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 I

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

6220 CULEBRA ROAD • POST OFFICE DRAWER 28510 • SAN ANTONIO, TEXAS, USA 78228-0510 • (512) 684-5111 • TELEX 244846

November 30, 1990

Ms. Linda Uljon
Building 30, DK32
NASA - Johnson Space Center
Houston, Texas 77058

Subject: Delivery of the Final Report for Continuation of Research into Software for Space Operations Support; NASA Grant NAG 9-388; SwRI Project Number 05-2984

Dear Ms. Uljon,

Enclosed with this letter is the final report pertaining to the research performed for workstation executive technology applicable to environments such as the upgraded Mission Control Center. This document includes all code and information for the four research efforts performed during the course of this grant, including the HISDE conversion, the Real-time widget prototype, the X Windows performance evaluation, and the X Windows/Motif-based Display Manager prototype. If you have any questions or comments, please call me at (512) 522-3437.

Sincerely,

Mark D. Collier
Senior Research Analyst
Software Engineering Section
Data Systems Department

Approved:

Melvin A. Schrader
Director
Data Systems Department

MDC:vc

Enclosures

cc: Nancy L. Martin
Ronnie Killough
Susan B. Crumrine
William A. Bayliss
Thomas J. Purk, NASA-JSC, BG 211
NASA Scientific and Technical Information Facility (2 copies)

Table of Contents

1.0 INTRODUCTION

This document serves as the final report describing the activity on NASA Grant NAG 9-388, which is entitled "Continuation of Research in Software for Space Operations Support". The purpose of this grant was to continue the research direction defined for NASA Grant NAG 9-269, during which SwRI developed a prototype workstation executive called the Hardware Independent Software Development Environment (HISDE). The research direction of this grant was to research and evaluate software technologies relevant to workstation executives and to use HISDE as a test bed for prototyping efforts.

This document will describe the background for the research grant and describe all research performed.

2.0 RESEARCH BACKGROUND

During the past few years and continuing in the future, many centralized computing installations are migrating to environments characterized by distributed processing. This migration is driven primarily by the low cost and high performance delivered by state-of-the-art graphic workstations. Such an environment normally consists of a large number of workstations which are in turn connected via one or more high-speed networks.

Although a workstation-based distributed processing environment offers many advantages, it also introduces a number of new concerns. One problem is that engineering-class workstations most commonly use the UNIX operating system, which is difficult for computer novices to use effectively. Also, connecting a large number of workstations and expecting them to work as an integrated system is not easily achieved. The introduction of so many separate processors makes configuration management and security a real concern. In fact, the very flexibility which is inherent in workstations often becomes a problem. This is especially true for real-time critical command and control systems in which a failure or security break could have disastrous results.

In order to solve these problems, allow the environment to function as an integrated system, and present a functional development environment to application programmers, it is necessary to develop an additional layer of software. This "executive" software integrates the system, provides real-time capabilities, and provides the tools necessary to support the application requirements. Such an executive will be required for use in evolving systems such as the ground-based control centers planned at Johnson Space Center. These command and control environments will use a distributed processing architecture to provide real-time processing of telemetry and command data.

2.1 NASA Grant NAG 9-269 Background

For NASA Grant NAG 9-269, which was entitled "Research in Software for Space Operations Support", SwRI developed the HISDE prototype to serve as proof-of-concept for a hardware-independent workstation executive. The HISDE prototype introduced a number of advanced software technologies and concepts including:

- Exclusive use of software standards:
  - SVID UNIX Operating System
  - X Windows
  - GKS and PHIGS
  - ISO OSI Communications

Through the use of standards, HISDE was easily ported across multiple vendor workstations.

- Open use of UNIX - through a more flexible design and configuration management scheme, HISDE provided access to the UNIX file system via the familiar UNIX command line interface.
- CM Manager Workstation - this concept involves a workstation to which all user applications are loaded with certified libraries prior to being mission certified and uploaded to a configuration management host.

The purpose of this continuation grant was to research software technologies relevant to workstation executives. The HISDE prototype was used as a test bed for prototyping and practical evaluation of identified technologies.

3.0 RESEARCH PERFORMED

The research performed on this grant was directed towards the introduction of new X Windows software concepts and technology into workstation executives and related applications. The four research efforts performed include:

- Research into the usability and efficiency of Motif - this effort consisted of converting the existing Athena widget-based HISDE user interface to Motif. This research demonstrated the usability of Motif and provided insight into the level of effort required to translate an application from one widget set to another.
- Prototype a real-time data display widget - this effort consisted of researching methods for and prototyping the selected method of displaying textual values in an efficient manner. The prototype widget can be used by NASA in special purpose user applications and in system applications such as the Display Manager, which must display large amounts of text in an efficient manner.
- X Windows performance evaluation - this effort consisted of a series of performance measurements which demonstrated the ability of low-level X Windows to display textual information. The performance of X Windows was compared to the performance of similar operations performed in Graphic Kernel System (GKS) calls.
- Convert the Display Manager to X Windows/Motif - the Display Manager is the application used by NASA for data display during operational mode. This application primarily uses GKS for data display and user interface, which is somewhat inefficient and difficult for the basis of a user interface. SwRI developed a prototype of the Display Manager which only used X Windows and Motif for data display and user interface.

For more information on each of these efforts, refer to the following four sections of the document. Each section provides a description of the research effort and includes all relevant code and/or performance data.

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

CONTINUATION OF RESEARCH IN SOFTWARE FOR SPACE OPERATIONS SUPPORT

RESEARCH INTO THE USABILITY AND EFFICIENCY OF 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

The Open Software Foundation (OSF) distributes an X Windows-based graphic user interface system called Motif. Motif has become a standard and is the default X Windows-based user interface for a large number of workstations available at this time. Motif includes a highly functional widget set, a window manager, a user interface language, and a style guide.

As an exercise to demonstrate Motif, both in terms of appearance and usability from a programming standpoint, SwRI converted the user interface for the Hardware Independent Software Development Environment (HISDE) to Motif. This involved replacing the existing interfaces to the MIT Athena widget set with the corresponding interfaces for Motif.

2.0 RESEARCH GOALS

- Demonstrate the "look and feel" of Motif within the context of an application which is familiar to NASA (HISDE).
- Use Motif and formulate opinions on its usability, performance, and ease of transition from other widget sets.

3.0 RESEARCH DETAILS

When HISDE was developed, the introduction of improved widget sets was anticipated. The majority of the access to widget-specific functions was through a set of library functions which both made it easier to use the widgets and hid the details of the widgets from the actual application code. Therefore, the first step in converting from Athena to Motif widgets was to update these library functions. This was a relatively simple process which included converting user interface functions to Motif widgets and supporting functions.

The next step was to convert the actual application level code. Due to the amount of time available for this effort, only a subset of the HISDE user interface clients were actually converted. The clients which were converted include the following:

- h_advisory - advisory/message display client.
- h_bulletin - host advisory display client.
- h_cm_menu - configuration management user interface client.
- h_cmd - command interface client.
- h_help - help client.
- h_info - system information client.
- h_info_a - system information client.
- h_login - login client.
- h_logout - logout client.
- h_menu_edit - menu editor client.
- h_msg_look - advisory/message browse client.
- h_pbi_edit - PBI editor client.
- h_talk - remote communications client.

Due to the time available for this conversion effort, the following clients were not completely converted to Motif.

- h_menu - this client implements user-defined menus, which were not available in the Athena widget set. Motif includes a rich collection of menus, therefore rendering this client obsolete.
- h_cm_status - this client is the user interface for the CM Manager Workstation. Conversion of this client was begun, but not completed.
- h_pbi - this client displays push buttons used to emulate Push Button Indicators (PBI's). Conversion of this client was begun, but not completed.
- h_status - this client provides system status displays. Conversion of this client was begun, but not completed. This client depends on a widget (the load widget) from the Athena widget set. Conversion of this client to Motif requires integration of widgets from two different sets.

Because the source code for these clients was not in a finished state, the code is not included in this document. The actual interim code is however present on the delivered set of tapes.

Note that Motif includes a User Interface Language (UIL) which purportedly simplifies development of user interfaces. After review of this language it was determined that direct use of the Motif widget set was more efficient.

4.0 RESEARCH CONCLUSIONS

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

- Motif provides a highly functional and robust set of widgets. The Motif widgets are superior to both Athena and Hewlett Packard widgets in terms of functionality, usability, documentation, and reliability.
- Transition from one set of widgets to another is a relatively simple process. This is especially so if the actual interfaces to the widgets are isolated from application code. If this is not done, transition will be possible, but will require a very large number of tedious changes.
- The Motif widget set, due to its large size, greatly increases the size of an executable program. In an operating system which does not support shared libraries, a copy of the widget set will be present in every application. This will quickly use up memory and swap space.
- The Motif widget set provides acceptable performance. In applications demanding high performance display of graphics, Motif could be used to develop the user interface and to provide windows in which lower level functions would be used for high performance graphics display.
- Motif provides the concept of "gadgets", which are an alternative for many simple widgets such as labels, push buttons, and toggle buttons. Unlike a widget, a gadget does not require display of an individual window and therefore is faster and more efficient (at a cost of less functionality and configurability).

5.0 ATTACHMENTS

The following pages contain the actual code for the translated HISDE user interface. The code and related files which are present include:

- User interface library Makefile and code.
- Makefiles and code for each of the converted clients:
  - h_advisory.
  - h_bulletin.
  - h_cm_menu.
  - h_cmd.
  - h_help.
  - h_info.
  - h_info_a.
  - h_login.
  - h_logout.
  - h_menu_edit.
  - h_msg_look.
  - h_pbi_edit.
  - h_talk.

ATTACHMENT 1 - User Interface Library

```
#############################################################################################
# Makefile for the HISDE user interface library.                                            #
#############################################################################################

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

TARGET      = libui.a

#
# Initialize include and library search paths to include current directory and the
# HISDE directories.
#

INCDIR      = /hisde/src/include
LIBDIR      = /usr/lib
INCDIRS     = -I. -I$(INCDIR)

#
# Define the libraries to search.  This includes the HISDE library and all X
# windows libraries.
#

LIBRARIES   = -lhisde -lXaw -lXt -lX11

#
# Define the compiler and linker flags.
#

CFLAGS      = -O $(INCDIRS)
LDFLAGS     = -O

#
# Define any programs and options to use.
#

AR          = ar rv
RM          = rm -f
RANLIB      = ranlib

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

OBJS        =\
    create_label.o\
    create_cmd.o\
    create_cas.o\
    create_form.o\
    create_text.o\
    create_tog.o\
    display_msg.o\
    ld_text_wid.o\
    upd_text_wid.o\
    ins_text_wid.o\
    clr_text_wid.o\
    get_text_wid.o\
    get_txts_wid.o\
    get_txtp_wid.o\
    init_list.o\
    bad_syntax.o

#
```

```
# Define all header files required.
#

HDRS        =\
    $(INCDIR)/hisde.h

#
# Make the target.
#

all:        $(TARGET)

$(TARGET):  $(OBJS)
    $(AR) $(LIBDIR)/$(TARGET) $(OBJS)
    $(RANLIB) $(LIBDIR)/$(TARGET)

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

```
/*****************************************************************************
*                                                                            *
* MODULE NAME AND FUNCTION ( bad_syntax )                                    *
*                                                                            *
* This function is called to output a warning to the system message client to inform the *
* user that he has specified an invalid command syntax.  This function formats the mes- *
* sage, outputs it, and calls (exit) to terminate the client.                *
*                                                                            *
*                                                                            *
* SPECIFICATION DOCUMENTS:                                                   *
*                                                                            *
*      /hisde/req/requirements                                               *
*      /hisde/design/design                                                  *
*                                                                            *
*                                                                            *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                        *
*                                                                            *
*      Mark D. Collier - Software Engineering Section                        *
*                        Data System Science and Technology Department       *
*                        Automation and Data Systems Division                *
*                        Southwest Research Institute                        *
*****************************************************************************/


#include <hisde.h>


int bad_syntax ( syntax )

    char    *syntax;                /* String containing the correct syntax of the
                                     * calling client.  It will be output to the system
                                     * message client.
                                     */

{
/*
 *  Define the string which will be formatted with the message.
 */

    static char message[MAX_MESSAGE_LENGTH] = "Invalid Syntax - Try: ";

/*
 *  Concatenate the correct syntax string to the warning string and output to the
 *  system message client.  Upon return, exit from the client.
 */

    h_message ( MSG_WARNING, strcat ( message, syntax ) );
    exit ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION ( clear_text_widget )
 *
 * This function is used to clear all text within a text widget.
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *      /hisde/req/requirements
 *      /hisde/design/design
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *      Mark D. Collier - Software Engineering Section
 *                        Data System Science and Technology Department
 *                        Automation and Data Systems Division
 *                        Southwest Research Institute
 *****************************************************************************/


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


void clear_text_widget ( widget )

    Widget  widget;                 /* Pointer to the text widget which will be cleared
                                     * of all text.
                                     */
{
/*
 *  Clear all text from the widget.
 */

    XmTextSetString ( widget, "" );
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION ( create_cascade )                               *
 *                                                                           *
 * This function is called to create a MOTIF cascade widget.                 *
 *                                                                           *
 *                                                                           *
 * SPECIFICATION DOCUMENTS:                                                  *
 *                                                                           *
 *      /hisde/req/requirements                                              *
 *      /hisde/design/design                                                 *
 *                                                                           *
 *                                                                           *
 * ORIGINAL AUTHOR AND IDENTIFICATION:                                       *
 *                                                                           *
 *      Mark D. Collier - Software Engineering Section                       *
 *                        Data System Science and Technology Department      *
 *                        Automation and Data Systems Division               *
 *                        Southwest Research Institute                       *
 *****************************************************************************/


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


Widget create_cascade ( instance, parent, menu, label )

                                    /* This function returns the return value of the
                                     * XtCreateManagedWidget function call.  This will
                                     * be a pointer to a widget.
                                     */

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

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

    Widget  parent,                 /* The parent widget to which the command widget will
                                     * be attached.
                                     */

            menu;                   /* Menu which will be activated when the cascade is
                                     * selected.
                                     */

{
/*
 *  Define the array which will contain all arguments required to create the command
 *  widget.
 */

    Widget          widget;         /* Pointer to the created widget.
                                     */

    Arg             args[ 1 ];      /* Argument list for cascade widget.
                                     */

    register int    count = 0;      /* Counts the number of arguments initialized.
                                     */

/*
 *  attach it to the parent, and initialize all arguments.
 */

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

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

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

    return ( widget );
}
```

```
/*****************************************************************************
* MODULE NAME AND FUNCTION ( create_command )                                *
*                                                                            *
* This function is called to create a command widget.                        *
*                                                                            *
*                                                                            *
* SPECIFICATION DOCUMENTS:                                                   *
*                                                                            *
*      /hisde/req/requirements                                               *
*      /hisde/design/design                                                  *
*                                                                            *
*                                                                            *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                        *
*                                                                            *
*      Mark D. Collier - Software Engineering Section                        *
*                        Data System Science and Technology Department       *
*                        Automation and Data Systems Division                *
*                        Southwest Research Institute                        *
*****************************************************************************/


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


Widget create_command ( instance, parent, label, callback )

                                    /* This function returns the return value of the
                                     * XtCreateManagedWidget function call.  This will
                                     * be a pointer to a widget.
                                     */

    char    *instance,              /* The instance name of the widget.  It uniquely
                                     * defines the widget.
                                     */
            *label;                 /* The string which this command widget will display.
                                     */

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

    XtCallbackList  callback;       /* Specifies an array containing the list of func-
                                     * tions called upon command callback.  It may be
                                     * NULL if no functions are present.
                                     */
{
    Widget          widget;         /* Pointer to the created widget.
                                     */

    XmString        string;         /* Compound string to which the label is converted.
                                     */

    Arg             args[ 2 ];      /* Argument list used to initialize widget resources.
                                     */

    register int    count = 0;      /* Incremented each time a argument is initialized
                                     * in the (args) array.  When the widget is created,
                                     * this value which indicate the number of arguments.
                                     */

/*
 *  Convert the label to a compound string and save 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 memory allocated for the compound string.
 */

    XtManageChild ( widget = XmCreatePushButton ( parent, instance, args, count ) );
    XmStringFree ( string );

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

    if ( callback )
        XtAddCallbacks ( widget, XmNactivateCallback, callback );

    return ( widget );
}
```

```
/*****************************************************************************
*                                                                            *
* MODULE NAME AND FUNCTION ( create_form )                                   *
*                                                                            *
* This function is called to create a MOTIF form widget.                     *
*                                                                            *
*                                                                            *
* SPECIFICATION DOCUMENTS:                                                   *
*                                                                            *
*      /hisde/req/requirements                                               *
*      /hisde/design/design                                                  *
*                                                                            *
*                                                                            *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                        *
*                                                                            *
*      Mark D. Collier - Software Engineering Section                        *
*                        Data System Science and Technology Department       *
*                        Automation and Data Systems Division                *
*                        Southwest Research Institute                        *
*****************************************************************************/


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

Widget create_form ( instance, parent )

                                /* This function returns the return value of the
                                 * XtCreateManagedWidget function call.  This will
                                 * be a pointer to a widget.
                                 */

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

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

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

    XtManageChild ( widget = XmCreateForm ( parent, instance, NULL, 0 ) );

    return ( widget );
}
```

```
/*****************************************************************************
* MODULE NAME AND FUNCTION ( create_label )
*
* This function is called to create a MOTIF label widget.
*
*
* SPECIFICATION DOCUMENTS:
*
*      /hisde/req/requirements
*      /hisde/design/design
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*      Mark D. Collier - Software Engineering Section
*                        Data System Science and Technology Department
*                        Automation and Data Systems Division
*                        Southwest Research Institute
*****************************************************************************/


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


Widget create_label ( instance, parent, label )

                                  /* This function returns the return value of the
                                   * XtCreateManagedWidget function call.  This will
                                   * be a pointer to a widget.
                                   */

    char    *instance,            /* The instance name of the widget.  It uniquely
                                   * defines the widget.
                                   */
            *label;               /* The string which this label widget will display.
                                   */
    Widget  parent;               /* The parent widget to which the label widget will
                                   * be attached.
                                   */
{
    Arg args[ 2 ];                /* Argument list used to initialize the widget
                                   * resources.
                                   */

    Widget      widget;           /* Pointer to the created widget.
                                   */

    XmString    string;           /* Points to the compound string created for the
                                   * label.
                                   */

    register int    count = 0;    /* Counter set to the number of arguments initialized.
                                   */

/*
 *  Initialize a compound string and set in the resource list.
 *  lists.
 */

    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 = XmCreateLabel ( parent, instance, args, count ) );
    XmStringFree ( string );

    return ( widget );
}
```

```
/*****************************************************************************
* MODULE NAME AND FUNCTION ( create_text )
*
* This function is called to create a MOTIF text widget.
*
*
* SPECIFICATION DOCUMENTS:
*
*      /hisde/req/requirements
*      /hisde/design/design
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*      Mark D. Collier - Software Engineering Section
*                        Data System Science and Technology Department
*                        Automation and Data Systems Division
*                        Southwest Research Institute
*****************************************************************************/


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


Widget create_text ( instance, parent, text, scrolled, mode, edit_flag )

                                    /* This function returns the return value of the
                                     * XtCreateManagedWidget function call.  This will
                                     * be a pointer to a widget.
                                     */

    char    *instance,              /* The instance name of the widget.  It uniquely
                                     * defines the widget.
                                     */
            *text;                  /* The ascii text which will be displayed in the
                                     * text widget.
                                     */

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

    int     mode,                   /* Indicates whether the widget will be single or
                                     * multiple lines:
                                     *
                                     *   XmSINGLE_LINE_EDIT
                                     *   XmMULTI_LINE_EDIT
                                     */
            scrolled,               /* Indicates whether or not the data can be scrolled.
                                     */
            edit_flag;              /* Indicates whether or not the widget can be edited.
                                     */
{
    Widget          widget;         /* Pointer to the created widget.
                                     */

