previous value exists then copy it to current.
 */

                if ( htab->next_ptr->value != NULL ) {

/*
 *                  Allocate space for value and copy it.
 */

                    htab->value = (char *)malloc ( htab->next_ptr->decom_ent.size );
                    memcpy ( htab->value, htab->next_ptr->value,
                             htab->next_ptr->decom_ent.size );

/*
 *                  Copy the decom information into current struct.
 */

                    htab->decom_ent.length      = htab->next_ptr->decom_ent.length;
                    htab->decom_ent.size        = htab->next_ptr->decom_ent.size;
                    htab->decom_ent.offset      = htab->next_ptr->decom_ent.offset;
                    htab->decom_ent.num_samps   = htab->next_ptr->decom_ent.num_samps;
                    htab->decom_ent.attribute   = htab->next_ptr->decom_ent.attribute;
                    htab->decom_ent.error       = htab->next_ptr->decom_ent.error;
                    htab->decom_ent.sample_size = htab->next_ptr->decom_ent.sample_size;

                    if ( htab->htab_entr != INVALID ) {

/*
 *                      Extract the information from value.
 */

                        status = extract ( htab->value, &htab->decom_ent );

/*
 *                      Display value to the screen.
 */

                        msid_ptr = Msid + htab->msid_index - 1;
                        tab_ptr = Tab + msid_ptr->Tab_Index - 1;
                        updtfg ( Disp_Num, &htab->decom_ent,
                                 Msid + htab->msid_index - 1, tab_ptr, status);
                    }
                }
```

```
                htab = htab->next_ptr;
            }

        /*
         *          Update the most recent history tab value.
         */

            htab->value = (char *)malloc ( size );
            memcpy ( htab->value, value, size );
            free ( value );

        /*
         *          Update the most recent history tab decom buffer.
         */

            htab->decom_ent.size        = size;
            htab->decom_ent.length      = length;
            htab->decom_ent.offset      = 0;
            htab->decom_ent.num_samps   = num_samps;
            htab->decom_ent.attribute   = attribute;
            htab->decom_ent.error       = error;
            htab->decom_ent.sample_size = size / num_samps;

            if ( htab->htab_entr != INVALID ) {

        /*
         *              Extract value to be displayed.
         */

                status = extract ( htab->value, &htab->decom_ent );

        /*
         *              Display value to the screen.
         */

                msid_ptr = Msid + htab->msid_index - 1;
                tab_ptr = Tab + msid_ptr->Tab_Index - 1;
                updtfg(Disp_Num, &htab->decom_ent, msid_ptr, tab_ptr, status);
            }

            while ( htab->next_ptr != NULL && htab->next_ptr->time_cntr == 0 ) {
                htab = htab->next_ptr;
                if ( htab->value != NULL )
                    free ( htab->value );
                htab->value = (char *)malloc ( 20 );
                if ( htab->value == NULL ) {
                    tui_msg ( M_YELLOW, "Error allocating space for history tab value" );
                    return ( -1 );
                }
                if ( htab->llimit_flag == 'Y' )
                    sprintf ( htab->value, "%f", lolimit );
                else if ( htab->ulimit_flag == 'Y' )
                    sprintf ( htab->value, "%f", hilimit );
            }

        }
    }

    D(printf("END updtht\n"));
    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME: val_dt.c
 *
 *  This function validates a data stream type.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  K. Noonan       - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <ctype.h>
#include <constants.h>
#include <disp.h>
#include <wex/EXmsg.h>


extern struct dm_shmemory *Dm_Address;

extern short    Good_Strm,          /* YES, if data type is valid             */
                Disp_Num;           /* Display Manager number                 */


int short val_dt ( strm_type )

    char            strm_type[];    /* strm/data type                         */
{
    union mixed_values {
        short           data_type;
        char            ascii_str[2];
    }               ascii_val;      /* union of 2 ASCII characters and a short */

    char            new_char;       /* contains upper case letter             */

    short           i;              /* array index                            */

    D(printf("START val_dt\n"));
/*
 *  Initialize the ASCII variable to blanks.  Convert lower case letters to
 *  upper case and move into the ASCII variable.
 */

    ascii_val.data_type = TWO_BLANKS;

    for ( i = 0; i < 2; i++ ) {
        if ( ( islower ( strm_type[i] ) ) != 0 ) {
            new_char = toupper ( strm_type[i] );
            strm_type[i] = new_char;
        }
        ascii_val.ascii_str[i] = strm_type[i];
    }

/*
 *  Copy the stream type into the the display information table.
 */
```

```
    strncpy ( Dm_Address->display[Disp_Num].strm_type, strm_type, 2 );
    Dm_Address->display[Disp_Num].strm_type[2] = 0;

/*
 *  Validate the data type and set a flag if the data type is good.  If the
 *  data type is invalid, then advise and set a flag.
 */

    switch ( ascii_val.data_type ) {
    case RR:
    case R1:
    case R2:
    case SR:
    case S1:
    case S2:
        Good_Strm = YES;
        break;
    default:
        tui_msg ( M_YELLOW, "Invalid data type <%s>", strm_type );
        Good_Strm = NO;
    }

    D(printf("END val_dt\n"));
    return ( Good_Strm );
}
```

```
/*****************************************************************************
 * MODULE NAME: val_fn.c
 *
 *  The Validate Filename routine validates the length of a filename without
 *  a path specified.  In addition, if the mode is in operational and the
 *  filename has a path specified, then validation is done to see if the
 *  path starts with "/WEX/" or "/user/display/."
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  K. Noonan        - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <constants.h>
#include <disp.h>
#include <wex/EXmsg.h>


extern struct dm_shmemory *Dm_Address;      /* ptr to Display Manager shared memory  */


int val_fn ( file_name, chk_wex )

    char           *file_name;              /* pointer to the filename to validate  */

    short           chk_wex;                /* YES to chk for OPS mode and WEX      */
{
    int             len;                    /* length of the filename               */

    short           valid;                  /* return flag                          */

    char            no_path_fn[DNAME_LEN];  /* filename without a directory         */

    D(printf("START val_fn\n"));
/*
 * Check to see if the length of the filename is greater than zero.  If it is, then
 * check to see if a directory has been associated with it.  If no directory then the
 * length of the filename must be less than equal to NO_DISP_PATH.
 */

    valid = NO;
    len = strlen ( file_name );
    if ( len > 0 ) {
        if ( *file_name != '/' ) {
            if ( len > NO_PATH_DISP )
                tui_msg ( M_YELLOW, "Invalid filename - name too long" );
            else {
                valid = YES;
            }
        }

/*
 *      Filename has a path associated with it.  Check the length of the filename
 *      without the path.  If the name is greater than NO_DISP_PATH, set the file
```

```
 *      to invalid.  Check to see if an ops mode check needs to be made.  If mode
 *      is OPS, then check to see if the path starts with "/WEX".  If no ops mode
 *      is to be check then verify whether the file is located in "/WEX."  If it is
 *      then set the file to invalid.
 *
 */

        else {
            if ( len > DNAME_LEN - 1 )
                tui_msg ( M_YELLOW, "Invalid filename - name too long" );
            else {
                get_fn ( file_name, no_path_fn );
                if ( strlen ( no_path_fn ) > NO_PATH_DISP ) {
                    valid = NO;
                    tui_msg ( M_YELLOW, "Invalid filename - name too long" );
                } else {
                    if ( Dm_Address->process.wex_mode == OPS ) {
                        if ( chk_wex ) {
                            if ( ( strncmp ( file_name, "/WEX/", 5 ) ) != 0 ) {
                                tui_msg ( M_YELLOW,
                                    "Invalid filename - must be located in /WEX" );
                            } else
                                valid = YES;
                        } else {
                            if ( ( strncmp ( file_name, "/user/display/", 14 ) ) != 0 )
                                tui_msg ( M_YELLOW,
                                    "Invalid filename - must be located in /user/display"
);
                            else
                                valid = YES;
                        }
                    } else {
                        if ( !chk_wex )
                            if ( ( strncmp ( file_name, "/WEX/", 5 ) ) == 0 )
                                tui_msg ( M_YELLOW,
                                    "Invalid filename - cannot be located in /WEX" );
                            else
                                valid = YES;
                        else
                            valid = YES;
                    }   /* end check for no ops validation  */
                }   /* end chk filename length validation  */
            }   /* end chk on filename length  */
        }       /* end chk on directory  */
    } else {
        tui_msg ( M_YELLOW, "Invalid filename" );
    }

    D(printf("END val_fn\n"));
    return ( valid );
}
```

```
/*****************************************************************************
 * MODULE NAME: val_msid.c
 *
 *  This function scans the list of active MSID's for a specified MSID.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/PushB.h>
#include <constants.h>
#include <disp.h>
#include <DDdisp.h>
#include <wex/EXmsg.h>


extern struct msid_ent          *Msid;
extern struct fg_file_header    *Ffile;


int val_msid ( list, count, msid )

    char    **list,
            *msid;

    int     count;
{
    register int    i;

    D(printf("START val_msid\n"));
/*
 *  Save pointer to first MSID in valid list and then scan the list for a match
 *  with (msid).
 */

    for ( i = 0; i < count; i++ )
        if ( strcmp ( *list, msid ) == 0 )
            break;
        else
            list++;

/*
 *  If no match is found, generate an error and return 0.
 */

    if ( i == count ) {
        tui_msg ( M_YELLOW, "Invalid MSID specified" );
        return ( -1 );
    }

/*
 *  Search the complete list of MSID's to return index.
 */

    for ( i = 0; i < Ffile->Entry_Num; i++ )
        if ( strcmp ( (Msid+i)->MSID, msid ) == 0 )
```

```
            break;

    /*
     *  Return index at which MSID was matched.
     */

        D(printf("END val_msid\n"));
        return ( i );
    }
```

```
/*****************************************************************************
 * MODULE NAME: val_ppl.c
 *
 *  The Validate Filename routine validates the length of a filename without
 *  a path specified.  In addition, if the filename has a path, then the file
 *  is checked to see if it is under /WEX.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  K. Noonan       - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <constants.h>
#include <wex/EXmsg.h>


int val_ppl ( file_name )

    char            *file_name;             /* pointer to the filename to validate */
{
    int             len;                    /* length of the filename              */

    short           valid;                  /* return flag                         */

    char            no_path_fn[DNAME_LEN];  /* filename without a directory        */

    D(printf("START val_ppl\n"));
/*
 *  Check to see if the length of the filename is greater than zero.  If it is,
 *  then check to see if a directory has been associated with it.  If no
 *  directory then the length of the filename must be less than equal to
 *  PPL_NAME_LEN.
 */

    valid = NO;
    len = strlen ( file_name );
    if ( len > 0 ) {
        if ( *file_name != '/' ) {
            if ( len > PPL_NAME_LEN )
                tui_msg ( M_YELLOW, "Invalid filename - name too long" );
            else
                valid = YES;
        }

/*
 *  Filename has a path associated with it.  Check the length of the filename
 *  without the path.  If the name is greater than PPL_NAME_LEN, set the
 *  file to invalid.  Check to see if the file is in "/WEX".
 */

        else {
            get_fn ( file_name, no_path_fn );
            if ( strlen ( no_path_fn ) > PPL_NAME_LEN )
                tui_msg ( M_YELLOW, "Invalid filename - name too long" );
```

```
            else {
                if ( ( strncmp ( file_name, "/WEX/", 5 ) ) != 0 )
                    tui_msg ( M_YELLOW, "Invalid filename - must be located in /WEX" );
                else
                    valid = YES;
            }
        }
    } else
        tui_msg ( M_YELLOW, "Invalid filename" );

    D(printf("END val_ppl\n"));
    return ( valid );
}
```

```
/*****************************************************************************
 * MODULE NAME: val_src.c
 *
 *  This function validates a data source.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  K. Noonan       - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <ctype.h>
#include <constants.h>
#include <wex/EXmsg.h>


int val_src ( data_src, real_src )

    char            data_src[4],    /* data source to be validated          */

                    real_src[4];    /* actual data source - PPM or EVN      */
{
    int             valid;          /* YES, if the data source is valid      */

    char            new_char;       /* return value character                */

    short           i,              /* index counter                         */
                    number;         /* YES, if source is numeric             */

    D(printf("START val_src\n"));
/*
 *  Clear the real source variable.
 */

    real_src[0] = 0;

/*
 *  Convert the data source to upper case if the data source is not a User Comp
 *  or numeric.  If the first letter is a "U", then do not convert the other
 *  characters.
 */

    number = YES;
    valid = YES;
    i = 0;
    while ( ( i < 3 ) && ( valid == YES ) ) {
        if ( isalnum ( data_src[i] ) != 0 ) {
            if ( i == 0 ) {
                if ( isdigit ( data_src[i] ) == 0 ) {
                    number = NO;
                    new_char = toupper ( data_src[i] );
                    data_src[i] = new_char;
                }
            } else {
                if ( data_src[0] != 'U' ) {
```

```
                    if ( isdigit ( data_src[i] ) == 0 ) {
                        number = NO;
                        new_char = toupper ( data_src[i] );
                        data_src[i] = new_char;
                    }
                }
            }
        } else
            valid = NO;

        i++;
    }

/*
 *  Compare the data source with the valid data sources.
 */

    if ( valid ) {
        valid = NO;
        if ( !strncmp ( data_src, "PTM", 3 ) )
            valid = YES;
        else if ( !strncmp ( data_src, "N", 3 ) )
            valid = YES;
        else if ( number == YES ) {
            valid = YES;
            strcpy ( real_src, "PPM" );
        } else if ( !strncmp ( data_src, "DSC", 3 ) ) {
            valid = YES;
            strcpy ( real_src, "EVN" );
        } else if ( !strncmp ( data_src, "MOC", 3 ) ) {
            valid = YES;
            strcpy ( real_src, "EVN" );
        } else if ( !strncmp ( data_src, "GDR", 3 ) )
            valid = YES;
        else if ( !strncmp ( data_src, "MTM", 3 ) )
            valid = YES;
        else if ( !strncmp ( data_src, "DBM", 3 ) )
            valid = YES;
        else if ( data_src[0] == 'U' )
            valid = YES;
    }

    D(printf("END val_src\n"));
    return ( valid );
}
```

```
/*****************************************************************************
 * MODULE NAME: valmsid.c
 *
 *  This function validates an MSID by examining each letter or digit making
 *  up a string.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  K. Noonan        - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Mark D. Collier - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <const.h>
#include <wex/EXmsg.h>


int valmsid ( msid )

    char            msid[11];
{
    short           valid;