    Arg             args[ 3 ];      /* Argument list used to initialize the widget
                                     * resources.
                                     */

    register int    count = 0;      /* Used to count the number of arguments initialized.
                                     */
```

```
/*
 *  Initialize the widget text (not a compound string), the line size mode, and the
 *  edit mode.
 */

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

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

    if ( scrolled )
        widget = XmCreateScrolledText ( parent, instance, args, count );
    else
        widget = XmCreateText         ( parent, instance, args, count );
    XtManageChild ( widget );

    return ( widget );
}
```

```
/*****************************************************************************
* MODULE NAME AND FUNCTION ( create_toggle )
*
* This function is called to create a toggle button widget.
*
*                                                                            *
* SPECIFICATION DOCUMENTS:                                                   *
*                                                                            *
*      /hisde/req/requirements                                               *
*      /hisde/design/design                                                  *
*                                                                            *
*                                                                            *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                        *
*                                                                            *
*      Mark D. Collier - Software Engineering Section                        *
*                        Data System Science and Technology Department       *
*                        Automation and Data Systems Division                *
*                        Southwest Research Institute                        *
*****************************************************************************/

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

Widget create_toggle ( instance, parent, label )

                                /* This function returns the return value of the
                                 * XtCreateManagedWidget function call.  This will
                                 * be a pointer to a widget.
                                 */

    char    *instance,          /* The instance name of the widget.  It uniquely
                                 * defines the widget.
                                 */
            *label;             /* The string which this label widget will display.
                                 */
    Widget  parent;             /* The parent widget to which the label widget will
                                 * be attached.
                                 */
{
    Arg args[ 2 ];              /* Argument list used to initialize the widget
                                 * resources.
                                 */

    Widget      widget;         /* Pointer to the created widget.
                                 */

    XmString    string;         /* Points to the compound string created for the
                                 * label.
                                 */

    register int    count = 0;  /* Counter set to the number of arguments initialized.
                                 */

/*
 *  Convert the label to a compound string and initialize 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 = XmCreateToggleButton ( parent, instance, args, count ) );
        XmStringFree ( string );

        return ( widget );
    }
```

```
/*****************************************************************************
* MODULE NAME AND FUNCTION ( display_message )
*
* This function displays different types of popups for different message types. It dis-
* plays a modal popup which when acknowledged, is automatically removed. This function
* also calls (h_message). Note that all user interface clients should use this function
* to present hisde messages.
*
*
* SPECIFICATION DOCUMENTS:
*
*      /hisde/req/requirements
*      /hisde/design/design
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*      Mark D. Collier - Software Engineering Section
*                        Data System Science and Technology Department
*                        Automation and Data Systems Division
*                        Southwest Research Institute
*****************************************************************************/


#include <X11/Intrinsic.h>
#include <X11/MwmUtil.h>
#include <Xm/MessageB.h>
#include <hisde.h>


extern  Widget top;


int display_message ( type, message )
    /* Function returns the return value of h_message
 * call.
                                    */

    int     type;                   /* Type of the message. Used to determine the type
                                     * of popup displayed:
                                     *
                                     *  MSG_APPLICATION     1
                                     *  MSG_ERROR           2
                                     *  MSG_HOST            3
                                     *  MSG_INFORMATION     4
                                     *  MSG_WARNING         5
                                     */
    char    *message;               /* Message text to actually display.
                                     */
{
    Arg             args[ 1 ];      /* Argument list used to initialize the widget
                                     * resources.
                                     */

    static Widget   widget;         /* Pointer to the created widget.
                                     */

    XmString        string;         /* Points to the compound string created for the
                                     * label.
                                     */

    register int    count = 0;      /* Counts the number of arguments.
                                     */
```

```
/*
 *  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    MSG_APPLICATION:
    case    MSG_HOST:
    case    MSG_INFORMATION:
        widget = XmCreateInformationDialog ( top, "", args, count );
        break;
    case    MSG_ERROR:
        widget = XmCreateErrorDialog       ( top, "", args, count );
        break;
    case    MSG_WARNING:
        widget = XmCreateWarningDialog     ( top, "", args, count );
        break;
    default:
        break;
    }

/*
 *  Set the modal flag on the popup shell widget.
 */

    count = 0;
    XtSetArg ( args[ count ], XmNmwmInputMode, MWM_INPUT_APPLICATION_MODAL ); count++;
    XtSetValues ( XtParent ( widget ), args, count );

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

    XtManageChild ( widget );
    XmStringFree ( string );

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

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

/*
 *  Call h_message to send the message to the advisory client and return.
 */

    return ( h_message ( type, message ) );
}
```

```
/*******************************************************************************
* MODULE NAME AND FUNCTION ( get_text_widget )
*
* This function is used to retrieve all text within a text widget.
*
*                                                                              *
* SPECIFICATION DOCUMENTS:                                                     *
*                                                                              *
*      /hisde/req/requirements                                                 *
*      /hisde/design/design                                                    *
*                                                                              *
*                                                                              *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                          *
*                                                                              *
*      Mark D. Collier - Software Engineering Section                          *
*                        Data System Science and Technology Department         *
*                        Automation and Data Systems Division                  *
*                        Southwest Research Institute                          *
*******************************************************************************/

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


char *get_text_widget ( widget )

    Widget  widget;                 /* Pointer to the text widget from which the data
                                     * will be retrieved.
                                     */
{
/*
 *  Retrieve all text from the widget. Note that this is not a compound string.
 */

    return ( XmTextGetString ( widget ) );
}
```

```
/*****************************************************************************
* MODULE NAME AND FUNCTION ( get_text_insertion_widget )                     *
*                                                                            *
* This function is used to return the position of the cursor in a text widget.*
*                                                                            *
*                                                                            *
* SPECIFICATION DOCUMENTS:                                                   *
*                                                                            *
*      /hisde/req/requirements                                               *
*      /hisde/design/design                                                  *
*                                                                            *
*                                                                            *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                        *
*                                                                            *
*      Mark D. Collier - Software Engineering Section                        *
*                        Data System Science and Technology Department       *
*                        Automation and Data Systems Division                *
*                        Southwest Research Institute                        *
*****************************************************************************/


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


int get_text_insertion_widget ( widget )

    Widget  widget;                 /* Pointer to the text widget from which the cursor
                                     * position is desired.
                                     */

{
/*
 *  Get and return the text cursor position.
 */

    return ( XmTextGetInsertionPosition ( widget ) );
}
```

```
/*****************************************************************************
* MODULE NAME AND FUNCTION ( get_text_sel_widget )
*
* This function is used to retrieve the currently highlighted text within a text widget.
*
*
* SPECIFICATION DOCUMENTS:
*
*      /hisde/req/requirements
*      /hisde/design/design
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*      Mark D. Collier - Software Engineering Section
*                        Data System Science and Technology Department
*                        Automation and Data Systems Division
*                        Southwest Research Institute
*****************************************************************************/


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


char *get_text_sel_widget ( widget )

    Widget  widget;                 /* Pointer to the text widget from which the high-
                                     * lighted text is desired.
                                     */
{
/*
 *  Get and return a pointer to the current text selection.
 */

    return ( XmTextGetSelection ( widget ) );
}
```

```
/*****************************************************************************
* MODULE NAME AND FUNCTION ( init_list )                                     *
*                                                                            *
* This function initializes the entries in an XmList widget.                *
*                                                                            *
*                                                                            *
* SPECIFICATION DOCUMENTS:                                                   *
*                                                                            *
*      /hisde/req/requirements                                               *
*      /hisde/design/design                                                  *
*                                                                            *
*                                                                            *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                        *
*                                                                            *
*      Mark D. Collier - Software Engineering Section                        *
*                        Data System Science and Technology Department       *
*                        Automation and Data Systems Division                *
*                        Southwest Research Institute                        *
*****************************************************************************/


#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Cardinals.h>
#include <Xm/List.h>
#include <hisde.h>


void init_list ( widget, data_list )

    Widget  widget;                 /* Set to the list widget which will be updated.
                                     */

    char    *data_list;             /* String containing the logical strings (those
                                     * terminated by newlines) to be placed in the
                                     * list.
                                     */

{
    char        temp[ SIZE_HOSTNAME  + 1 ];
                                    /* String used to contain the current entry as parsed
                                     * from the (data_list). This value will be converted
                                     * to an XmString and saved in (list). A hostname is
                                     * the largest entry placed in a list.
                                     */

/*
 *  Scan the list and create XmStrings for placement in the selection box. Note that
 *  (data_list) includes a number of logical strings terminated by newlines. The
 *  physical strings is terminated by a newline. Note that the list is terminated by
 *  a NULL entry.
 */

    while ( *data_list ) {
        sscanf ( data_list, "%s", temp );
        XmListAddItem ( widget, XmStringCreateLtoR ( temp, XmSTRING_DEFAULT_CHARSET ),
                        0 );
        data_list += strlen ( temp ) + 1;
    }
}
```

```
/*****************************************************************************
* MODULE NAME AND FUNCTION ( insert_text_widget )
*
* This function inserts a new string into a text widget at the current cursor position.
*                                                                             *
*                                                                             *
* SPECIFICATION DOCUMENTS:                                                    *
*                                                                             *
*      /hisde/req/requirements                                                *
*      /hisde/design/design                                                   *
*                                                                             *
*                                                                             *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                         *
*                                                                             *
*      Mark D. Collier - Software Engineering Section                         *
*                        Data System Science and Technology Department        *
*                        Automation and Data Systems Division                 *
*                        Southwest Research Institute                         *
*****************************************************************************/


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


void insert_text_widget ( widget, new_text )

    Widget  widget;                 /* Pointer to the text widget which will be updated
                                     * with the new text.
                                     */
    char    *new_text;              /* The new string which is to be initialized in the
                                     * text widget.
                                     */
{
    register int    pos;            /* Set to the position of the cursor in the text
                                     * widget.
                                     */

/*
 *  Get the current position of the text cursor and use to add the new text.
 */


    pos = XmTextGetInsertionPosition ( widget );
    XmTextReplace ( widget, pos, pos, new_text );
}
```

```
/*****************************************************************************
* MODULE NAME AND FUNCTION ( load_text_widget )                              *
*                                                                            *
* This function initializes a text widget from a file.                       *
*                                                                            *
*                                                                            *
* SPECIFICATION DOCUMENTS:                                                   *
*                                                                            *
*      /hisde/req/requirements                                               *
*      /hisde/design/design                                                  *
*                                                                            *
*                                                                            *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                        *
*                                                                            *
*      Mark D. Collier - Software Engineering Section                        *
*                        Data System Science and Technology Department       *
*                        Automation and Data Systems Division                *
*                        Southwest Research Institute                        *
*****************************************************************************/


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


int load_text_widget ( file, widget, ptr )
                                    /* This function reads a file and loads the data into
                                     * a text widget.
                                     *
                                     *      ( 0) - Successful operation
                                     *      (-1) - Error occurred.
                                     */

    char            *file;          /* Name of the file to be initialized.
                                     */

    Widget          widget;         /* Text widget to be initialized with file data.
                                     */

    int             ptr;            /* Pointer into the text widget where the new text
                                     * will go.
                                     */

{
    FILE            *fp;            /* File pointer used to open and access the user's
                                     * history file.
                                     */

    register int    i = 0,          /* Pointer used to maintain position in the (string)
                                     * buffer when initializing command list.
                                     */
                    c;              /* Used to contain last character read (for EOF
                                     * checking).
                                     */

    char        string[ 101 ];      /* Buffer used to read in the command list data
                                     * (100 bytes at a time).
                                     */

/*
 *  Open the file. If this fails, log and error and return.
 */
```

```
    if ( ( fp = fopen ( file, "r" ) ) == NULL )
        return ( -1 );

/*
 *  If starting pointer is -1, clear the text widget first.
 */

    if ( ptr == -1 ) {
        clear_text_widget ( widget );
        ptr = 0;
    }

/*
 *  Read data from the file. Read 100 bytes at a time and add to the text widget's
 *  string.
 */

    while ( ptr != EOF ) {
        while ( i < 100 && ( string[ i ] = c = getc ( fp ) ) != EOF )
            i++;
        string[ i ] = NULL;
        XmTextReplace ( widget, ptr, ptr, string );
        if ( c == EOF )
            ptr = EOF;
        else {
            ptr += i;
            i = 0;
        }
    }

/*
 *  Close the file.  If an error occurs, output an error to the system message
 *  client.
 */

    if ( fclose ( fp ) != 0 )
        return ( -1 );

    return ( 0 );
}
```

```
/*****************************************************************************
* MODULE NAME AND FUNCTION ( update_text_widget )                            *
*                                                                            *
* This function is used to update all text within a text widget.             *
*                                                                            *
*                                                                            *
* SPECIFICATION DOCUMENTS:                                                   *
*                                                                            *
*      /hisde/req/requirements                                               *
*      /hisde/design/design                                                  *
*                                                                            *
*                                                                            *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                        *
*                                                                            *
*      Mark D. Collier - Software Engineering Section                        *
*                        Data System Science and Technology Department       *
*                        Automation and Data Systems Division                *
*                        Southwest Research Institute                        *
*****************************************************************************/


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


void update_text_widget ( widget, new_text )

    Widget  widget;                 /* Pointer to the text widget which will be updated.
                                     */
    char    *new_text;              /* The new string which is to be initialized in the
                                     * text widget.
                                     */

{
/*
 *  Replace the old text with the new text.
 */

    XmTextSetString ( widget, new_text );
}
```

ATTACHMENT 2 - Client Code

```
##############################################################################
# Makefile for HISDE user interface client h_advisory.                       #
##############################################################################

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

TARGET      = h_advisory

#
# Initialize include and library search paths to include current directory and the
# HISDE directories.  Note that the library path also includes the user interface
# library.
#

BINDIR          = /hisde/bin
INCDIR          = /hisde/src/include
INCDIRS         = -I. -I$(INCDIR)

#
# Define the libraries to search.  This includes the HISDE library, the local user
# interface library, and all required X libraries.
#

LIBRARIES       = -lui -lhisde -lXm -lXt -lX11

#
# Define the compiler and linker flags.
#

CFLAGS          = -O $(INCDIRS)
LDFLAGS         = -O $(EXTRAFLAGS)

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

OBJS        =\
            cbr_exit_com.o\
            tmr_stat_upd.o\
            h_adv_bullet.o\
            h_adv_msg.o\
            h_advisory.o

#
# Define all header files required.
#

HDRS        =\
            $(INCDIR)/h_advisory.h\
            $(INCDIR)/h_advisory.bit\
            $(INCDIR)/hisde.h

#
# Make the target.
#

all:        $(TARGET)

$(TARGET):  $(OBJS)
    $(CC) -o $@ $(OBJS) $(LIBRARIES) $(LDFLAGS)
    strip $(TARGET)
    mv $(TARGET) $(BINDIR)
```

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

```
/*****************************************************************************
* MODULE NAME AND FUNCTION: ( h_advisory )                                   *
*                                                                            *
*       The h_advisory client provides the user with the advisory window for the HISDE *
*       system.  It allows the user to view received messages from the system, *
*       host, and other applications.  There are five types of messages which may be *
*       received by this client.  They are:                                  *
*                                                                            *
*               1)  application messages,                                    *
*               2)  error messages,                                          *
*               3)  host messages,                                           *
*               4)  informative messages, and                                *
*               5)  warning messages.                                        *
*                                                                            *
*       This client displays the received messages in a scrolling window and *
*       keeps a counter for each message type indicating the number of messages *
*       which have been displayed.  This counter is displayed above the scroll window. *
*                                                                            *
*       There is also a command button for each message type, which allows the user *
*       to turn a filter on and off for each message type.  If the user selects *
*       a command button, turning the filter on, any messages received of that *
*       type are ignored.  (That is, they are written to the log file, but not *
*       displayed in the scrolling window.)  If the user selects the command button *
*       again, the filter is turned off, thus allowing messages of that type *
*       to be displayed again.  The default for all filters will be 'OFF', but *
*       the user is allowed to run h_advisory with parameters to initialize *
*       particular message type filters as 'ON'.  Whenever the state of a filter *
*       is changed, the command button's background and foreground colors *
*       are reversed to indicate the change.                                 *
*                                                                            *
*       This client uses a timer routine to check the message queue for new *
*       messages.  The default timer value is 2 seconds.  If the user wants *
*       to change the interval, he/she may do so in the command line when *
*       running advisory by using the '-interval' option.                    *
*                                                                            *
*       The log files created by this routine are the host bulletin log file *
*       which contains the host messages received in the message queue and *
*       the message log file which contains all of the messages received in *
*       the message queue.  These files may be viewed by running h_bulletin *
*       for the host bulletin log and h_msg_look for the message log.        *
*                                                                            *
*                                                                            *
* DESCRIPTION OF MAIN FUNCTION:                                              *
*                                                                            *
*       This is the main driver for the h_advisory client of the HISDE system.  It *
*       initializes the X Windows system and then creates the widgets *
*       necesssary for the h_advisory window.  The window created contains *
*       a label for the advisory window, an exit command button, a command button to *
*       turn each message type's filter on or off, labels and text for each type's *
*       unacknowledged message count, and a scrolling window for the display of *
*       messages.                                                            *
*                                                                            *
*       This client will display the window and then enter the XtMainLoop routine *
*       and periodically check for messages.  It will also handle the user selecting *
*       a command button.  If a filter button is selected the associated command *
*       function will be executed to switch the filter's state and reverse the *
*       button's color.  The functions associated with each message type are: *
*                                                                            *
*               Application messages    - appl_command(),                    *
*               Error messages          - err_command(),                     *
*               Host messages           - host_command(),                    *
*               Informative messages    - info_command(), and                *
*                                                                            *
*               Warning messages        - warn_command().                    *
```

```
*
*       If the exit button is selected, the exit_command() function is executed
*       and h_advisory is terminated.
*
*       In order to periodically check the message queue for messages, a timer
*       is started before entering XtMainLoop.  When this timer expires,
*       the update_status() function is executed.  This function will retrieve
*       any messages from the queue, check the message types and display any
*       messages whose filter is not turned on.  Once all messages have been
*       retrieved from the queue, the timer is started again.  This will
*       continue until the user selects the exit button.
*
*
* SPECIFICATION DOCUMENTS:
*
*      /hisde/req/requirements
*      /hisde/design/design
*
*
* EXECUTION SEQUENCE:
*
*       h_advisory [-appl] [-err] [-host] [-info] [-warn] [-interval seconds]
*
*       In addition to the X Windows options which may be used when
*       running h_advisory, the following options are defined:
*
*       -appl    - turns the filter on for application messages.
*       -err     - turns the filter on for error messages.
*       -host    - turns the filter on for host messages.
*       -info    - turns the filter on for the informative messages.
*       -warn    - turns the filte on for warning messages.
*       -interval [seconds]  - indicates the interval, in seconds, desired
*                  by the user.
*
*
* FILES USED AND APPLICATION DEFINED FORMATS:
*
*      /hisde/.msg_log  - This file is used by the h_advisory client to log all
*                         messages received in the message queue.  It is set up
*                         as a circular file with a maximum number of messages.
*                         Because it is a circular file, each message written
*                         to this file must be of the same length.  Therefore,
*                         each message is written to a blank message buffer of
*                         the maximum message size possible.  In order to
*                         maintain this file, the last position written to in the
*                         file each time a message is added is written at the
*                         beginning of the file.  The maximum sizes for this
*                         file are defined in the h_logfiles.h header file.
*
*            struct .msg_log {
*                char[POSITION_OFFSET]                    last_position;
*                char[MAX_NUM_MSG * MAX_MESSAGE]          messages;
*            }
*
*      /hisde/.host_log - This file is used by the h_advisory client to log all
*                         host messages received in the message queue.  It is set
*                         up as a circular file with a maximum number of messages.
*                         Because it is a circular file, each message written
*                         to this file must be of the same length.  Therefore,
*                         each message plus a newline are written to a blank
*                         message buffer of the maximum message size possible.
*                         The newline is necessary for the display of these
*                         message in the h_bulletin client.  In order to
*                         maintain this file, the last position written to in the
```