    D(printf("START valmsid\n"));
/*
 *  Validate msid, An example of a valid msid: A11A1111A or B22B2222BB
 */

    valid = YES;        /* assume msid is valid at this point */

    if ( ( msid[0] < 'A' ) || ( msid[0] > 'Z' ) )
        valid = NO;
    if ( ( msid[1] < '0' ) || ( msid[1] > '9' ) )
        valid = NO;
    if ( ( msid[2] < '0' ) || ( msid[2] > '9' ) )
        valid = NO;
    if ( ( msid[3] < 'A' ) || ( msid[3] > 'Z' ) )
        valid = NO;
    if ( ( msid[4] < '0' ) || ( msid[4] > '9' ) )
        valid = NO;
    if ( ( msid[5] < '0' ) || ( msid[5] > '9' ) )
        valid = NO;
    if ( ( msid[6] < '0' ) || ( msid[6] > '9' ) )
        valid = NO;
    if ( ( msid[7] < '0' ) || ( msid[7] > '9' ) )
        valid = NO;
    if ( ( msid[8] < 'A' ) || ( msid[8] > 'Z' ) )
        valid = NO;
    if ( ( msid[9] < 'A' ) || ( msid[9] > 'Z' ) )
        if ( msid[9] != 0 )
            valid = NO;

    D(printf("END valmsid\n"));
    return ( valid );
}
```

```
/*****************************************************************************
 * MODULE NAME: zoom.c
 *
 *  This function is called when the user selects the "Zoom" or "Reset Zoom"
 *  menu option. It adds an input callback routine for all plot widgets in
 *  the effective display which effects the zoom as soon as the user selects
 *  a zoom focus point.  If the user selects a zoom focus point outside
 *  any plot window, the cb_pbi() callback routine (set up in init_disp())
 *  will be called and will issue an advisory (only plots may be zoomed).
 *  If the command is "Zoom" (not "Reset Zoom"), the cursor is changed to
 *  a cross-hairs to signal focus point selection needed.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  K. Noonan        - Ford Aerospace Corporation
 *
 *
 * MODIFIED FOR X WINDOWS BY:
 *
 *  Ronnie Killough - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <X11/cursorfont.h>
#include <Xm/Xm.h>
#include <constants.h>
#include <disp.h>
#include <DDplot.h>
#include <wex/EXmsg.h>


extern Widget   Top;                            /* Top level widget          */

extern struct dm_shmemory   *Dm_Address;    /* ptr to DM shared memory       */
extern struct plot_ptrs     *Plot_info_ptr; /* ptr to list of plots          */

extern short    Nbr_of_plots;               /* # of plots in plot list       */


int zoom (disp_num)

    short   disp_num;                       /* effective display number      */

{
    static Cursor       cursor = NULL;      /* cross-hair cursor             */

    XtCallbackProc      cb_zoom();          /* zoom callback procedure       */

    struct plot_ptrs    *plot_ptr;          /* ptr thru list of plots        */

    int     i;                              /* loop counter                  */


   D(printf("START zoom\n"));
/*
 * RLK 11/14/90
 *
 * For zoom:
```

```
 *
 *  1. Setup callbacks for all plot window widgets.
 *  2. Change the cursor to a cross-hair.
 *
 * For reset zoom:
 *
 *  1. Setup callbacks for all plot window widgets.
 *  2. Change the cursor to a cross-hair.
 */

/*
 *  Add a callback for each plot window widget.
 */

    for (i=0; i<Nbr_of_plots; i++) {
        plot_ptr = Plot_info_ptr + i;
        XtAddCallback(plot_ptr->draw_win, XmNinputCallback, cb_zoom, disp_num);
    }

/*
 *  If the cross-hair cursor has not yet been defined, define it.
 */

    if (cursor == NULL)
        cursor = XCreateFontCursor(XtDisplay (Top), XC_crosshair);

/*
 *  Set the cursor on the top level shell
 */

    XDefineCursor ( XtDisplay ( Top ), XtWindow ( Top ), cursor );

    if (Dm_Address->shell[disp_num])
        XDefineCursor(XtDisplay(Top),
                      XtWindow(Dm_Address->shell[disp_num]), cursor);

/*
 *  Synchronize the display to cause the new cursor to appear.
 */

    XSync(XtDisplay(Top), FALSE);

    D(printf("END zoom\n"));
    return(0);
}
```

ATTACHMENT 4 - Utility Programs

```
#BINDIR = /home/project/2984/db/dm
BINDIR = /WEX/Exec
TARGET = ${BINDIR}/pdt_feed
INCLUDE = -I. -I/home/project/2984/db/include
CFLAGS = -DDEBUG -g

all: pdt_feed.o
    cc -o ${TARGET} pdt_feed.o -g

pdt_feed.o:
    cc ${CFLAGS} ${INCLUDE} -c pdt_feed.c
```

```
#BINDIR = /home/project/2984/db/dm
BINDIR = /WEX/Exec
TARGET = ${BINDIR}/pdt_feed
INCLUDE = -I. -I/home/project/2984/db/include
CFLAGS = -misalign -DDEBUG -g

all: pdt_feed.o
    cc -o ${TARGET} pdt_feed.o -g

pdt_feed.o:
    cc ${CFLAGS} ${INCLUDE} -c pdt_feed.c
```

```
/*****************************************************************************
 * mk_pdt
 *
 * This program generates a plot data file for use by the Data Displayer.
 *
 * Development Notes:
 *  o   Not functional for time plots (low/high scale)
 *****************************************************************************/

#include <stdio.h>
#include <X11/Xlib.h>
#include <fcntl.h>
#include <signal.h>
#include <sys/types.h>
#include <constants.h>
#include <disp.h>
#include <DDdisp.h>
#include <DDplot.h>
#include <wex/EXmsg.h>

#define DATA_DIR    "/WEX/Datafiles/display/SWRITEST/"
#define X   0
#define Y   1
#define CYCLES 400

extern int errno;

int plot_fp;
struct msid_info *Pmsids;
struct axis_info Paxis[2][10];
struct plot_hdr *Pheader;
char Overscale_axis;
int  Overscale_num;

read_plot_file(plot_name)
    char *plot_name;
{
    FILE    *fp;

    struct plot_hdr *header_ptr;    /* plot hdr file                        */
    struct axis_info    *axis_ptr;  /* plot axis infomation                 */
    struct msid_info    *msid_ptr;  /* plot msid infomation                 */
    struct lim_lines    *nline_ptr; /* plot nominal line infomation         */
    struct lim_lines    *lline_ptr; /* plot limit line infomation           */
    struct plot_pts *nline_pts_ptr; /* plot nominal point pairs pointer     */
    struct plot_pts *lline_pts_ptr; /* plot limit point pairs pointer       */

    short   upd_rate;
    short   version;        /* local variable for software version  */
    short   xaxes_num;      /* nbr of xaxes records                 */
    short   yaxes_num;      /* nbr of yaxes records                 */
    short   plot_hori;      /* the horizontal font size             */
    short   plot_vert;      /* the vertical font size               */
    short   match;
    short   restricted;
    short   color;          /* temp holder for color # read from DDF file */

    unsigned    size;       /* size for memory allocation           */
    int i, j, k, w, m, n, p;    /* loop count variable for rec. header  */
    int     access_rs;  /* access restriction code                  */
    int     graph_num;      /* local variable for total nbr of graph. rec */
    int     char_num;       /* local variable for total nbr of char */
    int     msid_num;       /* nbr of msid records                  */
    int     actual_msids;   /* nbr of actual msid records       */
```

```
    int     nline_num;      /* nbr of nominal line records          */
    int     lline_num;      /* nbr of limit line records            */
    int     total_nbr_records;  /* total nbr of plot records            */
    int     total_nbr_axes;
    int     name_len;
    int     x, y;

    char    temp[15];       /* use to try out character stuff    */
    char    sample[4];
    char    plot_style[5];  /* the character style */
    char    plot_fn[50];