```
*                       file each time a message is added is written at the
*                       beginning of the file.  The maximum sizes for this
*                       file are defined in the h_logfiles.h header file.
*
*           struct .host_log {
*               char[POSITION_OFFSET]              last_position;
*               char[MAX_NUM_HOST * MAX_MESSAGE]   messages;
*           }
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*      Nancy L. Martin - Software Engineering Section
*                        Data System Science and Technology Department
*                        Automation and Data Systems Division
*                        Southwest Research Institute
*****************************************************************************/


#include <stdio.h>
#include <X11/IntrinsicP.h>
#include <X11/StringDefs.h>
#include <X11/Cardinals.h>
#include <X11/Shell.h>
#include <Xm/MainW.h>
#include <Xm/RowColumn.h>
#include <Xm/Form.h>
#include <fcntl.h>
#include <hisde.h>
#include <h_user_inter.h>
#include <h_advisory.h>
#include <h_advisory.bit>
#include <h_logfiles.h>


/*
 *  Declare all external widgets to be used by the h_advisory application.
 *  This is required for their use in the callback and action routines.
 */

Widget  top, m_main, mb_main, mp_file, aform, widget,
        appl_txt, appl_tog,
        err_txt,  err_tog,
        host_txt, host_tog,
        info_txt, info_tog,
        warn_txt, warn_tog,
        msg_scrll;

/*
 *  Declare the filter flags and counters for each message type.
 */

int     mtype_counters[ NUMBER_MSG_TYPES ];

/*
 *  Declare the current position values in the host bulletin log and the message
 *  log.
 */

long    last_position,
        log_position;

/*
 *  Declare the interval to be used when checking the message queue for
```

```
 *  messages.  It's default is 2 seconds.  This may be changed in the
 *  command line with the -interval parameter.
 */

unsigned    long    timer_interval = DEFAULT_INTERVAL;

/*
 *  Declare the callback procedures to be executd when a command button is selected.
 */

extern  XtCallbackProc  exit_command();

/*
 *  Declare the callback procedure to be executed when the timer value expires.
 */

extern  XtTimerCallbackProc     update_status();


main ( argc, argv )
    int     argc;
    char    **argv;
{
/*
 *  Flags indicating whether the user requested any message type filters to
 *  be initially set to 'off'.  This can be indicated in the command line by
 *  the -appl, -err, -host, -info, and -warn options.
 */

    static  Boolean fil1,           /*  Application message.            */
                    fil2,           /*  Error message.                  */
                    fil3,           /*  Host message.                   */
                    fil4,           /*  Informative message.            */
                    fil5;           /*  Warning message.                */

/*
 *  Declare the application-specific resources allowed by this client.  The
 *  resources which may be set are the message type filters and the interval desired
 *  for checking the message queue.
 *
 */

    static XrmOptionDescRec options[ ] = {
        {"-appl",       "AAppl",    XrmoptionNoArg,     "True"},
        {"-err",        "AErr",     XrmoptionNoArg,     "True"},
        {"-host",       "AHost",    XrmoptionNoArg,     "True"},
        {"-info",       "AInfo",    XrmoptionNoArg,     "True"},
        {"-warn",       "AWarn",    XrmoptionNoArg,     "True"},
        {"-interval",   "Interval", XrmoptionSepArg,    NULL  }
    };

    static  XtResource  resources[ ] = {
   { "aappl",       "AAppl",    XtRBoolean, sizeof(Boolean),    (Cardinal)&fil1,
                    XtRString,  "False" },
   { "aerr",        "AErr",     XtRBoolean, sizeof(Boolean),    (Cardinal)&fil2,
                    XtRString,  "False" },
   { "ahost",       "AHost",    XtRBoolean, sizeof(Boolean),    (Cardinal)&fil3,
                    XtRString,  "False" },
   { "ainfo",       "AInfo",    XtRBoolean, sizeof(Boolean),    (Cardinal)&fil4,
                    XtRString,  "False" },
   { "awarn",       "AWarn",    XtRBoolean, sizeof(Boolean),    (Cardinal)&fil5,
                    XtRString,  "False" },
   { "interval",    "Interval", XtRInt,     sizeof(int),        (Cardinal)&timer_interval,
                    XtRInt,     (caddr_t)&timer_interval }
```

```
    };

/*
 *  Declare the callback list array to be used when creating command widgets.
 *  This array will contain the routines to be executed when the associated
 *  command button is selected.
 */

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

    Arg             icon_arg,       /* Argument used to initialize the icon.
                                     */
                    args[ 1 ];      /* Argument list used to initialize various
                                     * widget resources.
                                     */

    XtIntervalId    id;             /* The ID necessary for identifying the timer.
                                     */

    int             fd,             /* The file descriptor of the opened log files.
                                     */
                    i;              /* Used to step through the array of message
                                     * counters.
                                     */
    char            position[POSITION_OFFSET];
                                    /* Character string used to read the last
                                     * the last position written to value from the
                                     * log files.
                                     */


/*
 *  Initialize the message counters to zero.
 */

    for ( i = 0; i < NUMBER_MSG_TYPES; i++ )
        mtype_counters[ i ] = 0;

/*
 *  Initialize the file positions for the message and host bulletin log files.
 *  Open the files, read the position value, convert the value to an integer,
 *  and assign it to the appropriate external variable for each file.
 *  If there is not a log file, assign the position value to be zero.
 */

    if (( fd  = open ( HISDE_HOST_LOG, O_RDONLY )) <= NULL ) {
        last_position = ZERO;
    } else {
        if ( read ( fd, position, POSITION_OFFSET ) != POSITION_OFFSET ) {
            fprintf ( stderr, "h_advisory:  Cannot read host bulletin file position." );
            close (fd);
            exit (-1);
        } else {
            last_position = atoi ( position );
            close (fd);
        }
    }
    if (( fd  = open ( HISDE_MSG_LOG, O_RDONLY )) <= NULL ) {
        log_position = ZERO;
    } else {
        if ( read ( fd, position, POSITION_OFFSET ) != POSITION_OFFSET ) {
            fprintf ( stderr, "h_advisory:  Cannot read message log file position." );
```

```
            close (fd);
            exit (-1);
        } else {
            log_position = atoi ( position );
            close (fd);
        }
    }

/*
 *  Initialize the X Windows system and create the top level widget for the advisory
 *  screen.
 */

    top = XtInitialize ( ADVISORY_SHELL, ADVISORY_CLASS, options, XtNumber(options),
                         &argc, argv );

/*
 *  If there were invalid arguments on the command line which could not be parsed,
 *  call the function, bad syntax, to display the correct syntax and exit from
 *  the client.
 */

    if ( argc > 1 )
        bad_syntax (
        "h_advisory [-interval time] [-appl] [-err] [-host] [-info] [-warn]" );

/*
 *  Initialize the icon bitmap for this client.
 */

    XtSetArg ( icon_arg, XtNiconPixmap,
               XCreateBitmapFromData ( XtDisplay(top), XtScreen(top) -> root,
               h_advisory_bits, h_advisory_width, h_advisory_height ) );

    XtSetValues ( top, &icon_arg, ONE );

/*
 *  Retrieve any application-specific resources which were initialized previously or
 *  in the command line.  This includes both initialization of message type filters
 *  and the message queue read interval.  Multiply the specified interval by 1000 to
 *  get it into milliseconds.
 */

    XtGetApplicationResources (top, (caddr_t)NULL, resources, XtNumber(resources),
                               NULL, ZERO );
    timer_interval = timer_interval *  1000;

/*
 *  Create the main window widget and the menu bar which will contain all commands.
 */

    m_main = XmCreateMainWindow ( top, "", NULL, 0 );
    XtManageChild ( m_main );

    mb_main = XmCreateMenuBar ( m_main, "", NULL, 0 );
    XtManageChild ( mb_main );

/*
 *  Create pulldown for file commands.
 */

    command_callbacks[ 0 ].callback = (XtCallbackProc)exit_command;
    mp_file = XmCreatePulldownMenu ( mb_main, "", NULL, 0 );
    create_cascade ( "", mb_main, mp_file, LABEL_FILE                    );
```

```
    create_command ( "", mp_file,          LABEL_EXIT, command_callbacks );

/*
 *  Create the help cascade.
 */

    widget = create_cascade ( "", mb_main, NULL, LABEL_HELP );
    XtSetArg ( args[ 0 ], XmNmenuHelpWidget, widget );
    XtSetValues ( mb_main, args, 1 );

/*
 *  Create the HISDE Advisory  window which will contain command buttons to filter
 *  each message type, text widgets to display the unacknowledged message counts
 *  for each type, and a scrolling window for the display of received messages.
 */

    aform = create_form ( "", m_main );

/*
 *  The text widgets will contain strings which have a message type label and the
 *  message counter.  The counter is initialized to zero.
 *  An assignment to the command callback list is made to indicate which callback
 *  function is to be called when the created button is selected.
 */

    appl_tog = create_toggle ( W_TG_APPL, aform, LABEL_APPL );
    appl_txt = create_text   ( W_T_APPL,  aform, "0", 0, XmSINGLE_LINE_EDIT, 0 );

    err_tog  = create_toggle ( W_TG_ERR,  aform, LABEL_ERR  );
    err_txt  = create_text   ( W_T_ERR,   aform, "0", 0, XmSINGLE_LINE_EDIT, 0 );

    host_tog = create_toggle ( W_TG_HOST, aform, LABEL_HOST );
    host_txt = create_text   ( W_T_HOST,  aform, "0", 0, XmSINGLE_LINE_EDIT, 0 );

    info_tog = create_toggle ( W_TG_INFO, aform, LABEL_INFO );
    info_txt = create_text   ( W_T_INFO,  aform, "0", 0, XmSINGLE_LINE_EDIT, 0 );

    warn_tog = create_toggle ( W_TG_WARN, aform, LABEL_WARN );
    warn_txt = create_text   ( W_T_WARN,  aform, "0", 0, XmSINGLE_LINE_EDIT, 0 );

/*
 *  Create the text widget to be used as the message window.  It is created
 *  with a vertical scrollbar to allow the user to page through displayed
 *  messages.
 */

    msg_scrll = create_text ( MSG_TEXT_SW, aform, "", 1, XmMULTI_LINE_EDIT, 0 );

/*
 *  Initialize the first iteration of the timer.  This will cause the update_status
 *  callback routine to be executed.  This routine will reset the timer each time
 *  it completes its function.
 */

     id = XtAddTimeOut ( timer_interval, update_status, NULL );

/*
 *  Call XtRealizeWidget on the top level widget to display the h_advisory window.
 *  Next, enter the XtMainLoop routine to process events, timers, and actions.
 *  This client will be terminated in a callback routine when the user has
 *  requested to exit the advisory window.
 */

     XtRealizeWidget ( top );
```

```
/*
 *  If a message type filter has been selected to be set, turn the filter
 *  on and reverse the command button's colors.
 */
        if ( fil1 )
            XmToggleButtonSetState ( appl_tog, TRUE, FALSE );
        if ( fil2 )
            XmToggleButtonSetState ( err_tog,  TRUE, FALSE );
        if ( fil3 )
            XmToggleButtonSetState ( host_tog, TRUE, FALSE );
        if ( fil4 )
            XmToggleButtonSetState ( info_tog, TRUE, FALSE );
        if ( fil5 )
            XmToggleButtonSetState ( warn_tog, TRUE, FALSE );

/*
 *  Enter the Xtoolkit main loop to coordinate processing of all widget events.
 *  This loop is terminated when the user selects the exit command button and
 *  the associated callback procedure is executed to terminate this client.
 */

        XtMainLoop ( );
}
```

```
/*****************************************************************************
* MODULE NAME AND FUNCTION: ( update_host_bulletin )
*
*       This function will open the host bulletin log file and add the most
*       recent host message into the next available position in the file.
*       If the file has exceeded its maximum size, the new message will be
*       written over the oldest message in the file.  In order to maintain
*       this circular file, the last position written to in the file is
*       stored in the first twenty bytes of the file.   This position value
*       indicates where the next message should be written on the next pass
*       through this function.
*
*
* SPECIFICATION DOCUMENTS:
*
*      /hisde/req/requirements
*      /hisde/design/design
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*      Nancy L. Martin - Software Engineering Section
*                        Data System Science and Technology Department
*                        Automation and Data Systems Division
*                        Southwest Research Institute
*****************************************************************************/


#include <stdio.h>
#include <fcntl.h>
#include <hisde.h>
#include <h_advisory.h>
#include <h_logfiles.h>


update_host_bulletin ( new_record, last_position )
    char    *new_record;        /*  Specifies the new host message to be    */
                                /*  added to the file.                      */
    long    last_position;      /*  Specifies the last position written to  */
                                /*  in the file.                            */
{
/*
 *  Declare the buffers used to write the last position value and the messages
 *  to the host bulletin log file.
 */
    char    buffer[MAX_MESSAGE],
            position[POSITION_OFFSET];
/*
 *  Declare the values needed to open the host bulletin log file.
 */
    int     open(),     fd;
/*
 *  Declare the value to be used to step through the output buffers.
 */
    register    int i;
/*
 *  Initialize the message output buffer to blanks.
 */
    for ( i = 0; i < MAX_MESSAGE;  i++ )
        buffer[i] = BLANK;
/*
 *  Open the host bulletin log file.  If the file is not already created, create it
 *  with the appropriate protections.
 *
```

```
    *  If the open is not successful, display a message and exit h_advisory.
    */
       if (( fd = open ( HISDE_HOST_LOG, O_WRONLY | O_CREAT, 0666 )) <= NULL ) {
           fprintf ( stderr, "h_advisory:  Cannot open host bulletin file" );
           exit (-1);
       }
   /*
    *  Build a constant size buffer for writing the passed message to the host bulletin
    *  log file.
    */
       sprintf ( buffer, "%s", new_record );
   /*
    *  Determine where the message should be written in the log file.  If the
    *  last position written to in this file was at the end of the file or
    *  if it is the first message being written to the file, set the file
    *  position to be just passt the bytes allotted for the file position value.
    *  Then set the last-position-written-to value to be the size of the previous
    *  position value plus the size of the new message.
    */
       if (( last_position >= MAX_HOST_LOG ) || ( last_position == 0 )) {
           lseek ( fd, POSITION_OFFSET, 0 );
           last_position = MAX_MESSAGE + POSITION_OFFSET;
   /*
    *  Otherwise, set the file position to the last file position and increase the
    *  last position written to by the size of the new message.
    */
       } else {
           lseek ( fd, last_position, 0 );
           last_position += MAX_MESSAGE;
       }
   /*
    *  Write the new message to the current file position.  If an errror occurs,
    *  display a message and exit the h_advisory client.
    */
       if ( write ( fd, buffer, MAX_MESSAGE ) != MAX_MESSAGE ) {
           fprintf ( stderr,
           "h_advisory:  Cannot write to host bulletin file." );
           close ( fd );
           exit (-1);
       }
   /*
    *  Set the position output buffer to blanks.
    */
       for ( i = 0; i < POSITION_OFFSET;  i++ )
           position[i] = BLANK;
   /*
    *  Assign the new position value to the output buffer and write it
    *  to the beginning of the file.
    *
    *  If an error occurs, display a message, close the file, and exit the
    *  h_advisory client.
    */
       sprintf ( position, "%d", last_position );
       lseek ( fd, 0L, 0 );
       if ( write ( fd, position, POSITION_OFFSET ) != POSITION_OFFSET ) {
           fprintf ( stderr,
           "h_advisory:  Cannot write to host bulletin file." );
           close ( fd );
           exit (-1);
       }
   /*
    *  If the new message was successfully written to the host bulletin log
    *  file, close the file and return the last position written to value
    *  to the calling routine for use on future calls to update this file.
```

```
    */
        close ( fd );
        return ( last_position );
    }
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: update_status()
 *
 *       This function is a timer callback procedure which is executed when the
 *       timer interval expires.  This function executes a loop until there
 *       are no longer any messages waiting in the message queue.
 *
 *       Within this loop, a message and message type are received.  As each
 *       message is received it is written to the message log file (and the
 *       host bulletin log file, if it is a host message).  If the filter
 *       for the received message type is set, processing is stopped on this
 *       message and the next message is retrieved.  Otherwise, a message counter
 *       is incremented to indicate the number of unacknowledged messages which
 *       have been received for this message type.  Next, the text containing
 *       the message counter is updated by calling update_text_widget
 *       for the appropriate message type's text widget.  When the change has
 *       been completed, the new counter and label are copied over the old
 *       text and a flag is set to indicate that there is a new message to
 *       be displayed.
 *
 *       If a new message is to be added to the list of displayed messages,
 *       a new list of messages is allocated and created by adding the new
 *       message to the end of the old list, old_message.  If this is accomplished
 *       without error, the new list, new_message, is copied over the old_list,
 *       old_message.  This process is done until there are no more
 *       messages waiting in the message queue.
 *
 *       When all messages have been retrieved from the queue and processed,
 *       the change flag is checked.  If this flag is set then the size of
 *       the new list is checked.  If this size exceeds the current size
 *       of the text widget, list_size, then the widget is destroyed and
 *       recreated with a text size that has been incremented to accomodate
 *       the new messages.  Otherwise, the new messages are inserted at the
 *       end of the text currently displayed in the scroll window.
 *       Once the widget has been recreated or updated, the
 *       input cursor is set at the beginning of the most recently added
 *       message by calling XtTextSetInsertionPoint using the size of the
 *       previously displayed list of messages as the marker.
 *
 *       Finally, update_status reinitializes the timer value.  This will cause
 *       update_status to be called continually, at the specified interval, to
 *       check the message queue for messages.
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *      /hisde/req/requirements
 *      /hisde/design/design
 *
 *
 * EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *      Nancy L. Martin - Software Engineering Section
 *                        Data System Science and Technology Department
 *                        Automation and Data Systems Division
 *                        Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
```

```
#include <X11/StringDefs.h>
#include <X11/Cardinals.h>
#include <Xm/Text.h>
#include <Xm/ToggleB.h>
#include <hisde.h>
#include <h_user_inter.h>
#include <h_advisory.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>


/*
 *  Declare the current position values in the host bulletin log and the message
 *  log.
 */

extern  long    last_position,
                log_position;

/*
 *  Declare the timer interval value for use in starting the timer back up.
 */

extern  long    timer_interval;

/*
 *  Declare the message counter array and filters so they may be checked and
 *  updated.
 */

extern  int     mtype_counters[ ];

/*
 *  Declare the widgets which are accessed when the message counters and
 *  messages are updated.
 */

extern  Widget  appl_txt, appl_tog,
                err_txt,  err_tog,
                host_txt, host_tog,
                info_txt, info_tog,
                warn_txt, warn_tog,
                msg_scrll;

extern  char    *malloc();


XtTimerCallbackProc update_status ( client_data, id )

    caddr_t         client_data;    /* Specifies the client date that was registered
                                     * registered for this procedure in XtAddTimeOut.
                                     */

    XtIntervalId    *id;            /* Specifies the ID returned from the corresponding
                                     * corresponding XtAddTimeOut call.
                                     */

{
    char    *temp_message,          /* Will point to complete formatted message.
                                     */
            *message_type,          /* Will point to text message type.
                                     */
            *old_message;           /* Will point to existing message text. It is
                                     * needed to compute length.
```

```
                                        */

    int     message_size = 0,           /* Set to the current length of the message list.
                                         */
            updates = FALSE,            /* Flag which indicates if a message was added to
                                         * the list.
                                         */
            size;                       /* Specifies the size of the message to displayed.
                                         */

    struct advisory_struct message;
    key_t msg_queue;


/*
 *  Get the contents and length of the current message list.
 */

    old_message  = get_text_widget ( msg_scrll );
    message_size = strlen ( old_message );
    XtFree ( old_message );

/*
 *  Enter a loop to retrieve all messages that have been received since the last
 *  interval check.
 */

    msg_queue = H_ADV_MESSAGE_KEY + (key_t)h_get_tty();

    while ( msgrcv ( msgget( msg_queue, 0 ), &message, MAX_MESSAGE_LENGTH,
                     0L, IPC_NOWAIT ) != -1 )
    {
        updates = FALSE;

/*
 *  If a message is recieved, udpate the appropriate message count.
 */

        if      ( message.adv_mtype == MSG_APPLICATION ) {
            message_type = APPL_TYPE;
            updates = update_label ( message.adv_mtype, appl_txt, appl_tog );
        } else if ( message.adv_mtype == MSG_ERROR ) {
            message_type = ERR_TYPE;
            updates = update_label ( message.adv_mtype, err_txt, err_tog );
        } else if ( message.adv_mtype == MSG_HOST ) {
            message_type = HOST_TYPE;
            updates = update_label ( message.adv_mtype, host_txt, host_tog );
            if ( ( last_position = update_host_bulletin ( message.adv_mtext,
                                                          last_position ) ) < 0 ) {
                fprintf ( stderr, "h_advisory:  Cannot log host bulletin" );
                exit (-1);
            }
        } else if ( message.adv_mtype == MSG_INFORMATION ) {
            message_type = INFO_TYPE;
            updates = update_label ( message.adv_mtype, info_txt, info_tog );
        } else if ( message.adv_mtype == MSG_WARNING ) {
            message_type = WARN_TYPE;
            updates = update_label ( message.adv_mtype, warn_txt, warn_tog );
        } else
            fprintf ( stderr, "h_advisory: Invalid message type received: \n%s\n",
                      message.adv_mtext );

/*
 *  Allocate storage for the received message, its type, and its type name.
```

```
 *  Next, create the full message in the temp_message buffer.
 */

        size = strlen ( message.adv_mtext ) + MESSAGE_LEADIN_SIZE;
        if ( ( temp_message = malloc ( (unsigned)size) ) == NULL ) {
            fprintf ( stderr, "Cannot allocate space to build current message" );
            exit (-1);
        }
        sprintf ( temp_message, "%s (%d) %s%s\n", MESSAGE_LEADIN,
                    message.adv_mtype, message_type, message.adv_mtext );

/*
 *  Call  update_message_log to add the full message to the message log file.
 */

        if ( ( log_position = update_message_log ( temp_message, log_position ) ) < 0 ) {
            fprintf ( stderr, "h_advisory:  Cannot update log file with new message" );
            exit (-1);
        }

/*
 *  Append the message to the existing list.
 */

        if ( updates ) {
            XmTextSetInsertionPosition ( msg_scrll, message_size );
            insert_text_widget ( msg_scrll, temp_message );
            message_size += strlen ( temp_message );
        }
    }

/*
 *  When the message queue has been emptied and all updates have been made, reset
 *  the timer so that this routine will be called continually until the user
 *  selects to exit the h_advisory client.
 */

    *id = XtAddTimeOut ( timer_interval, update_status, NULL );
}


/*****************************************************************************
* MODULE NAME AND FUNCTION:  update_label                                    *
*                                                                            *
*       This function will, if the appropriate filter is off, update the appropriate     *
*       message count. In this instance, it will return TRUE.                *
*                                                                            *
*                                                                            *
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)      *
*                                                                            *
*       mtype_counters[NUMBER_MSG_TYPES] (int) (I/O) -                       *
*               An array of integers containing the unacknowledged           *
*               message counts for each message type.  Each type's           *
*               counter is accessed by its message type number.              *
*****************************************************************************/


update_label ( index, text_widget, toggle_widget )

    int     index;                  /* Specifies the message type index.
                                       */

    Widget  text_widget,            /* Text widget which will be updated.
                                       */
```

```
            toggle_widget;          /* Toggle widget which determines the state of the
                                     * filter.
                                     */
{
    char    temp[ 10 ];             /* Buffer used to format the new message count.
                                     */

/*
 *  If the filter is disabled, increment the counter and update the text widget.
 */

    if ( XmToggleButtonGetState ( toggle_widget ) == FALSE ) {
        ++mtype_counters[ index - 1 ];
        sprintf ( temp, "%d", mtype_counters[ index - 1 ] );
        update_text_widget ( text_widget, temp );
        return ( TRUE );

/*
 *  Otherwise, return FALSE to indicate that the filter is enabled.
 */

    } else
        return ( FALSE );
}
```

```
/*****************************************************************************************
* MODULE NAME AND FUNCTION: ( update_message_log )
*
*       This function will open the message log file and add the most recent
*       message into the next available position in the file.  If the file has
*       exceeded its maximum size, the new message will be written over the
*       oldest message in the file.  In order to maintain this circular file,
*       the last position written to in the file is stored in the first twenty
*       bytes of the file.  This position value indicates where the next message
*       should be written on the next pass through this function.
*
*
* SPECIFICATION DOCUMENTS:
*
*       /hisde/req/requirements
*       /hisde/design/design
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*       Nancy L. Martin - Software Engineering Section
*                         Data System Science and Technology Department
*                         Automation and Data Systems Division
*                         Southwest Research Institute
*****************************************************************************************/


#include <stdio.h>
#include <fcntl.h>
#include <hisde.h>
#include <h_advisory.h>
#include <h_logfiles.h>


update_message_log ( new_record, log_position )
    char    *new_record;            /*  Specifies the new message to be     */
                                    /*  added to the file.                  */
    long    log_position;           /*  Specifies  the last position        */
                                    /*  written to in the file.             */

{
/*  Define the buffers used to write the last position value and the messages
 *  to the message log file.
 */
    char    buffer[MAX_MESSAGE],
            position[POSITION_OFFSET];
/*
 *  Define the values needed to open the message log file.
 */
    int     open(),     fd;
/*
 *  Declare the value to be used to step through the output buffers.
 */
    register    int i;
/*
 *  Initialize the message output buffer to blanks.
 */
    for ( i = 0; i < MAX_MESSAGE;  i++ )
        buffer[i] = BLANK;
/*
 *  Open the message log file.  If the file is not already created, create it
 *  with the appropriate protections.
 *
 *  If the open is not successful, display a message and exit h_advisory.
 */
```

```
        if (( fd = open ( HISDE_MSG_LOG, O_WRONLY | O_CREAT, 0666 )) <= NULL ) {
            fprintf ( stderr, "h_advisory:  Cannot open message log file" );
            exit (-1);
        }
    /*
     *  Build a constant size buffer for writing the passed message to the message
     *  log file.
     */
        sprintf ( buffer, "%s", new_record );
    /*
     *  Determine where the message should be written in the log file.  If the
     *  last position written to in this file was at the end of the file or
     *  if it is the first message being written to the file, set the file
     *  position to be just past the bytes allotted for the file position value.
     *  Then set the last-position-written-to value to be the size of the previous
     *  position value plus the size of the new message.
     */
        if (( log_position >= MAX_MSG_LOG ) || ( log_position == 0 )) {
            lseek ( fd, POSITION_OFFSET, 0 );
            log_position = MAX_MESSAGE + POSITION_OFFSET;
    /*
     *  Otherwise, set the file position to the last file position and increase the
     *  last position written to by the size of the new message.
     */
        } else {
            lseek ( fd, log_position, 0 );
            log_position += MAX_MESSAGE;
        }
    /*
     *  Write the new message to the current file position.  If an error occurs,
     *  display a message and exit the h_advisory client.
     */
        if ( write ( fd, buffer, MAX_MESSAGE ) != MAX_MESSAGE ) {
            fprintf ( stderr,
            "h_advisory:  Cannot write to message log file." );
            close ( fd );
            exit (-1);
        }
    /*
     *  Set the position output buffer to blanks.
     */
        for ( i = 0; i < POSITION_OFFSET;  i++ )
            position[i] = BLANK;
    /*
     *  Assign the new position value to the output buffer and write it
     *  to the beginning of the file.
     *
     *  If an error occurs, display a message, close the file, and exit the
     *  h_advisory client.
     */
        sprintf ( position, "%d", log_position );
        lseek ( fd, 0L, 0 );
        if ( write ( fd, position, POSITION_OFFSET ) != POSITION_OFFSET ) {
            fprintf ( stderr,
            "h_advisory:  Cannot write to message log file." );
            close ( fd );
            exit (-1);
        }
    /*
     *  If the new message was successfully written to the message log
     *  file, close the file and return the last-position-written-to value
     *  to the calling routine for use on future calls to update this file.
     */
        close ( fd );
```

```
        return ( log_position );
    }
```

```
/*****************************************************************************************
 * MODULE NAME AND FUNCTION: exit_command()                                              *
 *                                                                                       *
 *       The exit_command function is a callback procedure attached to the exit          *
 *       command button of the h_advisory client.  This function causes the client       *
 *       to terminate naturally when the user selects the exit button.                   *
 *                                                                                       *
 *                                                                                       *
 * SPECIFICATION DOCUMENTS:                                                              *
 *                                                                                       *
 *                                                                                       *
 *      /hisde/req/requirements                                                          *
 *      /hisde/design/design                                                             *
 *                                                                                       *
 *                                                                                       *
 * EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)                 *
 *                                                                                       *
 *       top (Widget) (I) - The top level form widget for the h_advisory client.         *
 *                                                                                       *
 *                                                                                       *
 * ORIGINAL AUTHOR AND IDENTIFICATION:                                                   *
 *                                                                                       *
 *      Nancy L. Martin - Software Engineering Section                                   *
 *                        Data System Science and Technology Department                  *
 *                        Automation and Data Systems Division                           *
 *                        Southwest Research Institute                                   *
 *****************************************************************************************/


#include <X11/Intrinsic.h>


/*
 *  Declare the top level widget.
 */

extern  Widget  top;


XtCallbackProc exit_command ( 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.
                                     */
            calldata;               /* Specifies any callback-specific data the widget
                                     * needs to pass to the client.  This parameter is
                                     * is not used by this function.
                                     */
{
/*
 *  Remove the top level widget and then close the h_advisory display.
 */

    XtUnmapWidget ( top );
    XCloseDisplay ( XtDisplay ( top ) );

/*
 *  Exit the h_advisory client with a zero.
 */
```

```
        exit ( 0 );
    }
```

```
##############################################################################
# Makefile for HISDE user interface client h_bulletin.
##############################################################################

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

TARGET      = h_bulletin

#
# Initialize include and library search paths to include current directory and the
# HISDE directories.  Note that the library path also includes the user interface
# library.
#

BINDIR          = /hisde/bin
INCDIR          = /hisde/src/include
INCDIRS         = -I. -I$(INCDIR)

#
# Define the libraries to search.  This includes the HISDE library, the local user
# interface library, and all required X libraries.
#

LIBRARIES       = -lui -lhisde -lXm -lXt -lX11

#
# Define the compiler and linker flags.
#

CFLAGS          = -O $(INCDIRS)
LDFLAGS         = -O $(EXTRAFLAGS)

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

OBJS        =\
            tmr_bul_upd.o\
            cbr_exit_com.o\
            update_win.o\
            h_bulletin.o

#
# Define all header files required.
#

HDRS        =\
            $(INCDIR)/h_bulletin.h\
            $(INCDIR)/h_bulletin.bit\
            $(INCDIR)/hisde.h

#
# Make the target.
#

all:        $(TARGET)

$(TARGET):  $(OBJS)
    $(CC) -o $@ $(OBJS) $(LIBRARIES) $(LDFLAGS)
    strip $(TARGET)
    mv $(TARGET) $(BINDIR)
```

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

```
/*****************************************************************************
* MODULE NAME AND FUNCTION: ( h_bulletin )
*
*       The h_bulletin client provides the user with the host bulletin window for the
*       HISDE system.  It allows the user to view the host messages which have been
*       received.
*
*       This client displays the host bulletin log file in a scroll window which
*       allows the user to view the last twenty host messages which were logged.
*
*       This client uses a timer routine to check if new host messages have
*       been logged.  The default timer value is 2 seconds.  If the user wants
*       to change the interval, he/she may do so in the command line when
*       running bulletin  by using the '-interval' option.  Whenever the timer
*       expires, the last position written to is read from the host_log file
*       and compared to the previous value read from the file.  If the value
*       has changed it is an indication that new messages have been written to the
*       file.
*
*
* DESCRIPTION OF MAIN FUNCTION:
*
*       This is the main driver for the h_bulletin client of the HISDE system.  It
*       initializes the X Windows system and then creates the widgets necessary for the
*       h_bulletin window.   The window created contains a label for the bulletin
*       window, an exit command button, and a scroll window for the display of the
*       logged host bulletins.
*
*       This client will display the window and then enter the XtMainLoop routine
*       and periodically update the display.  It will also handle the user selecting
*       a command button.
*
*       If the exit button is selected, the exit_command() function is executed
*       and h_bulletin is terminated.
*
*       In order to periodically update the host bulletin display, a timer is started
*       before entering XtMainLoop.  When this timer expires, the update_bulletin()
*       function is executed.  This function will access the host bulletin log file and
*       check if the position last written to has changed.  If this is the case then new
*       messages have been received and the scroll window needs to be updated.  Once the
*       scroll window has been updated, the timer is started again.  This will continue
*       until the user selects the exit button.
*
*
* SPECIFICATION DOCUMENTS:
*
*      /hisde/req/requirements
*      /hisde/design/design
*
*
* EXECUTION SEQUENCE:
*
*       h_bulletin [-interval seconds]
*
*       In addition to the X Windows options which may be used when running h_bulletin,
*       the following options are defined:
*
*       -interval [seconds]  - indicates the interval, in seconds, desired by the user.
*
*
* FILES USED AND APPLICATION DEFINED FORMATS:
*
*       /hisde/.host_log - This file is used by the h_bulletin client to retrieve
*                          all host messages received in the message queue.  It
```

```
*                       is set up as a circular file with a maximum number of
*                       messages.  Because it is a circular file, each message
*                       written to this file must be of the same length.
*                       Therefore, each message is read into a blank message
*                       buffer of the maximum message size possible.  The
*                       In order to maintain this file, the last position
*                       written to in the file the last time a message was
*                       added is stored at the beginning of the file.  The
*                       maximum sizes for this file are defined in the
*                       h_logfiles.h header file.
*
*           struct .host_log {
*               char[POSITION_OFFSET]               last_position;
*               char[MAX_NUM_HOST * MAX_MESSAGE]    messages;
*           }
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*      Nancy L. Martin - Software Engineering Section
*                        Data System Science and Technology Department
*                        Automation and Data Systems Division
*                        Southwest Research Institute
*****************************************************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Cardinals.h>
#include <X11/Shell.h>
#include <Xm/MainW.h>
#include <Xm/RowColumn.h>
#include <Xm/Form.h>
#include <hisde.h>
#include <h_user_inter.h>
#include <h_bulletin.h>
#include <h_bulletin.bit>
#include <h_logfiles.h>


/*
 *  Declare all external widgets to be used by the h_bulletin application.
 *  This is required for their use in the callback and action routines.
 */

Widget  top, m_main, mb_main, mp_file, bform, widget,
        msg_scrll;

/*
 *  Declare the interval to be used for redisplaying the host bulletin log.
 *  It's default is 2 seconds.  This may be changed in the command line
 *  with the -interval parameter.
 */

unsigned    long    timer_interval = DEFAULT_INTERVAL;

/*
 *  Declare the callback procedures to be executed when a command button is selected.
 */

extern  XtCallbackProc          exit_command();

/*
```

```
 *  Declare the callback procedure to be executed when the timer value expires.
 */

extern  XtTimerCallbackProc     update_bulletin();


main ( argc, argv )
    int     argc;
    char    **argv;
{
/*
 *  Declare the application-specific resources allowed by this client.  The
 *  resource which may be set is the interval desired for updating the scroll
 *  window.
 */

    static XrmOptionDescRec options[] = {
        {"-interval",   "Interval", XrmoptionSepArg,    NULL  }
    };

    static  XtResource  resources[] = {
   { "interval",    "Interval", XtRInt,     sizeof(int),        (Cardinal)&timer_interval,
                    XtRInt,     (caddr_t)&timer_interval }
   };

/*
 *  Declare the callback list array to be used when creating command widgets.
 *  This array will contain the routines to be executed when the associated
 *  command button is selected.
 */

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

    Arg             icon_arg,       /* Argument used to initialize the icon.
                                     */
                    args[ 1 ];      /* Argument list used to initialize various
                                     * widget resources.
                                     */

    XtIntervalId    id;             /* The ID necessary for identifying the timer.
                                     */

/*
 *  Initialize the X Windows system and create the top level widget for the
 *  host bulletin screen.
 */

    top = XtInitialize ( BULLETIN_SHELL, BULLETIN_CLASS, options, XtNumber(options),
                         &argc, argv );

/*
 *  If there were invalid arguments on the command line which could not be parsed,
 *  call the function, bad syntax, to display the correct syntax and exit from
 *  the client.
 */

    if ( argc > 1 )
        bad_syntax ( "h_bulletin [-interval time]" );

/*
 *  Initialize the icon bitmap for this client.
```

```
 */

    XtSetArg ( icon_arg, XtNiconPixmap,
               XCreateBitmapFromData ( XtDisplay(top), XtScreen(top) -> root,
               h_bulletin_bits, h_bulletin_width, h_bulletin_height ) );

    XtSetValues ( top, &icon_arg, ONE );

/*
 *  Retrieve any application-specific resources which were initialized previously or
 *  in the command line.  This includes the scroll window update interval.
 *  Multiply the specified interval by 1000 to convert in into milliseconds.
 */

    XtGetApplicationResources (top, (caddr_t)NULL, resources, XtNumber(resources),
                               NULL, ZERO );
    timer_interval = timer_interval *  1000;

/*
 *  Create the main window widget and the menu bar which will contain all commands.
 */

    m_main = XmCreateMainWindow ( top, "", NULL, 0 );
    XtManageChild ( m_main );

    mb_main = XmCreateMenuBar ( m_main, "", NULL, 0 );
    XtManageChild ( mb_main );

/*
 *  Create pulldown for file commands.
 */

    command_callbacks[ 0 ].callback = (XtCallbackProc)exit_command;
    mp_file = XmCreatePulldownMenu ( mb_main, "", NULL, 0 );
    create_cascade ( "", mb_main, mp_file, LABEL_FILE                      );
    create_command ( "", mp_file,          LABEL_EXIT, command_callbacks );

/*
 *  Create the help cascade.
 */

    widget = create_cascade ( "", mb_main, NULL, LABEL_HELP );
    XtSetArg ( args[ 0 ], XmNmenuHelpWidget, widget );
    XtSetValues ( mb_main, args, 1 );

/*
 *  Create the main form.
 */

    bform = create_form ( "", m_main );

/*
 *  Create the text widget to be used as the message window.  It is created
 *  with a vertical scrollbar to allow the user to page through displayed
 *  messages.
 */

    msg_scrll = create_text ( W_T_BULL, bform, "", 1, XmMULTI_LINE_EDIT, 0 );

/*
 *  Initialize the first iteration of the timer.  This will cause the update_bulletin
 *  callback routine to be executed.  This routine will reset the timer each time
 *  it completes its function.
 */
```

```
    id = XtAddTimeOut ( timer_interval, update_bulletin, NULL );

/*
 *  Call XtRealizeWidget on the top level widget to display the h_bulletin window.
 */

    XtRealizeWidget ( top );
/*
 *  Enter the Xtoolkit main loop to coordinate processing of all widget events.
 *  This loop is terminated when the user selects the exit command button and
 *  the associated callback procedure is executed to terminate this client.
 */

    XtMainLoop ( );
}
```

```
/*******************************************************************************
* MODULE NAME AND FUNCTION: exit_command()                                     *
*                                                                              *
*       The exit_command function is a callback procedure attached to the exit *
*       command button of the h_bulletin client.  This function causes the client *
*       to terminate naturally when the user selects the exit button.          *
*                                                                              *
*                                                                              *
* SPECIFICATION DOCUMENTS:                                                     *
*                                                                              *
*                                                                              *
*      /hisde/req/requirements                                                 *
*      /hisde/design/design                                                    *
*                                                                              *
*                                                                              *
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)        *
*                                                                              *
*       top (Widget) (I) - The top level form widget for the h_bulletin client. *
*                                                                              *
*                                                                              *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                          *
*                                                                              *
*      Nancy L. Martin - Software Engineering Section                          *
*                        Data System Science and Technology Department         *
*                        Automation and Data Systems Division                  *
*                        Southwest Research Institute                          *
*******************************************************************************/