    D(printf("START read_plot_file\n"));

    strcpy(plot_fn, plot_name);
    strncat(plot_fn, ".plt\0", 5);

    if ((fp = fopen(plot_fn,"r")) == NULL) {
        fprintf(stderr,"Error %d on reading plot file %s",errno,plot_fn);
        return(-1);
    }

/*
 *  Read the software version. If correct version continue processing by read-
 *  ing in the plot file information.
 */

    fscanf (fp, "%hd", &version);

    fscanf (fp, "%*51c");
    fscanf (fp, "%hd", &xaxes_num);
    fscanf (fp, "%hd", &yaxes_num);
    fscanf (fp, "%d", &msid_num);
    fscanf (fp, "%d", &actual_msids);
    fscanf (fp, "%d", &nline_num);
    fscanf (fp, "%d", &lline_num);
    fscanf (fp, "%hd", &upd_rate);
    fscanf (fp, "%d", &access_rs);

    if (version >= 3)
        fscanf (fp, "%*5c");

    total_nbr_records =
                xaxes_num + yaxes_num + actual_msids + nline_num + lline_num;

    if (total_nbr_records == 0) {
        fprintf(stderr, "There are no plot records ");
        fclose (fp);
        return (-1);
    }

/*
 *  Reassign pointer to beginning of file
 */

    rewind (fp);

/*
 *  Set up local pointer to beginning of plot header file and read header
 */

    header_ptr = (struct plot_hdr  *) calloc (1, sizeof (struct plot_hdr));

    if (header_ptr == NULL) {
```

```
        fprintf(stderr, "Error %d allocating plot header memory", errno);
        fclose (fp);
        return (-1);
    }

    Pheader = header_ptr;

    fscanf (fp, "%hd", &version);
    fscanf (fp, "%*51c");
    fscanf (fp, "%d", &header_ptr->xaxes_num);
    fscanf (fp, "%d", &header_ptr->yaxes_num);
    fscanf (fp, "%d", &header_ptr->msid_num);
    fscanf (fp, "%d", &header_ptr->actual_msids);
    fscanf (fp, "%d", &header_ptr->nline_num);
    fscanf (fp, "%d", &header_ptr->lline_num);
    fscanf (fp, "%hd", &header_ptr->upd_rate);
    fscanf (fp, "%hd", &header_ptr->access_rs);

    if (version >= 3)
        fscanf (fp, "%*5c");

/*
    if (msid_num > 0) {
        plot_info_ptr->plt_decom = (struct shm_decom *)
            calloc(plot_info_ptr->header->msid_num, sizeof(struct shm_decom));

        if (plot_info_ptr->plt_decom == NULL) {
            fprintf(stderr, "Error on allocating memory for plot decom");
            fclose(fp);
            return (-1);
        }
    }
*/

/*
 *  Read in the plot definition file msid records and store them into memory.
 */

    if (actual_msids > 0) {

        msid_ptr = (struct msid_info *)
                        calloc(actual_msids, sizeof(struct msid_info));

        if (msid_ptr == NULL) {
            fprintf(stderr, "Error %d allocating plot msid memory", errno);
            fclose (fp);
            return (-1);
        }

        Pmsids = msid_ptr;

        for (j = 0; j < actual_msids; j++) {
            fscanf (fp, "%hd", &msid_ptr->msid_indx);
            msid_ptr->msid_indx = j;
            fscanf (fp, "%s", msid_ptr->msid_name);
            fscanf (fp, "%s", sample);

            if (sample[0] != 'L')
                msid_ptr->sample = atoi (sample);
            else
                msid_ptr->sample = -1;

            fscanf (fp, "%s", msid_ptr->data_src);
```

```
            /* skip the ppl file and occr numbers */

            if (version >= 3)
                fscanf (fp, "%*10c");

            fscanf (fp, "%s", temp);
            msid_ptr->xory_axis = temp[0];
            fscanf (fp, "%d", &msid_ptr->axis_num);
            fscanf (fp, "%s", msid_ptr->plot_msid);
            fscanf (fp, "%s", temp);
            msid_ptr->plot_type = temp[0];
            fscanf (fp, "%d", &msid_ptr->line_type);
            fscanf (fp, "%f", &msid_ptr->line_width);
            fscanf (fp, "%s", msid_ptr->plot_char);
            fscanf (fp, "%s", plot_style);
            fscanf (fp, "%hd", &plot_hori);
            fscanf (fp, "%hd", &plot_vert);

/* RLK 9/12/90  More font stuff to fix.
            DBfontnum (plot_style, plot_hori, plot_vert, &msid_ptr->plot_font);
*/

            fscanf (fp, "%hd", &msid_ptr->icon_indx);
            fscanf (fp, "%s", temp);
            msid_ptr->plot_conn = temp[0];
            fscanf (fp, "%hd", &color);
            fscanf (fp, "%d", &msid_ptr->stat_flag);
            fscanf (fp, "%d", &msid_ptr->miss_flag);
            fscanf (fp, "%hd", &color);
            fscanf (fp, "%hd", &color);
            fscanf (fp, "%hd", &color);
            fscanf (fp, "%hd", &color);
            fscanf (fp, "%hd", &color);
            fscanf (fp, "%d", &msid_ptr->oper_type);
            fscanf (fp, "%f", &msid_ptr->oper_width);
            fscanf (fp, "%d", &msid_ptr->crit_type);
            fscanf (fp, "%f", &msid_ptr->crit_width);
            msid_ptr->pair_ptr = NULL;
            msid_ptr->first_pt = YES;

            msid_ptr++;

        }   /* End of -for- (total nbr of msids) */

/*
 *      Set the pair index pointers
 */

        msid_ptr = Pmsids;

        for (i = 0; i < actual_msids; i++) {

            /* the current msid is represented by msid_ptr + i  */

            if ((msid_ptr + i)->pair_ptr == NULL) {
                match = NO;
                k = i + 1;

                while (match == NO && k < actual_msids) {

                    if ((msid_ptr + k)->pair_ptr == NULL) {

                        if (    !strcmp((msid_ptr + i)->msid_name,
                                        (msid_ptr + k)->plot_msid)
```

```
                        &&  !strcmp((msid_ptr + i)->plot_msid,
                                    (msid_ptr + k)->msid_name)) {
                        (msid_ptr + i)->pair_ptr = msid_ptr + k;
                        (msid_ptr + k)->pair_ptr = msid_ptr + i;
                        match = YES;
                    }
                    else
                        k++;
                }
                else
                    k++;

            }   /* end while */

        }       /* end of if ... == NULL */

    }           /* end of -for- (i) */

}   /* end -if- (actual msid > 0) */

/*
 *  Read in the plot definition file axis records and store them into memory.
 */

    total_nbr_axes = xaxes_num + yaxes_num;

    if (total_nbr_axes > 0) {

/*
        axis_ptr = (struct axis_info *)
                        calloc(total_nbr_axes, sizeof(struct axis_info));

        if (axis_ptr == NULL) {
            tui_msg(M_YELLOW, "Error %d allocating plot axis memory", errno);
            fclose (fp);
            return (-1);
        }
*/

        x = 0;
        y = 0;

        for (m = 0; m < total_nbr_axes; m++) {
            fscanf (fp, "%s", temp);

            if (temp[0] == 'X') {
                axis_ptr = &Paxis[X][x];
                x++;
            } else {
                axis_ptr = &Paxis[Y][y];
                y++;
            }

            axis_ptr->axis_xory = temp[0];
            fscanf (fp, "%d", &axis_ptr->axis_num);
            fscanf (fp, "%hd", &axis_ptr->axis_type);
            fscanf (fp, "%s", temp);
            axis_ptr->scal_type = temp[0];
            fscanf (fp, "%d", &axis_ptr->end_code);
            fscanf (fp, "%hd", &axis_ptr->axis_pos);
            fscanf (fp, "%hd", &color);

            fscanf (fp, "%s", axis_ptr->low_scale);
```

```
/*
            if (axis_ptr->scal_type == 'T')
                axis_ptr->low_value = (double) DBp_atimei(axis_ptr->low_scale);
            else
*/
                sscanf(axis_ptr->low_scale, "%lf", &axis_ptr->low_value);

            axis_ptr->org_low_val = axis_ptr->low_value;

            fscanf (fp, "%s", axis_ptr->high_scal);