#include <X11/Intrinsic.h>


/*
 *  Declare the top level widget.
 */

extern  Widget  top;


XtCallbackProc exit_command ( 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.
                                     */
            calldata;               /* Specifies any callback-specific data the widget
                                     * needs to pass to the client.  This parameter is
                                     * is not used by this function.
                                     */
{
/*
 *  Remove the top level widget and then close the h_bulletin display.
 */

    XtUnmapWidget ( top );
    XCloseDisplay ( XtDisplay(top) );


/*
 *  Exit the h_bulletin client with a zero.
 */
```

```
        exit(0);
}
```

```
/*****************************************************************************
* MODULE NAME AND FUNCTION: update_bulletin()                                *
*                                                                            *
*       This function is a timer callback procedure which is executed when the timer  *
*       interval expires.  This function updates the scroll window with the contents of *
*       the host bulletin log file if there have been messages added to the file.  *
*       (update_bulletin) determines whether there have been new messages added by  *
*       reading the position last written to the beginning of the file and comparing it *
*       to the value read from the file the last time an update was necessary.  If these *
*       numbers are not the same then the file has been updated.             *
*                                                                            *
*       Finally, update_bulletin reinitializes the timer value.  This will cause  *
*       update_bulletin to be called continually, at the specified interval, to update *
*       the host bulletin message scroll window when necessary.              *
*                                                                            *
*                                                                            *
* SPECIFICATION DOCUMENTS:                                                   *
*                                                                            *
*      /hisde/req/requirements                                               *
*      /hisde/design/design                                                  *
*                                                                            *
*                                                                            *
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)      *
*                                                                            *
*       bform (Widget) (I) -  The form widget created for the bulletin window.  *
*                                                                            *
*       timer_interval (unsigned long) (I) -                                 *
*               The interval used to set the timer for checking message queues.  This *
*               value is initialized to the the value defined as DEFAULT_INTERVAL in *
*               the h_advisory.h include file.  It may be changed in the command line *
*               when executing this client.  This value should be given in seconds.  *
*               It will be converted to milliseconds programmatically.       *
*                                                                            *
*       msg_scrll (Widget) (I/O) -                                           *
*               The file text widget created for the display of messages in the message *
*               window.  It is created with a vertical scroll bar on the left hand side *
*               to allow the user to page through displayed messages.        *
*                                                                            *
*                                                                            *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                        *
*                                                                            *
*      Nancy L. Martin - Software Engineering Section                        *
*                        Data System Science and Technology Department       *
*                        Automation and Data Systems Division                *
*                        Southwest Research Institute                        *
*****************************************************************************/


#include <stdio.h>
#include <fcntl.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <hisde.h>
#include <h_bulletin.h>
#include <h_logfiles.h>

/*
 *  Declare the timer interval value for use in starting the timer back up.
 */

extern  long    timer_interval;

/*
 *  Declare the widgets which are accessed for the update.
```

```
 */

extern  Widget  bform,
                msg_scrll;


XtTimerCallbackProc update_bulletin ( client_data, id )

    caddr_t         client_data;    /* Specifies the client date that was registered
                                     * registered for this procedure in XtAddTimeOut.
                                     */

    XtIntervalId    *id;            /* Specifies the ID returned from the corresponding
                                     * corresponding XtAddTimeOut call.
                                     */
{
    static  int     last_position = 0;
                                    /* The position value read from the file on the
                                     * previous update.
                                     */

    int             fd,             /* The file descriptor of the opened host bulletin
                                     * log file.
                                     */
                    new_position;   /* The value of the last position written to the
                                     * file.
                                     */

    char    position[POSITION_OFFSET + 1 ];
                                   /* The character string used to read in the last
                                    * position written to.
                                    */

/*
 *  Open the host bulletin log file for reading and read the value of
 *  the last position written to from the beginning of the file.
 */

    if ( ( fd = open ( HISDE_HOST_LOG, O_RDONLY ) ) <= NULL ) {
        h_message ( MSG_ERROR, "h_bulletin:  Cannot open host bulletin file." );
        exit ( -1 );
    }
    if ( read ( fd, position, POSITION_OFFSET ) != POSITION_OFFSET ) {
        h_message ( MSG_ERROR, "h_bulletin:  Cannot read host bulletin file position." );
        close ( fd );
        exit ( -1 );
    }

/*
 *  Convert the character string read from the file to an integer and compare
 *  it to the value read from the file on the previous update.  If the
 *  value has changed, assign the new position offset to the static variable,
 *  last_position, for use in the next pass through this function.  Next, call
 *  update_window to read the messages from the file and update the message
 *  scroll window.
 */

   new_position = atoi ( position );
   if ( new_position != last_position ) {
       last_position = new_position;
       update_window ( fd, new_position );
   }

/*
```

```
     *  After the window has been  updated, or if it did not need to be updated,
     *  close the host bulletin log file.
     */

        close ( fd );

    /*
     *  When the scroll window has been updated (if needed), reset the timer so that
     *  this routine will be called continually until the user selects to exit
     *  the h_bulletin client.
     */

        *id = XtAddTimeOut ( timer_interval, update_bulletin, NULL );
    }
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: update_window ( )                               *
 *                                                                           *
 *       This function is called to read in the host bulletin messages from the
 *       host bulletin log file starting with the oldest message.  As each message
 *       is read, it will be concatenated onto the end of the buffer to be written
 *       in the message scroll window.  When all message have been read from the
 *       the file, update_text_widget() is called with the buffer of host messages
 *       to update the message scroll window with the new messages.  The cursor will
 *       then be placed at the beginning of the newly added messages and the
 *       size of the message buffer is assigned to old_message_size for use during
 *       the next update.
 *
 *       In order to determine where the first message is in the circular log file,
 *       update_window will attempt to read the first message past the last position
 *       written to in the file.  If there is a message in this position then the
 *       file is full and this message is the oldest message.  If there is not a
 *       message following the last position written to, the file is not yet full
 *       and the oldest message is the first message in the file.
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *      /hisde/req/requirements
 *      /hisde/design/design
 *
 *
 * EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)
 *
 *       msg_scrll (Widget) (I/O) - Text widget created for display of host messages.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *      Nancy L. Martin - Software Engineering Section
 *                        Data System Science and Technology Department
 *                        Automation and Data Systems Division
 *                        Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <hisde.h>
#include <h_logfiles.h>


extern  Widget  msg_scrll;


update_window ( fd, new_position )

    int     fd,                     /* Specifies the file descriptor for the host
                                     * host bulletin log file.
                                     */

            new_position;           /* Specifies the last position written to the host
                                     * bulletin log file.
                                     */
{
    int     i,                      /* Used to initialize the message buffer to blanks.
                                     */
            position;               /* Maintains the current position in the file.
                                     */
```

```
    char    buffer      [ MAX_MESSAGE + 1  ];
                                    /* Used to read in each host message.
                                     */
    char    display_msg[ MAX_HOST_LOG + 1 ];
                                    /* Buffer which will contain all host messages.
                                     */


/*
 *  Initialize the scroll window buffer to blanks and assign the first position
 *  to be null for concatenation purposes.
 */

    for ( i = 0; i < MAX_HOST_LOG; i++ );
        display_msg[ i ] = BLANK;
    display_msg[ 0 ] = NULL;

/*
 *  Assign the last position written to as the position to seek to for reading.
 */

    position = new_position;

/*
 *  Try to read the next message after the most recently added message.  If
 *  the read fails, set the file position to the first message in the file
 *  past the position value, read that message, and assign the file position
 *  to be this message's starting point.
 *
 *  If neither read is successful, call h_message to inform the user that
 *  the host bulletin file cannot be read, close the file, and exit h_bulletin.
 */

    lseek ( fd, position, 0L );
    if ( read ( fd, buffer, MAX_MESSAGE ) <= 0 ) {
        lseek ( fd, POSITION_OFFSET, 0L );
        position = POSITION_OFFSET;
        if ( read ( fd, buffer, MAX_MESSAGE ) <= 0 ) {
            h_message ( MSG_ERROR, "h_bulletin: Cannot read first bulletin" );
            close ( fd );
            exit ( -1 );
        }
    }

/*
 *  If the oldest message was successfully read from the file, append a newline
 *  to the end of the message and then attach the message to the message buffer.
 *  Update the file position pointer to point to the next message.  Each message
 *  read from the file is the same size, MAX_MESSAGE.
 */

    strcat ( buffer, "\n" );
    strcat ( display_msg, buffer );
    position += MAX_MESSAGE;

/*
 *  If the new file position is greater than or equal to the maximum size of the
 *  host bulletin log file, wrap around to the first message in the file.  Note:
 *  the first message in the file is located after the value indicating the
 *  last position written to in the file.  This value is of the size,
 *  POSITION_OFFSET.
 */

    if ( position >= MAX_HOST_LOG )
```

```
        position = POSITION_OFFSET;

/*
 *  Loop through the file reading the next message until the end of file is reached
 *  or the file position returns to the oldest message.
 *
 *  For each message a newline is appended to the end of the message before it is
 *  attached to the end of the message buffer.  The file position is updated to
 *  point to the next message in the file each time.
 */

    while ( ( read ( fd, buffer, MAX_MESSAGE ) > 0 ) && ( position != new_position ) ) {
        strcat ( buffer, "\n" );
        strcat ( display_msg, buffer );
        position += MAX_MESSAGE;

/*
 *  If the new file position is greater than or equal to the maximum size of the
 *  host bulletin log file, wrap around to the first message in the file.  Note:
 *  the first message in the file is located after the value indicating the
 *  last position written to in the file.  This value is of the size,
 *  POSITION_OFFSET.
 */

        if ( position >= MAX_HOST_LOG ) {
            position = POSITION_OFFSET;
            lseek ( fd, position, 0L );
        }
    }

/*
 *  Update the text widget.
 */

    update_text_widget ( msg_scrll, display_msg );
    XmTextSetInsertionPosition ( msg_scrll, strlen ( display_msg ) );
}
```

```
#############################################################################################
# Makefile for HISDE user interface client (h_cm_menu)                                      #
#############################################################################################

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

TARGET      = h_cm_menu

#
# Initialize include and library search paths to include current directory and the
# HISDE directories.
#

BINDIR      = /hisde/bin
INCDIR      = /hisde/src/include
INCDIRS     = -I$(INCDIR)

#
# Define the libraries to search.  This includes the CM manager, user interface,
# HISDE main, and all X windows libraries.
#

LIBRARIES   = -lcmutil -lui -lhisde -lXm -lXt -lX11

#
# Define the compiler and linker flags.
#

CFLAGS      = -O $(INCDIRS)
LDFLAGS     = -O $(EXTRAFLAGS)

#
# Define all objects which make up this target.

OBJS        =\
            cbr_cm_trm.o\
            cbr_command.o\
            cbr_clear.o\
            set_to_insen.o\
            h_cm_menu.o

#
# Define all header files required.
#

HDRS        =\
            $(INCDIR)/h_cm_menu.h\
            $(INCDIR)/h_cm_menu.bit\
            $(INCDIR)/h_user_inter.h\
            $(INCDIR)/cm_util.h\
            $(INCDIR)/hisde.h

#
# Make the target.
#

all:        $(TARGET)

$(TARGET):  $(OBJS)
    $(CC) -o $@ $(OBJS) $(LIBRARIES) $(LDFLAGS)
    strip $(TARGET)
    mv $(TARGET) $(BINDIR)
```

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

```
/*****************************************************************************
* MODULE NAME AND FUNCTION: ( h_cm_menu )                                    *
*                                                                            *
* This client provides the user interface to the configuration management utilities *
* available on the local workstation.  These commands provide access to the CM manager *
* workstation and the central CM host.  This involves submitting jobs (applications) to *
* the CM manager workstation for compilation/loading, retrieving the executable files, *
* archiving the application to the central CM host, and finally, retrieving executable *
* files from the central CM host.  The entire list of commands is as follows:  *
*                                                                            *
*      SUBMIT - Submit a job (application) to the CM manager workstation for compilation *
*               and loading.                                                 *
*                                                                            *
*      STATUS - Obtain the status of a job active on the CM manager workstation. *
*                                                                            *
*      LISTDIR - List the contents of the directory corresponding to a job.  *
*                                                                            *
*      INFO - Display all information entered by the user for a submitted job. *
*                                                                            *
*      RETURN - Return the newly loaded executable files to the local workstation. *
*                                                                            *
*      ARCHIVE - Send all the files for a job to the central CM host.        *
*                                                                            *
*      CANCEL - Terminate a job and remove all associated files from the CM manager *
*               workstation.                                                 *
*                                                                            *
*      DOWNLOAD - Download a set of files resident on the central CM host, to the local *
*                 workstation.                                               *
*                                                                            *
*      HOSTDIR - Obtain a directory listing of files resident on the central CM host. *
*                                                                            *
* Note that the critical piece of data identifying a job is called the job control num- *
* ber.  When the user submits an application to the CM manager workstation, a unique job *
* control number will be assigned and returned to the user.  The majority of the remain- *
* ing commands will require entry of this number to identify the appropriate job.  Note *
* that some commands require additional data as well.  The entire list of data items is *
* as follows:                                                                *
*                                                                            *
*      Job Control Number - Identifies the specific job.  It is used for the STATUS, *
*                           LISTDIR, INFORMATION, RETURN, ARCHIVE, and CANCEL commands. *
*                                                                            *
*      Flight - Identifies the flight for which a job is to be certified for.  It is *
*               used in the SUBMIT command.                                  *
*                                                                            *
*      Directory - Identifies the source and/or destination of files.  It is used for *
*                  the SUBMIT and RETURN commands.                           *
*                                                                            *
*      Executables - Identifies a list of executables which will be returned to the *
*                    local workstation.  It is used for the SUBMIT and RETURN commands. *
*                                                                            *
*      Description - Provides a textual description of the application making up the *
*                    job.  It is used for the SUBMIT command.                *
*                                                                            *
* Note that some data items are required and others are optional.  A summarization of *
* the commands and required/optional data items is given below (Note that (R) indicates *
* a required item and (O) indicates an option item):                         *
*                                                                            *
*      SUBMIT    - (R) Flight                                                *
*                  (O) Directory                                             *
*                  (O) Executables                                           *
*                  (O) Description                                           *
*                                                                            *
*      STATUS    - (R) Job Control Number                                    *
*                                                                            *
```

```
*       LISTDIR  - (O) Job Control Number                                    *
*                                                                            *
*       INFO     - (R) Job Control Number                                    *
*                                                                            *
*       RETURN   - (R) Job Control Number                                    *
*                  (O) Directory                                             *
*                  (O) Executables                                           *
*                                                                            *
*       ARCHIVE  - (R) Job Control Number                                    *
*                                                                            *
*       CANCEL   - (R) Job Control Number                                    *
*                                                                            *
*       DOWNLOAD - (R) Flight                                                *
*                  (R) Directory                                             *
*                  (R) Executables                                           *
*                                                                            *
*       HOSTDIR -  (O) Flight                                                *
*                                                                            *
* When this client is executed, it will display a window which contains four distinct *
* functional sub-windows.  These include the following:                      *
*                                                                            *
*       ID Window - This window identifies the client and provides two commands.  These *
*                   include:                                                 *
*                                                                            *
*                        Clear - clear all input fields                      *
*                        Exit  - exit from this client                       *
*                                                                            *
*       Commands Window - This window contains each of the CM utility commands previously *
*                         discussed.                                         *
*                                                                            *
*       Input Window - This window contains the fields allowing entry of the data items *
*                      previously discussed.  Note that when no command is active, all *
*                      input fields will be set to an insensitive state.  In this state, *
*                      the borders and label text of the widgets will be displayed in a *
*                      different orientation and no mouse input will be acknowledged. *
*                                                                            *
*       Output Window - This window is used to display information output by certain *
*                       commands.                                            *
*                                                                            *
* In order to execute any of the CM utility commands, the user need simply select the *
* appropriate command button in the command window.  At this time, all applicable fields *
* will change to a sensitive state, in which they appear in their normal orientation and *
* mouse/keyboard input is allowed.  The user may now complete the appropriate fields and *
* then reselect the same command button to actually execute the command.  The user may *
* alternatively select any other command button to abort the command.        *
*                                                                            *
* It is important to note that the data in the input fields will not be cleared from *
* command to command.  This supports the basic sequence of CM commands, in which the *
* user submits a job (and receives a job control number), makes several status requests *
* to determine its state, retrieves the executables, and finally, archives the files to *
* the CM host.  With this sequence, the required data for each command is already pre- *
* sent in the input fields.  In addition, certain other fields provide information about *
* the makeup of the job (even though they do not affect subsequent commands).  Note that *
* if the user does need to clear data from fields, he may use the clear command or any *
* of the normal keyboard sequences which are supported by the text widget.  Note that *
* the clear command may be used in either command state (before and during data input). *
*                                                                            *
* Note that when a command is actually being executed, this client will not respond to *
* mouse and keyboard input.  Once the command is complete (call to the CM utility func- *
* tion), input will be accepted as normal.                                   *
*                                                                            *
* The majority of the CM commands return status information via messages to the standard *
* HISDE message client.  This includes successfull operation and error messages.  Also, *
* certain commands will return additional information which will be displayed in the *
```

```
* output window.  This includes the STATUS, LISTDIR, and HOSTDIR commands.  When such a *
* command executes, the current information in the output window will be removed and the *
* new data displayed.  This information will remain displayed until another such command *
* is used.  The output window will not be changed by commands which do not return this  *
* level of output information.                                                          *
*                                                                                       *
* To exit from this client, use the exit command in the ID window.  As with the clear   *
* command, it is possible to exit during the data input phase of a command.             *
*                                                                                       *
*                                                                                       *
* DESCRIPTION OF MAIN FUNCTION:                                                         *
*                                                                                       *
* This is the main function of the h_cm_menu client.  It is responsibile for initiali- *
* zation of the resource database and all widgets which make up the window.  Once all   *
* widgets and their associated callbacks are initialized and realized, this function    *
* calls the Xtoolkit intrinsic (XtMainLoop) to process all incoming events.  This in-   *
* cludes callbacks for the various command widgets (clear, exit, and CM command).       *
*                                                                                       *
* This function initializes a single hierarchy of widgets to present the menu of CM    *
* manager functions and input fields.  This consists of a main form and four child      *
* forms, one of which contains an additional form for each input field.  The complete   *
* hierarchy of widgets is summarized below:                                             *
*                                                                                       *
*      top ---> form -+-> form (Client  ) -+-> label                                    *
*                     |        (ID      )  +-> command (clear fields)                   *
*                     |                    +-> command (exit client)                    *
*                     |                                                                 *
*                     +--> form (CM      ) -+-> command (submit command)               *
*                     |          (Commands)  +-> command (status command)              *
*                     |                      +-> command (list dir command)            *
*                     |                      +-> command (info command)                *
*                     |                      +-> command (return command)              *
*                     |                      +-> command (archive command)             *
*                     |                      +-> command (cancel command)              *
*                     |                      +-> command (download command)            *
*                     |                      +-> command (host list command)           *
*                     |                                                                 *
*                     +--> form (Input   ) -+-> form (Job     ) -+-> label             *
*                     |          (Fields  )  |        (Number  )  +-> text (actual input) *
*                     |                      |                                          *
*                     |                      +-> form (Job     ) -+-> label             *
*                     |                      |        (Flight  )  +-> text (actual input) *
*                     |                      |                                          *
*                     |                      +-> form (Job     ) -+-> label             *
*                     |                      |        (Directry)  +-> text (actual input) *
*                     |                      |                                          *
*                     |                      +-> form (List Of ) -+-> label             *
*                     |                      |        (Executes)  +-> text (actual input) *
*                     |                      |                                          *
*                     |                      +-> form (Job     ) -+-> label             *
*                     |                               (Descrip )  +-> text (actual input) *
*                     |                                                                 *
*                     +--> form (Command ) -+-> label                                   *
*                               (Output  )  +-> text (actual input)                     *
*                                                                                       *
*                                                                                       *
* Each of the forms used is offset from other forms to maintain a consistent layout of  *
* information.  The widgets with each form are in turn offset from one another in the   *
* same way.  This insures that homogenous widgets remain in close proximity and in a    *
* sensible arrangement.                                                                 *
*                                                                                       *
* Once this function calls XtMainLoop, there are a number of callback events which may  *
* be executed.  These functions, the command widgets to which they are tied, and the    *
* operations they perform are as follows:                                               *
```