/*
            if (axis_ptr->scal_type == 'T')
                axis_ptr->high_value = (double) DBp_atimei(axis_ptr->high_scal);
            else
*/
                sscanf (axis_ptr->high_scal, "%lf", &axis_ptr->high_value);

            axis_ptr->org_high_val = axis_ptr->high_value;

            fscanf (fp, "%s", temp);
            axis_ptr->auto_flag = temp[0];
            fscanf (fp, "%hd", &axis_ptr->grad_vals);
            fscanf (fp, "%s", temp);
            axis_ptr->vis_flag = temp[0];
            fscanf (fp, "%s", temp);
            axis_ptr->grid_flag = temp[0];
            fscanf (fp, "%hd", &axis_ptr->grid_gran);
            fscanf (fp, "%hd", &axis_ptr->grid_type);
            fscanf (fp, "%hd", &color);
            fscanf (fp, "%hd", &axis_ptr->maj_ticks);
            fscanf (fp, "%hd", &axis_ptr->min_ticks);

            axis_ptr->axis_active = YES;
        }
    }

    fclose(fp);

    D(printf("END read_plot_file\n"));

    return(actual_msids);
}


/*****************************************************************************
 * Function:    gen_and_write_data
 * Purpose:
 *      To generate random data based on the axis scale values and
 *      write it to the plot data file.
 *****************************************************************************/

double gen_and_write_data(msid_ptr, axis_ptr, increment)
    struct msid_info *msid_ptr;
    struct axis_info *axis_ptr;
    double increment;
{
    long low, high, tmp;
    double d_val;
    long status;
    static short indx = 0;
    int low_flag;
    float mod_factor;
```

```
        low = (long) axis_ptr->low_value;
        high = (long) axis_ptr->high_value;

        low_flag = 0;
        if (low > high) {
            tmp = low;
            low = high;
            high = tmp;
            low_flag = 1;
        }

        status = 0;

        if (axis_ptr->axis_xory == Overscale_axis
                && axis_ptr->axis_num == Overscale_num)
            mod_factor = 1.5 * ((high - low) / (double)(CYCLES * .60));
        else
            mod_factor = 1.5 * ((high - low) / (double) CYCLES);

        if (mod_factor > 1.0)
            increment += (double) (random() % (long)mod_factor);
        else
            increment += (double) (random() % (long)(mod_factor * 1000.0)) / 1000.0;

    /*
     *  plot low value to high value
     */

        if (low > 0) {
            if (low_flag)
                d_val = (double) high - increment;
            else
                d_val = (double) low + increment;
        } else
                d_val = (double) low + increment;

    /*
        d_val = (double) (random() % (high - low + 1) + low);
    */

        write(plot_fp, (char *)&indx, sizeof(short));
        write(plot_fp, (char *)&status, sizeof(long));
        write(plot_fp, (char *)&d_val, sizeof(double));

        return(increment);
    }

    /*****************************************************************************
     * FP interrupt handler
     *****************************************************************************/

    int control_fpe()
    {
        signal(SIGFPE, control_fpe);
    }

    main(argc, argv)
        int argc;
        char **argv;
    {
        int x,y;
        int num_msids;
        char plt_file[50];
        char plot_data[50];
```

```
    char buffer[100];
    int size,length;
    short num_samps;
    char attribute, error;
    int i, j;
    struct msid_info *msid_ptr;
    struct axis_info *axis_ptr;
    double increment[2][20];
    int idx;


    signal(SIGFPE, control_fpe);

/*
 *  Extract plt file name and prefix path
 */

    argv++;
    sprintf(plt_file, "%s%s", DATA_DIR, *argv);
    strcpy(plot_data, plt_file);
    printf("Processing plot file %s\n", plt_file);

/*
 *  If overscale args are present, extract it, else
 *  set to overscale axis to 'N' (no axis should be overscaled).
 *  If an axis is specified for 'overscale', that
 *  means data for that axis will be generated intentionally
 *  high to test rescaling of axes.
 */

    if (argc > 3) {
        argv++;
        Overscale_num = atoi(*argv);
        argv++;
        Overscale_axis = **argv;
    } else {
        Overscale_num = 0;
        Overscale_axis = 'N';
    }

/*
 *  Read plot description file (.plt)
 */

    num_msids = read_plot_file(plt_file);

    if (num_msids == -1)
        return(-1);

/*
 *  Open plot data file (.pdt)
 */

    strcat(plot_data, ".pdt");
    printf("Processing data plot file %s\n", plot_data);

    plot_fp = open(plot_data, O_RDWR | O_CREAT | O_TRUNC, 0666);

    if (plot_fp == -1) {
        fprintf(stderr,"Error %d on creating the plot data file",errno);
         return(-1);
    }

/*
```

```
 *  Just put in blanks for header
 */

    strcpy(buffer, "
          ");

    write(plot_fp, buffer, 80);

/*
 *  Assume all data is double and write decom info for
 *  all non-time msids.
 *      size = 12    (4 for status)
 *      length = 8
 *      number samples = 1
 *      attribute = D
 *      error = 0
 */

    size = 12;
    length = 8;
    num_samps = 1;
    attribute = 'D';
    error = NULL;

    msid_ptr = Pmsids;

    for (i=0; i<num_msids; i++) {
        if (strcmp(msid_ptr->msid_name, LOCAL_TIME)) {
            write(plot_fp, (char *)&size, 4);
            write(plot_fp, (char *)&length, 4);
            write(plot_fp, &num_samps, 2);
            write(plot_fp, &attribute, 1);
            write(plot_fp, &error, 1);
            write(plot_fp, buffer, 12);
        }
    }

/*
 *  Generate random data for each msid -n- times
 */

    for (i=0; i<CYCLES; i++) {
        msid_ptr = Pmsids;

        for (j=0; j<num_msids; j++) {

            if (strcmp(msid_ptr->msid_name, LOCAL_TIME)) {

                if (msid_ptr->xory_axis == 'X') {
                    axis_ptr = &Paxis[X][(msid_ptr->axis_num-1)];
                    idx = X;
                } else {
                    axis_ptr = &Paxis[Y][(msid_ptr->axis_num-1)];
                    idx = Y;
                }

                increment[idx][j] = gen_and_write_data(msid_ptr, axis_ptr, increment[idx][
j]);
            }

            msid_ptr++;
        }

    }
```

```
/*
 *  Close plot data file
 */

    close(plot_fp);