```
 *
 *      function                event               operation
 *      --------                -----               ---------
 *      cbr_clear               clear command       clear all input fields
 *      cbr_cm_terminate        exit command        terminate h_cm_menu client
 *      cbr_command             any CM command      execute requested CM command
 *
 * For more information on these callback functions, refer to the code in the appropriate
 * source code file.
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *      /hisde/req/requirements
 *      /hisde/design/design
 *
 *
 * EXECUTION SEQUENCE:
 *
 *      h_cm_menu
 *
 *
 * EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)
 *
 *      This function initializes all declared widget variables.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *      Mark D. Collier - Software Engineering Section
 *                        Data System Science and Technology Department
 *                        Automation and Data Systems Division
 *                        Southwest Research Institute
 *
 ************************************************************************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Cardinals.h>
#include <X11/Shell.h>
#include <Xm/MainW.h>
#include <Xm/RowColumn.h>
#include <Xm/Form.h>
#include <h_cm_menu.bit>
#include <hisde.h>
#include <h_user_inter.h>
#include <h_cm_menu.h>
#include <cm_util.h>


/*
 *  Declare all widgets which will be used by this client.  This data is made external
 *  to allow simple access in callback functions.
 */

Widget  top, m_main, widget,
        mb_main, mp_file, mp_edit, mp_cmd,
        f_input,  l_job,    t_job,
                  l_flight, t_flight,
                  l_exec,   t_exec,
                  l_dir,    t_dir,
                  l_desc,   t_desc,
        f_output, l_output, t_output;
```

```
/*
 *  Declare all the callback functions used by this client.
 */

extern  XtCallbackProc  cbr_cm_terminate(),
                        cbr_command     (),
                        cbr_clear       ();


main ( argc, argv )
    int     argc;
    char    **argv;
{
/*
 *  Initialize the callback lists required for the clear fields, exit client, and CM
 *  manager commands functions.  These callbacks occur when the user selects one of the
 *  associated command widgets.
 */

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

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

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

    Arg icon_arg,                   /* Argument which will be used to initialize
                                     * the graphic icon for this client.
                                     */

        args[ 1 ];                  /* Arguments used to initialize widget resources.
                                     */


/*
 *  Initialize the Xtoolkit, parse command line, and return the root widget which will be
 *  the parent of the window.  Note that this client does not have any application
 *  specific resources (NULL and ZERO parameters).
 */

    top = XtInitialize ( NAME_SHELL, NAME_APLIC, NULL, ZERO, &argc, argv );

/*
 *  If there were arguments on the command line which could not be parsed, call the
 *  function (bad_syntax) to report the error, display the correct syntax, and exit from
 *  the client.
 */

    if ( argc > 1 )
        bad_syntax ( "h_cm_menu" );

/*
 *  Initialize the icon bitmap for this client.
 */

    XtSetArg ( icon_arg, XtNiconPixmap,
                    XCreateBitmapFromData ( XtDisplay(top), XtScreen(top)->root,
```

```
                                h_cm_menu_bits, h_cm_menu_width, h_cm_menu_height ) );

    XtSetValues ( top, &icon_arg, ONE );

/*
 *  Create the main window widget and the menu bar which will contain all commands.
 */

    m_main = XmCreateMainWindow ( top, "", NULL, 0 );
    XtManageChild ( m_main );

    mb_main = XmCreateMenuBar ( m_main, "", NULL, 0 );
    XtManageChild ( mb_main );

/*
 *  Create pulldown for file commands.
 */

    mp_file = XmCreatePulldownMenu ( mb_main, "", NULL, 0 );
    create_cascade ( "", mb_main, mp_file, LABEL_FILE                    );
    create_command ( "", mp_file,          LABEL_EXIT, cb_cm_terminate );

/*
 *  Create pulldown for edit commands.
 */

    mp_edit = XmCreatePulldownMenu ( mb_main, "", NULL, 0 );
    create_cascade ( "", mb_main, mp_edit, LABEL_EDIT           );
    create_command ( "", mp_edit,          LABEL_CLEAR, cb_clear );

/*
 *  Create pulldown for CM commands.
 */

    mp_cmd = XmCreatePulldownMenu ( mb_main, "", NULL, 0 );
    create_cascade ( NULL, mb_main, mp_cmd, LABEL_COMMANDS );

    cb_command[0].closure = (caddr_t)SUBMIT;
    create_command ( "", mp_cmd, LABEL_CMD_SUBMIT,    cb_command );
    cb_command[0].closure = (caddr_t)STATUS;
    create_command ( "", mp_cmd, LABEL_CMD_STATUS,    cb_command );
    cb_command[0].closure = (caddr_t)LISTDIR;
    create_command ( "", mp_cmd, LABEL_CMD_LISTDIR,   cb_command );
    cb_command[0].closure = (caddr_t)INFORMATION;
    create_command ( "", mp_cmd, LABEL_CMD_INFO,      cb_command );
    cb_command[0].closure = (caddr_t)RETURN;
    create_command ( "", mp_cmd, LABEL_CMD_RETURN,    cb_command );
    cb_command[0].closure = (caddr_t)ARCHIVE;
    create_command ( "", mp_cmd, LABEL_CMD_ARCHIVE,   cb_command );
    cb_command[0].closure = (caddr_t)CANCEL;
    create_command ( "", mp_cmd, LABEL_CMD_CANCEL,    cb_command );
    cb_command[0].closure = (caddr_t)DOWNLOAD;
    create_command ( "", mp_cmd, LABEL_CMD_DOWNLOAD, cb_command );
    cb_command[0].closure = (caddr_t)HOSTDIR;
    create_command ( "", mp_cmd, LABEL_CMD_HOSTDIR,   cb_command );

/*
 *  Create the help cascade.
 */

    widget = create_cascade ( "", mb_main, NULL, LABEL_HELP );
    XtSetArg ( args[ 0 ], XmNmenuHelpWidget, widget );
    XtSetValues ( mb_main, args, 1 );
```

```
/*
 *  Create the form used for the work area.
 */

    f_input = create_form ( W_F_INPUT, m_main );

/*
 *  Initialize the label and text widget for the job control number input field.
 *  Note this this and all text widgets are editable.
 */

    l_job   = create_label ( W_L_INPUT_JOB, f_input, LABEL_JOB );
    t_job   = create_text  ( W_T_INPUT_JOB, f_input, "", 0, XmSINGLE_LINE_EDIT, 1 );
    XmAddTabGroup ( t_job );

/*
 *  Initialize the label and text widget for the flight input field.
 */

    l_flight = create_label ( W_L_INPUT_FLIGHT, f_input, LABEL_FLIGHT );
    t_flight = create_text  ( W_T_INPUT_FLIGHT, f_input, "", 0, XmSINGLE_LINE_EDIT, 1 );
    XmAddTabGroup ( t_flight );

/*
 *  Initialize the label and text widget for the list of executables input
 *  field.  Note that this text widget includes a vertical scrollbar.
 */

    l_exec = create_label ( W_L_INPUT_EXEC, f_input, LABEL_EXEC );
    t_exec = create_text  ( W_T_INPUT_EXEC, f_input, "", 1, XmMULTI_LINE_EDIT, 1 );
    XmAddTabGroup ( t_exec );

/*
 *  Initialize the label and text widget for the source/destination directory
 *  field Note that thsi text widget includes a vertical scrollbar.
 */

    l_dir = create_label ( W_L_INPUT_DIR, f_input, LABEL_DIR );
    t_dir = create_text  ( W_T_INPUT_DIR, f_input, "", 0, XmSINGLE_LINE_EDIT, 1 );
    XmAddTabGroup ( t_dir );

/*
 *  Initialize the label and text widget for the job description field.
 *  Note that this text widget includes a vertical scrollbar.
 */

    l_desc = create_label ( W_L_INPUT_DESC, f_input, LABEL_DESC);
    t_desc = create_text  ( W_T_INPUT_DESC, f_input, "", 1, XmMULTI_LINE_EDIT, 1 );
    XmAddTabGroup ( t_desc );

/*
 *  Initialize the shell window for the output information area. It includes a label and
 *  a text widget.
 */

    f_output = XmCreateFormDialog ( top, W_F_OUTPUT, NULL, 0 );
    l_output = create_label ( W_L_OUTPUT, f_output, LABEL_OUTPUT);
    t_output = create_text  ( W_T_OUTPUT, f_output, "", 1, XmMULTI_LINE_EDIT, 0 );
    XtManageChild ( f_output );

/*
 *  Define the areas which constitute the main window widget.
 */
```

```
    XmMainWindowSetAreas ( m_main, mb_main, NULL, NULL, NULL, f_input );

/*
 *  Realize the top level widget.  This causes the main form of this client to be
 *  displayed, along with all child widgets.
 */

    XtRealizeWidget ( top );

/*
 *  Call the (set_to_insensitive) function to set all input fields (label and
 *  text widgets) to their initial insensitive state.  In this state, their visual
 *  orientation is different and no mouse/keyboard input is acknowledged.
 */

    set_to_insensitive ( );

/*
 *  Enter the normal Xtoolkit main loop, which coordinates processing of the various
 *  widget events.  This loop will terminate normally when the user selects the "Exit"
 *  command, which in turn causes the cbr_cm_terminate callback routine to be executed.
 */

    XtMainLoop ( );
}
```

```
/*****************************************************************************
* MODULE NAME AND FUNCTION ( cbr_command )                                   *
*                                                                            *
* This callback function is executed whenever the user selects one of the command wid-  *
* gets used to present the available CM manager functions.  This function is set up to   *
* process all commands.  It determines the command which was selected and peforms the    *
* actions necessary to execute it.                                           *
*                                                                            *
* Before any command is selected, all input fields (form/label/text widgets) are set to  *
* an insensitive state.  Once a command is selected, this function will set to sensi-    *
* tive, all fields which are required or optional for the command.  At this point, the   *
* the user may either enter data and then reselect the same command (which causes the    *
* command to be executed), or he may select any other CM manager command (which causes   *
* the command to be aborted).  This process is allowed by separating the actions taken   *
* when the command is first selected and those taken when the command is selected a sec-  *
* ond time.                                                                  *
*                                                                            *
* When a command is first selected, this only actions taken by this function are to set  *
* the appropriate fields to sensitive.                                       *
*                                                                            *
* When a command is selected a second time (assuming that it is the same command), all   *
* required fields are checked, the input fields are reset to insensitive, the CM manager *
* is called to execute the requested function, and the return data is displayed in the   *
* output window (if applicable).  Note that output may also be in the form of messages   *
* sent to the HISDE message client.  Note also that if the user omits a required piece   *
* of data, a message will be output and the command will remain in its entry state (in-  *
* put of data).                                                              *
*                                                                            *
*                                                                            *
* SPECIFICATION DOCUMENTS:                                                   *
*                                                                            *
*      /hisde/req/requirements                                               *
*      /hisde/design/design                                                  *
*                                                                            *
*                                                                            *
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)      *
*                                                                            *
*      t_output     (Widget) (I/O) - Pointer to the text widget used for the output text. *
*                                    It recreated by this function when the CM manager   *
*                                    returns an output buffer.               *
*                                                                            *
*      t_job        (Widget) (I)   - Pointer to the text widget containing the job data.  *
*                                    This widget is needed to clear the text.*
*                                                                            *
*      t_flight     (Widget) (I)   - Pointer to the text widget containing the flight     *
*                                    data.                                   *
*                                                                            *
*      t_dir        (Widget) (I)   - Pointer to the text widget containing the directory  *
*                                    data.                                   *
*                                                                            *
*      t_exec       (Widget) (I)   - Pointer to the text widget containing the list of    *
*                                    executables.                            *
*                                                                            *
*      t_desc       (Widget) (I)   - Pointer to the text widget containing the job desc-  *
*                                    cription.                               *
*                                                                            *
*      l_job        (Widget) (I)   - Pointer to the label widget containing the job data. *
*                                    This widget is needed to make the field appear      *
*                                    insensitive.                            *
*                                                                            *
*      l_flight     (Widget) (I)   - Pointer to the label widget containing the flight    *
*                                    data.                                   *
*                                                                            *
*      l_dir        (Widget) (I)   - Pointer to the label widget containing the directory *
```

```
*                                   data.                                      *
*                                                                              *
*      l_exec      (Widget) (I)   - Pointer to the label widget containing the list of
*                                   executables.                               *
*                                                                              *
*      l_desc      (Widget) (I)   - Pointer to the label widget containing the job desc-
*                                   cription.                                  *
*                                                                              *
*                                                                              *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                          *
*                                                                              *
*      Mark D. Collier - Software Engineering Section                          *
*                        Data System Science and Technology Department         *
*                        Automation and Data Systems Division                  *
*                        Southwest Research Institute                          *
*******************************************************************************/


#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <hisde.h>
#include <h_user_inter.h>
#include <h_cm_menu.h>
#include <cm_util.h>


extern  Widget  f_output, t_output,
                t_job, t_exec, t_flight, t_dir, t_desc,
                l_job, l_exec, l_flight, l_dir, l_desc;


XtCallbackProc cbr_command ( widget, closure, calldata )

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

    caddr_t closure,                /* Callback specific data.  This parameter will be
                                     * be set to a value which identifies the selected
                                     * command.
                                     */
            calldata;               /* Specifies any callback-specific data the widget
                                     * needs to pass to the client.  This parameter is
                                     * is not used by this function.
                                     */
{
    static int      in_command,     /* This variable indicates the state of the current
                                     * command.  If FALSE, no command is currently ac-
                                     * tive; if TRUE, a command has been selected and
                                     * the user is allowed to input data.
                                     */
                    command;        /* When (in_command) is TRUE, this variable will be
                                     * set to the command selected.  It is used to deter-
                                     * mine if the user selected the same command (exe-
                                     * cute) or a different command (abort).
                                     */

    int             job_num;        /* For the SUBMIT command, this variable will be set
                                     * to the assigned job number.
                                     */

    register int    temp_command,   /* Set to the last command selected.  It is compared
                                     * to (command) to determine if the user wants to
                                     * execute or abort a command.
```

```
                                */
                len;            /* Temporary variable used to save the length of
                                 * certain strings.
                                 */

    char        *temp_buffer,   /* Pointer which will be updated by the CM manager
                                 * function when a status buffer is returned.
                                 */
                *temp_job,      /* Temporary buffer for the job number.  The normal
                                 * data is placed in this buffer to allow the CM
                                 * manager function to update it without adversely
                                 * affecting the widget.
                                 */
                *temp_dir,      /* Temporary buffer for the filename.
                                 */
                *temp_flight,   /* Temporary buffer for the flight.
                                 */
                *temp_exec,     /* Temporary buffer for the executables.
                                 */
                *temp_desc,     /* Temporary buffer for the description.
                                 */
                job_number[ 6 ]; /* String needed to format and display a returned
                                 * job number.
                                 */

/*
 *  If just starting a command (flag in_command is FALSE), then determine which command
 *  was selected and save in the static variable (command).  This value is required to
 *  indicate which command was initially selected.  Also set the static variable
 *  (in_command) to TRUE to indicate the state of the command.
 */

    if ( in_command == FALSE ) {
        command    = (int)closure;
        in_command = TRUE;

/*
 *      Based on the selected command, set the appropriate input text and label widgets
 *      to a sensitive state.  This allows the user to enter data into the fields.
 *      Also clear any data which is not relevant to the operation.
 */


        if ( command == DOWNLOAD ) {
            XtSetSensitive ( t_flight, TRUE );
            XtSetSensitive ( t_dir,    TRUE );
            XtSetSensitive ( t_exec,   TRUE );
            XtSetSensitive ( l_flight, TRUE );
            XtSetSensitive ( l_dir,    TRUE );
            XtSetSensitive ( l_exec,   TRUE );
            clear_text_widget ( t_job  );
            clear_text_widget ( t_desc );

        } else if ( command == HOSTDIR ) {
            XtSetSensitive ( t_flight, TRUE );
            XtSetSensitive ( l_flight, TRUE );
            clear_text_widget ( t_job  );
            clear_text_widget ( t_dir  );
            clear_text_widget ( t_desc );
            clear_text_widget ( t_exec );

        } else if ( command == RETURN ) {
            XtSetSensitive ( t_job,  TRUE );
            XtSetSensitive ( t_dir,  TRUE );
```

```
            XtSetSensitive ( t_exec, TRUE );
            XtSetSensitive ( l_job,  TRUE );
            XtSetSensitive ( l_dir,  TRUE );
            XtSetSensitive ( l_exec, TRUE );
            clear_text_widget ( t_flight );
            clear_text_widget ( t_desc   );

        } else if ( command == SUBMIT ) {
            XtSetSensitive ( t_flight, TRUE );
            XtSetSensitive ( t_dir,    TRUE );
            XtSetSensitive ( t_exec,   TRUE );
            XtSetSensitive ( t_desc,   TRUE );
            XtSetSensitive ( l_flight, TRUE );
            XtSetSensitive ( l_dir,    TRUE );
            XtSetSensitive ( l_exec,   TRUE );
            XtSetSensitive ( l_desc,   TRUE );
            clear_text_widget ( t_job );

        } else {
            XtSetSensitive ( t_job, TRUE );
            XtSetSensitive ( l_job, TRUE );
            clear_text_widget ( t_flight );
            clear_text_widget ( t_dir    );
            clear_text_widget ( t_exec   );
            clear_text_widget ( t_desc   );
        }

/*
 *  Otherwise, we are in the second phase of the command.
 */

    } else {

/*
 *      To complete a command, the appropriate command widget must be selected again.
 *      Examine the command which was selected, if different, assume that the user
 *      wants the command aborted.  In this case, set all input fields to the insen-
 *      setive state, output a message, and reset the command state (set in_command to
 *      FALSE).
 */

        temp_command = (int)closure;
        if ( command != temp_command ) {
            set_to_insensitive ( );
            display_message ( MSG_WARNING, "Command was aborted - make a new selection" );
            in_command = FALSE;
            return;
        }

/*
 *      Get the data in each of the text widgets.
 */

        temp_job    = get_text_widget ( t_job    );
        temp_flight = get_text_widget ( t_flight );
        temp_dir    = get_text_widget ( t_dir    );
        temp_exec   = get_text_widget ( t_exec   );
        temp_desc   = get_text_widget ( t_desc   );

/*
 *      Verify that the user has completed all required input fields.  If a required
 *      field was omitted, output a message and exit from this function.  In this case,
 *      the command will still be in effect.  Note that if the LISTDIR command is left
 *      blank, it is assumed that the user requires status on all active jobs.  This
```

```
 *      requires setting the job number to a constanct recognized by the CM manager.
 *      For a discussion of the CM commands and the fields which are required for each,
 *      refer to the header block comment in the main (h_cm_menu) function.
 */

        if ( command == DOWNLOAD ) {
            if ( strlen(temp_flight) == 0 ||  strlen(temp_exec) == 0 ) {
                display_message ( MSG_WARNING,
                                  "Flight and executable fields are required" );
                return;
            }
        } else if ( command != SUBMIT && command != HOSTDIR ) {
            if ( ( strlen(temp_job) == 0 ) || ( ( job_num = atoi(temp_job) ) == 0 ) )
                if ( command == LISTDIR )
                    job_num = LIST_ALL_PROCESSES;
                else {
                    display_message ( MSG_WARNING,
                                      "The job control number field is required" );
                    return;
                }
        } else if ( command == SUBMIT ) {
            if ( strlen ( temp_flight ) == 0 ) {
                display_message ( MSG_WARNING, "The flight number field is required" );
                return;
            }
        }
/*
 *      Control will reach this point if all data entered by the user is valid.  This
 *      requires that all input fields be set to insensitive to indicate that they may
 *      no longer be changed.
 */

        set_to_insensitive ( );