}
```

```
/*****************************************************************************
 * mk_pdt
 *
 *  This program continuously generates plot data for use by the Data Displayer.
 *  It is invoked by the Display Manager immediately after the Data Handler is
 *  invoked, and halts itself when the DH halt flag is detected. It is virtually
 *  inactive until it detects a plot start flag in Display Manager shared
 *  memory.  This program handles all plot-related flags that would be handled
 *  by a fully functional Data Handler. It attaches to the Display Manager
 *  shared memory only.
 *
 *
 * Development Notes:
 *
 *****************************************************************************/

#include <stdio.h>
#include <X11/Xlib.h>
#include <fcntl.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/timeb.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <constants.h>
#include <disp.h>
#include <DDdisp.h>
#include <DDplot.h>
#include <wex/EXmsg.h>

#define DATA_DIR    "/WEX/Datafiles/display/SWRITEST/"
#define X   0
#define Y   1
#define CYCLES 1000

extern int errno;

struct dm_shmemory  *Dm_Address;
char Plot_name[MAX_PLOTS][50];
int Plot_fp[MAX_PLOTS];
struct msid_info *Pmsids[MAX_PLOTS];
struct axis_info Paxis[MAX_PLOTS][2][10];
struct plot_hdr *Pheader[MAX_PLOTS];
char Overscale_axis;
int  Overscale_num;
int Num_msids[MAX_PLOTS];

/*****************************************************************************
 * Function:    start_plot
 * Purpose:
 *      To read the plot file and initialize plot data file
 *****************************************************************************/

start_plot(disp_num)
    short disp_num;
{
    struct msid_info *plot_ptr;
    int i, empty, match1;
    char plot_data[50];
    char buffer[100];
    int size,length;
    short num_samps;
    char attribute, error;
    struct msid_info *msid_ptr;
```

```
    struct axis_info *axis_ptr;
    int plot_fp;

/*
 *  Make sure plot isn't already active.
 *  If it is, issue advisory and exit.
 *  Else, retain index of an empty plot slot.
 */

    for (i=0; i<MAX_PLOTS; i++) {

        if (Plot_name[i][0] == 0)
            empty = i;
        else if (!strcmp(Plot_name[i],
                         Dm_Address->display[disp_num].plot_name)) {
            printf("Plot %s already active",Dm_Address->display[disp_num].plot_name);
            return(0);
        }
    }

    strcpy(Plot_name[empty], Dm_Address->display[disp_num].plot_name);

/*
 *  Locate the plot in the active plot list
 *  in DM shared memory.
 */

    match1 = NO;
    i = 0;

    while( (i < MAX_PLOTS) && (match1 == NO)) {

        if (!strcmp(Dm_Address->display[disp_num].plot_name,
                        Dm_Address->plots.act_plots[i]))
            match1 = YES;
        else
             i++;
    }

    if (match1 == NO) {
        printf("Plot <%s> not found in active plot list",
                                    Dm_Address->display[disp_num].plot_name);
        return(-1);
    }

/*
 *  Read the plot information file (.plt).  If no error,
 *  place the new plot record in the list of plots.
 */

    Num_msids[empty] =
                read_plot_file(Dm_Address->display[disp_num].plot_name, empty);

/*
 *  Open plot data file (.pdt)
 */

/*
    sprintf(plot_data, "%s%s.pdt", DATA_DIR,
                                Dm_Address->display[disp_num].plot_name);
*/
    strcpy(plot_data, Dm_Address->display[disp_num].plot_name);
    strcat(plot_data, ".pdt");
```

```
    printf("Processing data plot file %s\n", plot_data);

    plot_fp = Plot_fp[empty] =open(plot_data, O_RDWR | O_CREAT | O_TRUNC, 0666);

    if (plot_fp == -1) {
        fprintf(stderr,"Error %d on creating the plot data file",errno);
         return(-1);
    }

/*
 *  Just put in blanks for header
 */

    strcpy(buffer, "
          ");

    write(plot_fp, buffer, 80);

/*
 *  Assume all data is double and write decom info for
 *  all non-time msids.
 *      size = 12   (4 for status)
 *      length = 8
 *      number samples = 1
 *      attribute = D
 *      error = 0
 */

    size = 12;
    length = 8;
    num_samps = 1;
    attribute = 'D';
    error = NULL;

    msid_ptr = Pmsids[empty];

    for (i=0; i<Num_msids[empty]; i++) {
            write(plot_fp, (char *)&size, 4);
            write(plot_fp, (char *)&length, 4);
            write(plot_fp, &num_samps, 2);
            write(plot_fp, &attribute, 1);
            write(plot_fp, &error, 1);
            write(plot_fp, buffer, 12);
    }

    return(0);

}


/*****************************************************************************
 * Function:    stop_plot
 * Purpose:
 *      To find the effective plot name in the plot name list,
 *      clear the plot name, and free any allocated memory for
 *      that plot record.
 *
 * Notes:
 *      This routine does not check to see if any other DM task is using
 *      the plot.
 *****************************************************************************/

stop_plot(disp_num)
    short disp_num;
```

```
{
    int i;

    i=0;
    while (i<MAX_PLOTS &&
            strcmp(Plot_name[i], Dm_Address->display[disp_num].plot_name))
        i++;

    if (i < MAX_PLOTS) {
        Plot_name[i][0] = 0;
        free(Pmsids[i]);
        Pmsids[i] = NULL;
        free(Pheader[i]);
        Pheader[i] = NULL;
        close(Plot_fp[i]);
        Plot_fp[i] = -1;
    }

    return(0);
}


/*****************************************************************************
 * Function:    read_plot_file
 * Purpose:
 *      To read the plot file into memory for use by the
 *      random generator.
 *****************************************************************************/

read_plot_file(plot_name, empty)
    char *plot_name;
    int empty;
{
    FILE    *fp;

    struct plot_hdr *header_ptr;    /* plot hdr file                        */
    struct axis_info    *axis_ptr;  /* plot axis infomation                 */
    struct msid_info    *msid_ptr;  /* plot msid infomation                 */
    struct lim_lines    *nline_ptr; /* plot nominal line infomation         */
    struct lim_lines    *lline_ptr; /* plot limit line infomation           */
    struct plot_pts *nline_pts_ptr; /* plot nominal point pairs pointer     */
    struct plot_pts *lline_pts_ptr; /* plot limit point pairs pointer       */

    short   upd_rate;
    short   version;        /* local variable for software version  */
    short   xaxes_num;      /* nbr of xaxes records                 */
    short   yaxes_num;      /* nbr of yaxes records                 */
    short   plot_hori;      /* the horizontal font size             */
    short   plot_vert;      /* the vertical font size               */
    short   match;
    short   restricted;
    short   color;          /* temp holder for color # read from DDF file */

    unsigned    size;       /* size for memory allocation           */
    int i, j, k, w, m, n, p;    /* loop count variable for rec. header  */
    int     access_rs;  /* access restriction code              */
    int     graph_num;      /* local variable for total nbr of graph. rec */
    int     char_num;       /* local variable for total nbr of char */
    int     msid_num;       /* nbr of msid records                  */
    int     actual_msids;   /* nbr of actual msid records       */
    int     nline_num;      /* nbr of nominal line records          */
    int     lline_num;      /* nbr of limit line records            */
    int     total_nbr_records;  /* total nbr of plot records            */
    int     total_nbr_axes;
```

```
    int     name_len;
    int     x, y;

    char    temp[15];       /* use to try out character stuff    */
    char    sample[4];
    char    plot_style[5];  /* the character style */
    char    plot_fn[50];

    D(printf("START read_plot_file\n"));