/*
 *  Insure that the last entry in the list of executables is terminated by a newline.
 *  This is done to simplify processing by the CM manager function.
 */

        if ( command == SUBMIT || command == RETURN || command == DOWNLOAD )
            if ( ( len = strlen ( temp_exec ) ) && temp_exec[len-1] != '\n' ) {
                temp_exec[  len] = '\n';
                temp_exec[++len] = NULL;
            }

/*
 *      Call the CM manager to execute the requested command with the supplied data.  If
 *      the SUBMIT command was selected, the assigned job number will be returned in the
 *      (job_num) variable.  The (temp_buffer) variable will be updated to point to the
 *      output of the command (this pointer will be NULL for commands which do not
 *      return any output).  Note that if this function returns a non-zero value if
 *      a fatal error occurred.
 */

        if ( cm_command ( command, &job_num, temp_flight, temp_dir, temp_exec,
                          temp_desc, &temp_buffer ) == 0 ) {

/*
 *          If the command selected was SUBMIT, then the CM manager will have returned
 *          the number assigned to the submitted job.  Take this value, convert to ascii,
 *          and use the user interface library function (update_text_widget) to place
 *          the value in the job number text widget (j_job).
 */
```

```
            if ( command == SUBMIT ) {
                sprintf ( job_number, "%05d", job_num );
                update_text_widget ( t_job, job_number );

/*
 *          If the command selected was INFORMATION, then the CM manager will have
 *          returned submit information in the flight, directory, executables, and
 *          description parameters.  Use the data in these parameters to update the
 *          appropriate text widgets.
 */

            } else if ( command == INFORMATION ) {
                update_text_widget ( t_flight, temp_flight );
                update_text_widget ( t_dir,    temp_dir    );
                update_text_widget ( t_exec,   temp_exec   );
                update_text_widget ( t_desc,   temp_desc   );
            }

/*
 *          If the command returned a status buffer of information (temp_buffer not NULL),
 *          update text widget.
 */

            if ( temp_buffer != NULL ) {
                update_text_widget ( t_output, temp_buffer );
                free ( temp_buffer );
            }

/*
 *      Otherwise (cm_command) call failed, so output a message (in addition to detailed
 *      message by CM.
 */
        } else
            display_message ( MSG_ERROR, "CM command failed - see advisory window" );

/*
 *      Set the state flag to indicate that the command is complete.
 */

        in_command = FALSE;

/*
 *  Free all memory allocated for text strings.
 */

        XtFree ( temp_job    );
        XtFree ( temp_flight );
        XtFree ( temp_dir    );
        XtFree ( temp_exec   );
        XtFree ( temp_desc   );
    } /* Of command state if */
}
```

```
/*****************************************************************************
* MODULE NAME AND FUNCTION ( set_to_insensitive )                            *
*                                                                            *
* This function is used to set each of the data input widgets to an insensitive state.   *
* Once in this state, the borders and labels of the widget are modified so that they are *
* less vivid (every other pixel is turned off).  In addition, insensitive widgets do not *
* recognize mouse pointer events.  This prevents the user from modifying any data within *
* a widget in this state.  This makes it easy for the user to identify data which is not *
* required for a given function.                                             *
*                                                                            *
*                                                                            *
* SPECIFICATION DOCUMENTS:                                                   *
*                                                                            *
*      /hisde/req/requirements                                               *
*      /hisde/design/design                                                  *
*                                                                            *
*                                                                            *
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)      *
*                                                                            *
*      t_job     (Widget) (I) - The text widget containing the job number.   *
*                                                                            *
*      t_flight  (Widget) (I) - The text widget containing the flight.       *
*                                                                            *
*      t_dir     (Widget) (I) - The text containing the destination directory.*
*                                                                            *
*      t_exec    (Widget) (I) - The text widget containing the list of executable files.  *
*                                                                            *
*      t_desc    (Widget) (I) - The text widget containing the job description.*
*                                                                            *
*      l_job     (Widget) (I) - The label widget containing the job number.  *
*                                                                            *
*      l_flight  (Widget) (I) - The label widget containing the flight.      *
*                                                                            *
*      l_dir     (Widget) (I) - The label containing the destination directory.*
*                                                                            *
*      l_exec    (Widget) (I) - The label widget containing the list of executable files. *
*                                                                            *
*      l_desc    (Widget) (I) - The label widget containing the job description.*
*                                                                            *
*                                                                            *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                        *
*                                                                            *
*      Mark D. Collier - Software Engineering Section                        *
*                        Data System Science and Technology Department       *
*                        Automation and Data Systems Division                *
*                        Southwest Research Institute                        *
*****************************************************************************/


#include <X11/Intrinsic.h>


extern  Widget  t_job,      l_job,
                t_flight,   l_flight,
                t_dir,      l_dir,
                t_exec,     l_exec,
                t_desc,     l_desc;


int set_to_insensitive ( )
{
/*
 *  Use the Xtoolkit intrinsic XtSetSensitive function to set each of the text and label
 *  widgets to the insensitive state.
```

```
*/

    XtSetSensitive ( t_job,     FALSE );
    XtSetSensitive ( t_flight,  FALSE );
    XtSetSensitive ( t_dir,     FALSE );
    XtSetSensitive ( t_exec,    FALSE );
    XtSetSensitive ( t_desc,    FALSE );

    XtSetSensitive ( l_job,     FALSE );
    XtSetSensitive ( l_flight,  FALSE );
    XtSetSensitive ( l_dir,     FALSE );
    XtSetSensitive ( l_exec,    FALSE );
    XtSetSensitive ( l_desc,    FALSE );
}
```

```
/*****************************************************************************
* MODULE NAME AND FUNCTION ( cbr_clear )                                     *
*                                                                            *
* This callback function is called when the user selects the clear command.  It simply   *
* clears all data from each of the input fields.  This provides a convienient means of   *
* initializing the fields when the user needs to enter data which is radically different *
* from that currently displayed.                                             *
*                                                                            *
*                                                                            *
* SPECIFICATION DOCUMENTS:                                                   *
*                                                                            *
*      /hisde/req/requirements                                               *
*      /hisde/design/design                                                  *
*                                                                            *
*                                                                            *
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)      *
*                                                                            *
*      t_job       (Widget) (I) - Pointer to the text widget containing the job number.  *
*                                                                            *
*      t_flight    (Widget) (I) - Pointer to the text widget containing the flight.      *
*                                                                            *
*      t_dir       (Widget) (I) - Pointer to the text widget containing the destination  *
*                                 directory.                                 *
*                                                                            *
*      t_exec      (Widget) (I) - Pointer to the text widget containing the list of      *
*                                 executable files.                          *
*                                                                            *
*      t_desc      (Widget) (I) - Pointer to the text widget containing the job des-     *
*                                 cription.                                  *
*                                                                            *
*                                                                            *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                        *
*                                                                            *
*      Mark D. Collier - Software Engineering Section                        *
*                        Data System Science and Technology Department       *
*                        Automation and Data Systems Division                *
*                        Southwest Research Institute                        *
*****************************************************************************/


#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <h_user_inter.h>


extern  Widget  t_job,
                t_flight,
                t_dir,
                t_exec,
                t_desc;


XtCallbackProc cbr_clear ( 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.
                                     */

            calldata;               /* Specifies any callback-specific data the widget
                                     * needs to pass to the client.  This parameter is
                                     * is not used by this function.
```

```
*/
{
/*
 *  Use the HISDE user interface library function (clear_text_widget) to clear each of
 *  the text input widgets.
 */

    clear_text_widget ( t_job    );
    clear_text_widget ( t_flight );
    clear_text_widget ( t_dir    );
    clear_text_widget ( t_exec   );
    clear_text_widget ( t_desc   );
}
```

```
/*****************************************************************************
* MODULE NAME AND FUNCTION ( cbr_cm_terminate )                              *
*                                                                            *
* This callback function is called when the user selects the exit command widget.  This  *
* function destroys the top level widget, which causes the entire hierarchy of widgets   *
* to be destroyed.                                                           *
*                                                                            *
*                                                                            *
* SPECIFICATION DOCUMENTS:                                                   *
*                                                                            *
*      /hisde/req/requirements                                               *
*      /hisde/design/design                                                  *
*                                                                            *
*                                                                            *
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)      *
*                                                                            *
*      top (Widget) (I) - Pointer to the root widget of the window.          *
*                                                                            *
*                                                                            *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                        *
*                                                                            *
*      Mark D. Collier - Software Engineering Section                        *
*                        Data System Science and Technology Department       *
*                        Automation and Data Systems Division                *
*                        Southwest Research Institute                        *
*****************************************************************************/


#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>


extern  Widget  top;


XtCallbackProc cbr_cm_terminate ( 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.
                                     */
            calldata;               /* Specifies any callback-specific data the widget
                                     * needs to pass to the client.  This parameter is
                                     * is not used by this function.
                                     */
{
    XEvent  event;                  /* Event structure needed to make the calls to the
                                     * XtNextEvent and XtDispatchEvent functions.
                                     */

/*
 *  Destroy the root application shell widget and thereby, all subordinate widgets which
 *  make up the window and any popup windows used for menus.
 */

    XtDestroyWidget ( top );

/*
 *  Determine if any events have been queued.  These will normally be events which
 *  cause the widgets destroy callback to be executed.  Waiting and then processing
 *  the events insures that all data structures initialized by the widgets are
```

```
     *  properly deallocated.
     */

        XtNextEvent     ( &event );
        XtDispatchEvent ( &event );

    /*
     *  Close the display to deallocate any connections set up by X Windows.  Next
     *  exit from the client.
     */

        XCloseDisplay ( XtDisplay ( top ) );
        exit ( 0 );
    }
```

```
###############################################################################
# Makefile for HISDE user interface client (h_cmd).                           #
###############################################################################

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

TARGET      = h_cmd

#
# Initialize include and library search paths to include current directory and the
# HISDE directories.  Note that the library path also includes the user interface
# library.
#

BINDIR          = /hisde/bin
INCDIR          = /hisde/src/include
INCDIRS         = -I. -I$(INCDIR)

#
# Define the libraries to search.  This includes the HISDE library, the local user
# interface library, and all required X libraries.
#

LIBRARIES       = -lui -lhisde -lXm -lXt -lX11

#
# Define the compiler and linker flags.
#

CFLAGS          = -O $(INCDIRS)
LDFLAGS         = -O $(EXTRAFLAGS)

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

OBJS        =\
            cbr_cmd_trm.o\
            cbr_command.o\
            load_cmds.o\
            save_cmds.o\
            get_home_dir.o\
            h_cmd.o

#
# Define all header files required.
#

HDRS        =\
            $(INCDIR)/h_cmd.h\
            $(INCDIR)/h_cmd.bit\
            $(INCDIR)/h_user_inter.h\
            $(INCDIR)/hisde.h

#
# Make the target.
#

all:        $(TARGET)

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

```
        strip $(TARGET)
        mv $(TARGET) $(BINDIR)

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

```
/*****************************************************************************
* MODULE NAME AND FUNCTION ( cbr_command )                                   *
*                                                                            *
* This callback function is activated when the user wants to execute a command.  It gets *
* the currently highlighted text from the command list text widget, determines how it is *
* to be executed, and then actually executes the command.                    *
*                                                                            *
* Note that this function is called for each of the four command widgets.  The (closure) *
* parameter will be set to a value which indicates which command was selected.  These    *
* widgets and the manner in which they cause the command to be executed are as follows:  *
*                                                                            *
*      NO WIN/NO ICON - This widget is used to execute a command which does not require  *
*                       initialization of a controlling window (X and HISDE clients).    *
*                                                                            *
*      WINDOW/NO ICON - This widget is used to execute a command which requires initial- *
*                       ization of a controlling window (normal UNIX commands).  It runs *
*                       an xterm window with a Bourne shell, which in turn executes the  *
*                       users command.  Note that when the command is complete, the user *
*                       is required to press the RETURN key to cause the window to be    *
*                       terminated.  This is necessary, as many commands complete as     *
*                       soon as they finish output of data.                              *
*                                                                            *
*      NO WIN/ICON    - This widget is used to execute a command which does not require  *
*                       initialization of a controlling window.  However, note that it   *
*                       executes the command in an iconic state.                         *
*                                                                            *
*      WINDOW/ICON    - This widget is used to execute a command which requires initial- *
*                       ization of a controlling window.  However, note that it executes *
*                       the command in an iconic state.                                  *
*                                                                            *
* Note that this function will not execute a command which spans multiple lines.  Errors *
* will be reported to the root or xterm window.                              *
*                                                                            *
*                                                                            *
* SPECIFICATION DOCUMENTS:                                                   *
*                                                                            *
*      /hisde/req/requirements                                               *
*      /hisde/design/design                                                  *
*                                                                            *
*                                                                            *
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)      *
*                                                                            *
*      t_list       (char []) (I) - Widget containing the command list.  This variable  *
*                                   is required in order to determine which command was *
*                                   highlighted by the user.                            *
*                                                                            *
*                                                                            *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                        *
*                                                                            *
*      Mark D. Collier - Software Engineering Section                        *
*                        Data System Science and Technology Department       *
*                        Automation and Data Systems Division                *
*                        Southwest Research Institute                        *
*****************************************************************************/


#include <X11/Intrinsic.h>
#include <string.h>
#include <hisde.h>
#include <h_user_inter.h>
#include <h_cmd.h>

#define AMPERSAND   '&'
```

```
extern  Widget  t_list;


XtCallbackProc cbr_command ( widget, closure, calldata )

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

    caddr_t closure,                /* Set to a value which indicates whether the command
                                     * is to be executed in a window and/or initialized
                                     * as an icon.  It will be one of the following
                                     * values:
                                     *
                                     *      NOWIN_NOICON - No window and no icon
                                     *      WIN_NOICON   - Window and no icon
                                     *      NOWIN_ICON   - No window and icon
                                     *      WINDOW_ICON  - Window and icon
                                     */
            calldata;               /* Specifies any callback-specific data the widget
                                     * needs to pass to the client.  This parameter is
                                     * is not used by this function.
                                     */
{
    register char   *ptr;           /* Temporary pointer used to scan the command (cmd)
                                     * for an ampersand.
                                     */

    char    *cmd,                   /* Set to the command which is highlighted by the
                                     * user.  It must be formatted before actually
                                     * used.
                                     */
            command[ FMT_SIZE + 1 ];/* Set to the final formatted command which will
                                     * actually be executed.
                                     */

/*
 *  Get the currently highlighted text.
 */

    cmd = get_text_sel_widget ( t_list );

/*
 *  If a command was not specified (no highlighted text), output a warning message and
 *  return.
 */

    if ( cmd == NULL ) {
        display_message ( MSG_WARNING,
                          "No command specified - Highlight the desired command" );
        return;
    }

/*
 *  Determine if the command contains a newline (multiple lines).  If so, output a warning
 *  and return.
 */

    if ( strchr ( cmd, NEWLINE ) || strlen ( cmd ) > CMD_SIZE ) {
        display_message ( MSG_WARNING,
                          "Command contains a newline or command is too long" );
        return;
    }
```

```
/*
 *  If the command includes an ampersand ('&'), remove it, as all commands are auto-
 *  matically run in the background.
 */

    if ( ptr = strchr ( cmd, AMPERSAND ) )
        *ptr = ' ';

/*
 *  Based on the command widget selected by the user, initialize the final command.
 */

    if      ( (int) closure == NOWIN_NOICON )
        sprintf ( command, FMT_CMD,     cmd );

    else if ( (int) closure == WIN_NOICON   )
        sprintf ( command, FMT_CMD_W,   cmd );

    else if ( (int) closure == NOWIN_ICON   )
        sprintf ( command, FMT_CMD_I,   cmd );

    else if ( (int) closure == WIN_ICON     )
        sprintf ( command, FMT_CMD_W_I, cmd );

/*
 *  Actually execute the command.  If an error occurs, output a warning to the system
 *  message client.
 */

    if ( system ( command ) )
        display_message ( MSG_WARNING, "Could not execute the specified command" );
}
```

```
/*****************************************************************************
* MODULE NAME AND FUNCTION: ( h_cmd )
*
* This HISDE client provides a means by which users can execute normal UNIX commands.
* It is intended to provide a reasonable alternative to the UNIX shell.  While at the
* current time it does not provide any type of command checking, it would be very easy
* to add.  While this client is not intended to replace the UNIX shell, it does provide
* a reasonable command interface in an environment where UNIX commands are required, but
* not frequently used.
*
* When this client executes, it first examines the user's home directory for a .history
* file.  If found, the file will be opened and all commands read.  These commands will
* then be displayed in a large text widget which dominates the window presented by this
* client.  This widget (which contains a scrollbar), allows the user to add, change,
* delete, and duplicate commands.  The user may of course also execute a command, which
* is done by selecting (with the mouse cursor) the command to be executed.  Once the
* command is selected (highlighted), the user must select the manner in which the com-
* mand is executed.  This will be via one of the four command widgets located beneath
* the command list text widget.  The four widgets and the manner in which they execute
* commands is as follows:
*
*      NO WIN/NO ICON - This widget is used to execute a command which does not require
*                       initialization of a controlling window (X and HISDE clients).
*
*      WINDOW/NO ICON - This widget is used to execute a command which requires initial-
*                       ization of a controlling window (normal UNIX commands).  It runs
*                       an xterm window with a Bourne shell, which in turn executes the
*                       users command.  Note that when the command is complete, the user
*                       is required to press the RETURN key to cause the window to be
*                       terminated.  This is necessary, as many commands complete as
*                       soon as they finish output of data.
*
*      NO WIN/ICON    - This widget is used to execute a command which does not require
*                       initialization of a controlling window.  However, note that it
*                       executes the command in an iconic state.
*
*      WINDOW/ICON    - This widget is used to execute a command which requires initial-
*                       ization of a controlling window.  However, note that it executes
*                       the command in an iconic state.
*
* Note that irregardless of the which command widget is used, the command will be exe-
* cuted without wait.  It will run independently from this client.  This allows the user
* to execute any number of commands from this client.  Note also that all commands are
* executed via the 'system' function call.  Therefore, all features of the Bourne shell
* will be available.
*
* Note that a command which is iconified executes as normal, but does not perform any
* input or output.  This allows icons to be created for background processes which do
* not communicate with the user, but still execute.
*
* Note that a major advantage of this client is that it maintains a history of commands
* in the same way the 'C' shell does.  It also allows users to interactively modify and
* execute previously entered commands.  This compensates for the loss of the history (!)
* function of the 'C' shell.  This assumes of course that the user is not allowed to use
* the 'C' shell.
*
* To exit from this client, the user need simply select the 'exit' command widget.  This
* causes the contents of the command list text widget to be saved to the user's history
* file.  Once this is complete, the client will terminate.
*
*
* DESCRIPTION OF MAIN FUNCTION:
*
* This is the main function of the h_cmd client.  It is responsibile for initialization
```