/*
    strcpy(plot_fn, DATA_DIR);
    strcat(plot_fn, plot_name);
*/
    strcpy(plot_fn, plot_name);
    strncat(plot_fn, ".plt\0", 5);

    if ((fp = fopen(plot_fn,"r")) == NULL) {
        fprintf(stderr,"Error %d on reading plot file %s",errno,plot_fn);
        return(-1);
    }

    printf("Reading plot file %s\n", plot_fn);

/*
 *  Read the software version. If correct version continue processing by read-
 *  ing in the plot file information.
 */

    fscanf (fp, "%hd", &version);

    fscanf (fp, "%*51c");
    fscanf (fp, "%hd", &xaxes_num);
    fscanf (fp, "%hd", &yaxes_num);
    fscanf (fp, "%d", &msid_num);
    fscanf (fp, "%d", &actual_msids);
    fscanf (fp, "%d", &nline_num);
    fscanf (fp, "%d", &lline_num);
    fscanf (fp, "%hd", &upd_rate);
    fscanf (fp, "%d", &access_rs);

    if (version >= 3)
        fscanf (fp, "%*5c");

    total_nbr_records =
                xaxes_num + yaxes_num + actual_msids + nline_num + lline_num;

    if (total_nbr_records == 0) {
        fprintf(stderr, "There are no plot records ");
        fclose (fp);
        return (-1);
    }

/*
 *  Reassign pointer to beginning of file
 */

    rewind (fp);

/*
 *  Set up local pointer to beginning of plot header file and read header
 */

    header_ptr = (struct plot_hdr  *) calloc (1, sizeof (struct plot_hdr));
```

```
    if (header_ptr == NULL) {
        fprintf(stderr, "Error %d allocating plot header memory", errno);
        fclose (fp);
        return (-1);
    }

    Pheader[empty] = header_ptr;

    fscanf (fp, "%hd", &version);
    fscanf (fp, "%*51c");
    fscanf (fp, "%d", &header_ptr->xaxes_num);
    fscanf (fp, "%d", &header_ptr->yaxes_num);
    fscanf (fp, "%d", &header_ptr->msid_num);
    fscanf (fp, "%d", &header_ptr->actual_msids);
    fscanf (fp, "%d", &header_ptr->nline_num);
    fscanf (fp, "%d", &header_ptr->lline_num);
    fscanf (fp, "%hd", &header_ptr->upd_rate);
    fscanf (fp, "%hd", &header_ptr->access_rs);

    if (version >= 3)
        fscanf (fp, "%*5c");

/*
    if (msid_num > 0) {
        plot_info_ptr->plt_decom = (struct shm_decom *)
            calloc(plot_info_ptr->header->msid_num, sizeof(struct shm_decom));

        if (plot_info_ptr->plt_decom == NULL) {
            fprintf(stderr, "Error on allocating memory for plot decom");
            fclose(fp);
            return (-1);
        }
    }
*/

/*
 *  Read in the plot definition file msid records and store them into memory.
 */

    if (actual_msids > 0) {

        msid_ptr = (struct msid_info *)
                        calloc(actual_msids, sizeof(struct msid_info));

        if (msid_ptr == NULL) {
            fprintf(stderr, "Error %d allocating plot msid memory", errno);
            fclose (fp);
            return (-1);
        }

        Pmsids[empty] = msid_ptr;

        for (j = 0; j < actual_msids; j++) {
            fscanf (fp, "%hd", &msid_ptr->msid_indx);
            msid_ptr->msid_indx = j;
            fscanf (fp, "%s", msid_ptr->msid_name);
            fscanf (fp, "%s", sample);

            if (sample[0] != 'L')
                msid_ptr->sample = atoi (sample);
            else
                msid_ptr->sample = -1;
```

```
            fscanf (fp, "%s", msid_ptr->data_src);

            /* skip the ppl file and occr numbers */

            if (version >= 3)
                fscanf (fp, "%*10c");

            fscanf (fp, "%s", temp);
            msid_ptr->xory_axis = temp[0];
            fscanf (fp, "%d", &msid_ptr->axis_num);
            fscanf (fp, "%s", msid_ptr->plot_msid);
            fscanf (fp, "%s", temp);
            msid_ptr->plot_type = temp[0];
            fscanf (fp, "%d", &msid_ptr->line_type);
            fscanf (fp, "%f", &msid_ptr->line_width);
            fscanf (fp, "%s", msid_ptr->plot_char);
            fscanf (fp, "%s", plot_style);
            fscanf (fp, "%hd", &plot_hori);
            fscanf (fp, "%hd", &plot_vert);

/* RLK 9/12/90  More font stuff to fix.
            DBfontnum (plot_style, plot_hori, plot_vert, &msid_ptr->plot_font);
*/

            fscanf (fp, "%hd", &msid_ptr->icon_indx);
            fscanf (fp, "%s", temp);
            msid_ptr->plot_conn = temp[0];
            fscanf (fp, "%hd", &color);
            fscanf (fp, "%d", &msid_ptr->stat_flag);
            fscanf (fp, "%d", &msid_ptr->miss_flag);
            fscanf (fp, "%hd", &color);
            fscanf (fp, "%hd", &color);
            fscanf (fp, "%hd", &color);
            fscanf (fp, "%hd", &color);
            fscanf (fp, "%hd", &color);
            fscanf (fp, "%d", &msid_ptr->oper_type);
            fscanf (fp, "%f", &msid_ptr->oper_width);
            fscanf (fp, "%d", &msid_ptr->crit_type);
            fscanf (fp, "%f", &msid_ptr->crit_width);
            msid_ptr->pair_ptr = NULL;
            msid_ptr->first_pt = YES;

            msid_ptr++;

        }   /* End of -for- (total nbr of msids) */

/*
 *      Set the pair index pointers
 */

        msid_ptr = Pmsids[empty];

        for (i = 0; i < actual_msids; i++) {

            /* the current msid is represented by msid_ptr + i  */

            if ((msid_ptr + i)->pair_ptr == NULL) {
                match = NO;
                k = i + 1;

                while (match == NO && k < actual_msids) {

                    if ((msid_ptr + k)->pair_ptr == NULL) {
```

```
                    if (    !strcmp((msid_ptr + i)->msid_name,
                                    (msid_ptr + k)->plot_msid)
                        &&  !strcmp((msid_ptr + i)->plot_msid,
                                    (msid_ptr + k)->msid_name)) {
                        (msid_ptr + i)->pair_ptr = msid_ptr + k;
                        (msid_ptr + k)->pair_ptr = msid_ptr + i;
                        match = YES;
                    }
                    else
                        k++;
                }
                else
                    k++;

            }   /* end while */

        }       /* end of if ... == NULL */

    }           /* end of -for- (i) */

}   /* end -if- (actual msid > 0) */

/*
 *  Read in the plot definition file axis records and store them into memory.
 */

    total_nbr_axes = xaxes_num + yaxes_num;

    if (total_nbr_axes > 0) {

/*
        axis_ptr = (struct axis_info *)
                        calloc(total_nbr_axes, sizeof(struct axis_info));

        if (axis_ptr == NULL) {
            tui_msg(M_YELLOW, "Error %d allocating plot axis memory", errno);
            fclose (fp);
            return (-1);
        }
*/

        x = 0;
        y = 0;

        for (m = 0; m < total_nbr_axes; m++) {
            fscanf (fp, "%s", temp);

            if (temp[0] == 'X') {
                axis_ptr = &Paxis[empty][X][x];
                x++;
            } else {
                axis_ptr = &Paxis[empty][Y][y];
                y++;
            }

            axis_ptr->axis_xory = temp[0];
            fscanf (fp, "%d", &axis_ptr->axis_num);
            fscanf (fp, "%hd", &axis_ptr->axis_type);
            fscanf (fp, "%s", temp);
            axis_ptr->scal_type = temp[0];
            fscanf (fp, "%d", &axis_ptr->end_code);
            fscanf (fp, "%hd", &axis_ptr->axis_pos);
            fscanf (fp, "%hd", &color);
```

```
            fscanf (fp, "%s", axis_ptr->low_scale);

/*
            if (axis_ptr->scal_type == 'T')
                axis_ptr->low_value = (double) DBp_atimei(axis_ptr->low_scale);
            else
*/
                sscanf(axis_ptr->low_scale, "%lf", &axis_ptr->low_value);

            axis_ptr->org_low_val = axis_ptr->low_value;

            fscanf (fp, "%s", axis_ptr->high_scal);

/*
            if (axis_ptr->scal_type == 'T')
                axis_ptr->high_value = (double) DBp_atimei(axis_ptr->high_scal);
            else
*/
                sscanf (axis_ptr->high_scal, "%lf", &axis_ptr->high_value);

            axis_ptr->org_high_val = axis_ptr->high_value;

            fscanf (fp, "%s", temp);
            axis_ptr->auto_flag = temp[0];
            fscanf (fp, "%hd", &axis_ptr->grad_vals);
            fscanf (fp, "%s", temp);
            axis_ptr->vis_flag = temp[0];
            fscanf (fp, "%s", temp);
            axis_ptr->grid_flag = temp[0];
            fscanf (fp, "%hd", &axis_ptr->grid_gran);
            fscanf (fp, "%hd", &axis_ptr->grid_type);
            fscanf (fp, "%hd", &color);
            fscanf (fp, "%hd", &axis_ptr->maj_ticks);
            fscanf (fp, "%hd", &axis_ptr->min_ticks);

            axis_ptr->axis_active = YES;
        }
    }

    fclose(fp);

    D(printf("END read_plot_file\n"));

    return(actual_msids);
}


/*****************************************************************************
 * Function:    gen_and_write_data
 * Purpose:
 *      To generate random data based on the axis scale values and
 *      write it to the plot data file.
 *****************************************************************************/

double gen_and_write_data(msid_ptr, axis_ptr, increment, plot)
    struct msid_info *msid_ptr;
    struct axis_info *axis_ptr;
    double increment;
    int plot;
{
    long low, high, tmp;
    double d_val;
    long status;
    static short indx = 0;
```

```
    int low_flag;
    float mod_factor;

    low = (long) axis_ptr->low_value;
    high = (long) axis_ptr->high_value;

    low_flag = 0;
    if (low > high) {
        tmp = low;
        low = high;
        high = tmp;
        low_flag = 1;
    }

    status = 0;

/*
    if (axis_ptr->axis_xory == Overscale_axis
            && axis_ptr->axis_num == Overscale_num)
        mod_factor = 1.5 * ((high - low) / (double)(CYCLES * .60));
    else
        mod_factor = 1.5 * ((high - low) / (double) CYCLES);

    if (mod_factor > 1.0)
        increment += (double) (random() % (long)mod_factor);
    else
        increment += (double) (random() % (long)(mod_factor * 1000.0)) / 1000.0;
*/

/*
 *  plot low value to high value

    if (low > 0) {
        if (low_flag)
            d_val = (double) high - increment;
        else
            d_val = (double) low + increment;
    } else
            d_val = (double) low + increment;
 */

/*
*/
    d_val = (double) (random() % (high - low + 1) + low);

    write(Plot_fp[plot], (char *)&indx, sizeof(short));
    write(Plot_fp[plot], (char *)&status, sizeof(long));
    write(Plot_fp[plot], (char *)&d_val, sizeof(double));

    return(increment);
}

/*****************************************************************************
 * FP interrupt handler
 *****************************************************************************/

int control_fpe()
{
    signal(SIGFPE, control_fpe);
}


/*****************************************************************************
 * Function:    main
```

```
*****************************************************************************/

main()
{
    int x,y;
    int i, j;
    struct msid_info *msid_ptr;
    struct axis_info *axis_ptr;
    double increment[MAX_PLOTS][2][20];
    int idx;
    int dm_shm_id;
    struct timeb cur_time, update;          /* current system time          */
    double new_time;                        /* holder for the conv after ftime*/
    double last_time;                       /* holder for the conv of last_upd*/
    int error;
    short disp_num;

/*
 *  Set up floating point exception handler
 */

    signal(SIGFPE, control_fpe);

    printf("Plot Data Feed task starting\n");

/*
 *  Initialize timer
 */

    update.time = 0;

/*
 *  Attach to DM shared memory
 */

    dm_shm_id = shmget(DM_SHM_KEY,sizeof(struct dm_shmemory), 0666);

    if (dm_shm_id == -1) {
        printf("There was an error %d on the DM SHM get",errno);
        return(-1);
    }

    Dm_Address = (struct dm_shmemory *) shmat(dm_shm_id,0,0);

    if ((int)Dm_Address == -1) {
        printf("There was an error %d on the DM SHM attach",errno);
        return(-1);
    }

/*
 *  While the DH halt flag is not set, loop
 */

    while (!Dm_Address->process.dh_not_halted) {

/*
 *      Loop through display numbers to check plot flags in DM shared memory.
 *      If detect command flag, branch to command processing routine.
 */

        for (disp_num = 0; disp_num < MAX_DISP; disp_num++) {

            if (Dm_Address->display[disp_num].disp_active
                    && Dm_Address->display[disp_num].dh_plot) {
```

```
                if (Dm_Address->display[disp_num].action == STRT_PLOT) {
                    error = start_plot(disp_num);
                    if (error == 0)
                        Dm_Address->display[disp_num].dh_plot_ack = YES;

                } else {
                    error = stop_plot(disp_num);
                    if (error == 0)
                        Dm_Address->display[disp_num].dh_plot_ack = YES;
                }

                Dm_Address->display[disp_num].dh_plot = NO;
            }
        }

/*
 *      Update the current time.
 */

        ftime(&cur_time);
        last_time = (update.time * 1000) + update.millitm;
        new_time = (cur_time.time * 1000) + cur_time.millitm;

/*
 *      If time to update the plot data files, generate random
 *      data for each msid for each plot.

        if ((new_time - last_time) >= UPDATE_RATE) {
 */

            for (i=0; i<MAX_PLOTS; i++) {
                msid_ptr = Pmsids[i];

                if (msid_ptr != NULL) {

                    for (j=0; j<Num_msids[i]; j++) {

                            if (msid_ptr->xory_axis == 'X') {
                                axis_ptr = &Paxis[i][X][(msid_ptr->axis_num-1)];
                                idx = X;
                            } else {
                                axis_ptr = &Paxis[i][Y][(msid_ptr->axis_num-1)];
                                idx = Y;
                            }

                            increment[i][idx][j] = gen_and_write_data
                                (msid_ptr, axis_ptr, increment[i][idx][j], i);

                        msid_ptr++;
                    }
                }

            }

            update.time = cur_time.time;
            update.millitm = cur_time.millitm;

/*
        }   /* end of time check */

/*
 *  Sleep for a few seconds
 */
```

```
        sleep(1);

    }   /* end of while (not halted) loop */

/*
 *  Detach from shared memory, close plot data files, and exit
 */

    for (i=0; i<MAX_PLOTS; i++) {
        if (Plot_name[i][0] != 0) {
            close(Plot_fp[i]);
        }
    }

    shmdt(Dm_Address);

    printf("Plot Data Feed exiting\n");
}
```