```
* of the resource database and all widgets which make up the client window.  Once all *
* widgets and their associated callbacks are initialized and realized, this function  *
* calls the Xtoolkit intrinsic (XtMainLoop) to process all incoming events.           *
*                                                                                     *
* The window presented by this client consists of a hierarchy of widgets.  Essentially, *
* it consists of a main form with several child forms, each of which present one major *
* function.  Each child form in turn controls several widgets.  The full hierarchy of *
* widgets is summarized below:                                                        *
*                                                                                     *
*      top -----> form --+--> form (Client  ) --+--> label                            *
*                        |         (ID      )   +--> command (exit client)           *
*                        |                                                           *
*                        +--> form (Command ) --+--> text    (actual command list data) *
*                        |         (List    )                                        *
*                        |                                                           *
*                        +--> form (Execute ) --+--> command (No Win/No Icon)        *
*                                  (Command )   +--> command (Window/No Icon)        *
*                                               +--> command (No Win/Icon   )        *
*                                               +--> command (Window/Icon   )        *
*                                                                                     *
* Each of the forms used is offset from other forms to maintain a consistent layout of *
* information.  The widgets with each form are in turn offset from one another in the *
* same way.  This insures that homogenous widgets remain in close proximity and in a  *
* sensible arrangement.                                                               *
*                                                                                     *
* Once this function calls XtMainLoop, there are a number of callback events which may *
* be executed.  These functions, the command widgets to which they are tied, and the  *
* operations they perform are as follows:                                             *
*                                                                                     *
*      function            event              operation                           *
*      --------            -----              ---------                           *
*      cbr_cmd_terminate   exit               terminate h_cmd client              *
*      cbr_command         execute command    execute a command 1 of 4 ways       *
*                                                                                     *
* For more information on these callback functions, refer to the appropriate source  *
* code file.                                                                          *
*                                                                                     *
*                                                                                     *
* SPECIFICATION DOCUMENTS:                                                            *
*                                                                                     *
*      /hisde/req/requirements                                                        *
*      /hisde/design/design                                                           *
*                                                                                     *
*                                                                                     *
* EXECUTION SEQUENCE:                                                                 *
*                                                                                     *
*      h_cmd                                                                          *
*                                                                                     *
*                                                                                     *
* FILES USED AND APPLICATION DEFINED FORMATS:                                         *
*                                                                                     *
*      Command History File - ~/.history                                              *
*                                                                                     *
*      A command history file is the normal history file maintaine by the UNIX 'C'    *
*      shell.  It is always found in the user's home directory and consistes of an arbi- *
*      trary number of logical lines (strings terminated by newlines).                *
*                                                                                     *
*                                                                                     *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                                 *
*                                                                                     *
*      Mark D. Collier - Software Engineering Section                                 *
*                        Data System Science and Technology Department                *
*                        Automation and Data Systems Division                         *
*                        Southwest Research Institute                                 *
```

```
*****************************************************************************************

#include <stdio.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Cardinals.h>
#include <X11/Shell.h>
#include <Xm/MainW.h>
#include <Xm/RowColumn.h>
#include <h_cmd.bit>
#include <hisde.h>
#include <h_user_inter.h>
#include <h_cmd.h>


/*
 *  Define the strings to contain the command history filename.
 */

char        file        [ SIZE_FILENAME + 1 ];

/*
 *  Declare all widgets which will be used by this client.  Again, this data is made
 *  external to allow simple access in callback function.
 */

Widget          top, widget,
                m_main, mb_main, f_main, mp_file, mp_run,
                f_list, t_list;

/*
 *  Declare all callback functions used by this client.
 */

extern  XtCallbackProc      cbr_cmd_terminate(),
                            cbr_command      ();


main ( argc, argv )
    int     argc;
    char    **argv;
{
/*
 *  Initialize the callback list required for the routine which terminates this client.
 *  This callback occurs when the user selects the "exit" command.
 */

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

/*
 *  Initialize  the callback list required for the function which executes a command.
 *  For each command widget, the appropriate value will be set for the (closure)
 *  member.
 */

    static XtCallbackRec cb_command[] = {
        { (XtCallbackProc)cbr_command, (caddr_t)NULL },
        { (XtCallbackProc)NULL,        (caddr_t)NULL }
    };
```

```
    Arg icon_arg,                   /* Argument used to initialize the graphic icon
                                     * for this client.
                                     */
        args[ 1 ];                  /* Argument list used to initialize widgets.
                                     */

/*
 *  Initialize the Xtoolkit, parse command line, and return the root widget which will be
 *  the parent of the main window.
 */

    top = XtInitialize ( NAME_SHELL, NAME_APLIC, NULL, ZERO, &argc, argv );

/*
 *  If there were arguments on the command line which could not be parsed, call the
 *  function (bad_syntax) to report the error, display the correct syntax, and exit from
 *  the client.
 */

    if ( argc > 1 )
        bad_syntax ( "h_cmd" );

/*
 *  Initialize the icon bitmap for this client.
 */

    XtSetArg ( icon_arg, XtNiconPixmap,
                    XCreateBitmapFromData ( XtDisplay(top), XtScreen(top)->root,
                                            h_cmd_bits, h_cmd_width, h_cmd_height ) );

    XtSetValues ( top, &icon_arg, ONE );

/*
 *  Create the main window widget and the menu bar which will contain all commands.
 */

    m_main = XmCreateMainWindow ( top, "", NULL, 0 );
    XtManageChild ( m_main );

    mb_main = XmCreateMenuBar ( m_main, "", NULL, 0 );
    XtManageChild ( mb_main );

/*
 *  Create pulldown for file commands.
 */

    mp_file = XmCreatePulldownMenu ( mb_main, "", NULL, 0 );
    create_cascade ( "", mb_main, mp_file, LABEL_FILE                );
    create_command ( "", mp_file,          LABEL_EXIT, cb_terminate );

/*
 *  Create pulldown for run commands.
 */

    mp_run  = XmCreatePulldownMenu ( mb_main, "",  NULL, 0 );
    create_cascade ( NULL, mb_main, mp_run, LABEL_RUN );

    cb_command[0].closure = (caddr_t) NOWIN_NOICON;
    create_command ( "", mp_run, LABEL_CMD_1, cb_command );
    cb_command[0].closure = (caddr_t) WIN_NOICON;
    create_command ( "", mp_run, LABEL_CMD_2, cb_command );
    cb_command[0].closure = (caddr_t) NOWIN_ICON;
    create_command ( "", mp_run, LABEL_CMD_3, cb_command );
    cb_command[0].closure = (caddr_t) WIN_ICON;
```

```
    create_command ( "", mp_run, LABEL_CMD_4, cb_command );

/*
 *  Create the help cascade.
 */

    widget = create_cascade ( "", mb_main, NULL, LABEL_HELP );
    XtSetArg ( args[ 0 ], XmNmenuHelpWidget, widget );
    XtSetValues ( mb_main, args, 1 );

/*
 *  Create the form which goes in the main window.
 */

    f_list = create_form ( W_F_LIST_M, m_main );

/*
 *  Initialize the text widget used for the main edit area.
 */

    t_list = create_text  ( W_T_LIST_M, f_list, "", 1, XmMULTI_LINE_EDIT, 1 );

/*
 *  Define the areas which constitute the main window widget.
 */

    XmMainWindowSetAreas ( m_main, mb_main, NULL, NULL, NULL, f_list );

/*
 *  Realize the top level widget.  This causes the main form of this client to be
 *  displayed.
 */


    XtRealizeWidget ( top );

/*
 *  Load in all commands from the user's command history (~/.history) file.  Note
 *  that it is not an error if this file does not as yet exit.
 */

    load_commands ( );

/*
 *  Enter the normal Xtoolkit main loop, which coordinates processing of the various
 *  widget events.  This loop will terminate normally when the user selects 'Exit'
 *  command, which in turn causes the cbr_cmd_terminate callback routine to be
 *  executed.
 */

    XtMainLoop ( );
}
```

```
/*****************************************************************************
* MODULE NAME AND FUNCTION ( save_commands )                                 *
*                                                                            *
* This function is called when the user exits from the client.  It simply saves the new  *
* list of commands to the command history file in the user's home directory.  Note that  *
* this will take place even if a history file does not already exit.         *
*                                                                            *
*                                                                            *
* SPECIFICATION DOCUMENTS:                                                   *
*                                                                            *
*      /hisde/req/requirements                                               *
*      /hisde/design/design                                                  *
*                                                                            *
*                                                                            *
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)      *
*                                                                            *
*      file          (char[]) (I) - String set to the command history filename.  *
*                                                                            *
*      t_list        (Widget) (I)   - Text widget to be updated with loaded commands.  *
*                                                                            *
*                                                                            *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                        *
*                                                                            *
*      Mark D. Collier - Software Engineering Section                        *
*                        Data System Science and Technology Department       *
*                        Automation and Data Systems Division                *
*                        Southwest Research Institute                        *
*****************************************************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <hisde.h>
#include <h_user_inter.h>
#include <h_cmd.h>


extern  char    file[ ];

extern  Widget  t_list;


int save_commands ( )                   /* This function saves all commands in the current
                                         * list of commands to the user's command history
                                         * (~/.history) file.  It will return one of the
                                         * following values:
                                         *
                                         *      ( 0) - Successful operation
                                         *      (-1) - Error occurred.
                                         */
{
    FILE            *fp;                /* File pointer used to open and access the
                                         * user's history file.
                                         */

    register char   *p;                 /* Pointer used to step through the command list
                                         * in order to write it out.
                                         */


/*
 *  Open the command history file.  If this fails, output an warning message to
 *  the system message client.
 */
```

```
    if ( ( fp = fopen ( file, "w" ) ) == NULL ) {
        display_message ( MSG_WARNING, "Could not open the command history file to save" )
;
        return ( -1 );
    }

/*
 *  Write all data to the file.
 */

    p = get_text_widget ( t_list );
    while ( *p )
        putc ( *p++, fp );

/*
 *  Close the history file.  If an error occurs, output an error to the system message
 *  client.
 */

    if ( fclose ( fp ) != 0 ) {
        display_message ( MSG_ERROR, "Could not close the command history file" );
        return ( -1 );
    } else
        return ( 0 );
}
```

```
/*****************************************************************************
* MODULE NAME AND FUNCTION: ( get_home_dir )                                 *
*                                                                            *
* This function is called to return the home directory of the current user.  It examines *
* the /etc/passwd file (via a UNIX function) to get the home directory and then copies   *
* the data into a passed string buffer.                                      *
*                                                                            *
*                                                                            *
* SPECIFICATION DOCUMENTS:                                                   *
*                                                                            *
*      /hisde/req/requirements                                               *
*      /hisde/design/design                                                  *
*                                                                            *
*                                                                            *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                        *
*                                                                            *
*      Mark D. Collier - Software Engineering Section                        *
*                        Data System Science and Technology Department       *
*                        Automation and Data Systems Division                *
*                        Southwest Research Institute                        *
*****************************************************************************/


#include <stdio.h>
#include <pwd.h>
#include <hisde.h>

int get_home_dir ( path )           /* This function provides the user's home directory.
                                     * It returns one of the following values:
                                     *
                                     *      ( 0) - Successful operation
                                     *      (-1) - Error occurred.
                                     */

    char    *path;                  /* Pointer to the string to be updated with the user's
                                     * home directory.
                                     */
{
    struct passwd   *pwd_ptr;       /* Set to point to the /etc/passwd entry for the
                                     * current user.  The home directory is then taken
                                     * from this structure.
                                     */

    extern struct passwd
                    *getpwnam();    /* Function used to get the current users /etc/passwd
                                     * entry.
                                     */

/*
 *  Use the (getpwnam) call to obtain the /etc/passwd entry for the current user.  This
 *    function returns a pointer to a structure containing this data.  If a NULL pointer
 *    is returned, output an error to the system message client and return.
 *  Otherwise (success), copy the user's home directory into the provided parameter and
 *    return.
 */

    if ( ( pwd_ptr = getpwnam( cuserid(NULL) ) ) == NULL ) {
        display_message ( MSG_ERROR, "Could not determine user's home directory" );
        return ( -1 );
    } else {
        strcpy ( path, pwd_ptr->pw_dir );
        return ( 0 );
    }
```

```
/*****************************************************************************
* MODULE NAME AND FUNCTION ( load_commands )                                 *
*                                                                            *
* This function is called to load all commands from the current user's ~/.history file *
* into the external variable (command_list).  This data will later be displayed in the *
* clients main text widget.                                                  *
*                                                                            *
*                                                                            *
* SPECIFICATION DOCUMENTS:                                                   *
*                                                                            *
*      /hisde/req/requirements                                               *
*      /hisde/design/design                                                  *
*                                                                            *
*                                                                            *
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)      *
*                                                                            *
*      file          (char[]) (I/O) - String updated to contain the command history file- *
*                                     name.                                  *
*                                                                            *
*      t_list        (Widget) (I)   - Text widget to be updated with loaded commands. *
*                                                                            *
*                                                                            *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                        *
*                                                                            *
*      Mark D. Collier - Software Engineering Section                        *
*                        Data System Science and Technology Department       *
*                        Automation and Data Systems Division                *
*                        Southwest Research Institute                        *
*****************************************************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <hisde.h>
#include <h_cmd.h>


extern  char    file[ ];

extern  Widget  t_list;


int load_commands ( )                   /* This function loads commands from the user's
                                         * ~/.history file and places them into the external
                                         * variable (command_list).  It will return one of
                                         * the following values:
                                         *
                                         *      ( 0) - Successful operation
                                         *      (-1) - Error occurred.
                                         */
{
    FILE            *fp;                /* File pointer used to open and access the user's
                                         * history file.
                                         */

    register int    i = 0,              /* Pointer used to maintain position in the (string)
                                         * buffer when initializing command list.
                                         */
                    ptr = 0,            /* Pointer used to maintain position in the command
                                         * list string in the text widget as this data is
                                         * being initialized.
                                         */
                    c;                  /* Used to contain last character read (for EOF
                                         * checking).
```

```
                                        */

    char        string[ 101 ];      /* Buffer used to read in the command list data
                                     * (100 bytes at a time).
                                     */

/*
 *  Use the user-defined function (get_home_dir) to update (file) with the name of the
 *  command history file to load.  Next append the name of the standard UNIX command
 *  history filename (with leading slash).
 */

    get_home_dir ( file );
    strcat ( file, FILENAME_HISTORY );

/*
 *  Open the command history file.  If this fails, output an information message to
 *  the system message client.  Note that it is not an error if such a file does not
 *  exist.
 */

    return ( load_text_widget ( file, t_list, 0 ) );
}
```

```
/*****************************************************************************
* MODULE NAME AND FUNCTION ( cbr_cmd_terminate )                             *
*                                                                            *
* This callback function is activated when the user selects the exit command widget.  It *
* is responsible for normal termination of the h_cmd client.  It simply destroys the top *
* level widget, which in turn causes all subordinate widgets to be destroyed.*
*                                                                            *
*                                                                            *
* SPECIFICATION DOCUMENTS:                                                   *
*                                                                            *
*      /hisde/req/requirements                                               *
*      /hisde/design/design                                                  *
*                                                                            *
*                                                                            *
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)      *
*                                                                            *
*      top (Widget) (I) - Pointer to the root widget of the main window.     *
*                                                                            *
*                                                                            *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                        *
*                                                                            *
*      Mark D. Collier - Software Engineering Section                        *
*                        Data System Science and Technology Department       *
*                        Automation and Data Systems Division                *
*                        Southwest Research Institute                        *
*****************************************************************************/


#include <X11/Intrinsic.h>


extern  Widget  top;


XtCallbackProc cbr_cmd_terminate ( 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.
                                     */
            calldata;               /* Specifies any callback-specific data the widget
                                     * needs to pass to the client.  This parameter is
                                     * is not used by this function.
                                     */
{
    XEvent  event;                  /* Event structure needed to make the calls to the
                                     * XtNextEvent and XtDispatchEvent functions.
                                     */

/*
 *  Save all commands to the user's command history file.  This will allow the commands
 *  to be used next time the user logs in.
 */

    save_commands ( );

/*
 *  Destroy the root application shell widget and thereby, all subordinate widgets which
 *  make up the window.
 */
```

```
    XtDestroyWidget ( top );

/*
 *  Determine if any events have been queued.  These will normally be events which
 *  cause the widgets destroy callback to be executed.  Waiting and then processing
 *  the events insures that all data structures initialized by the widgets are
 *  properly deallocated.
 */

    XtNextEvent     ( &event );
    XtDispatchEvent ( &event );

/*
 *  Close the display to deallocate any connections set up by X Windows.  Next
 *  exit from the client.
 */

    XCloseDisplay ( XtDisplay ( top ) );
    exit ( 0 );
}
```

```
#############################################################################
# Makefile for HISDE user interface client (h_info).                        #
#############################################################################

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

TARGET      = h_info

#
# Initialize include and library search paths to include current directory and the
# HISDE directories.  Note that the library path also includes the user interface
# library.
#

BINDIR          = /hisde/bin
INCDIR          = /hisde/src/include
INCDIRS         = -I. -I$(INCDIR)

#
# Define the libraries to search.  This includes the HISDE library, the local user
# interface library, and all required X libraries.
#

LIBRARIES       = -lui -lhisde -lXm -lXt -lX11

#
# Define the compiler and linker flags.
#

CFLAGS          = -O $(INCDIRS)
LDFLAGS         = -O $(EXTRAFLAGS)

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

OBJS        =\
            cbr_info_trm.o\
            cbr_select.o\
            tmr_mon_upd.o\
            h_info.o

#
# Define all header files required.
#

HDRS        =\
            $(INCDIR)/h_info.h\
            $(INCDIR)/h_info.bit\
            $(INCDIR)/h_user_inter.h\
            $(INCDIR)/hisde.h

#
# Make the target.
#

all:        $(TARGET)

$(TARGET):  $(OBJS)
    $(CC) -o $@ $(OBJS) $(LIBRARIES) $(LDFLAGS)
    strip $(TARGET)
    mv $(TARGET) $(BINDIR)
```

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

```
/*****************************************************************************
* MODULE NAME AND FUNCTION: ( h_info )                                       *
*                                                                            *
* This client is used to provide the user with all HISDE-specific information.  It pro-  *
* vides all data which is unique to the HISDE system, including the following items:     *
*                                                                            *
*      Username - the name of the user currently logged into the workstation.            *
*                                                                            *
*      Operation mode - the mode of operation on the current workstation (development,   *
*                       simulation, or flight).                              *
*                                                                            *
*      Classification - the classification mode of the current workstation (non-classi-  *
*                       fied, secret, or classified).                        *
*                                                                            *
*      Current Host - the host system to which unqualified communications requests will  *
*                     be routed.  Note that the user may update this item if desired.    *
*                                                                            *
*      Current Flight - the current flight being accessed and/or controlled by the HISDE *
*                       system.  Note that the user may update this item if desired.     *
*                                                                            *
*      Host list - a list of all host and workstation systems which are accessible from  *
*                  the current workstation.                                  *
*                                                                            *
*      Realtime data sources list - a list of all realtime data sources which are cur-   *
*                                   rently being accessed by processes on this work-     *
*                                   station.                                 *
*                                                                            *
*      Flight list - a list of all flights which are currently active.       *
*                                                                            *
* In addition to viewing this data, the user will be able to easily update the current  *
* host or flight.                                                            *
*                                                                            *
* When this client executes, it will display a main window which presents each of the   *
* data items described above.  For the username, operation mode, and classification, it  *
* is not possible to alter the contents of the item (this is true for the data lists as  *
* well).  The current host and flight fields however may be changed if the user desires. *
* In such a case, the user may simply enter the desired value or 'cut and paste' text    *
* from the appropriate data list field.  When the new text is in place, the user need    *
* simply exit from the field.  At this time, the system data item (and any initialized   *
* monitor window) will be updated.                                           *
*                                                                            *
* The three data list fields may at any given time contain data which is too large to be *
* seen at once on the field.  Therefore, each provides a scrollbar which allows the user *
* to easily page through the data.                                           *
*                                                                            *
*                                                                            *
* DESCRIPTION OF MAIN FUNCTION:                                              *
*                                                                            *
* This is the main function of the h_info client.  It is responsibile for initialization *
* of the resource database and all widgets which make up the main window and the popup   *
* monitor windows.  Once all widgets and their associated callbacks are initialized and  *
* realized, this routine calls the Xtoolkit intrinsic (XtMainLoop) to process all in-    *
* coming events.                                                             *
*                                                                            *
* This routine initializes the application-specific resources (or options) allowed by    *
* this client.  These resources may be set in the server, in an .Xdefaults file, or on   *
* the command line.  For a complete listing of these resources, refer to the main header *
* block.                                                                     *
*                                                                            *
* This routine initializes 6 distinct hierarchies of widgets to present the monitor and  *
* main windows.  The monitor windows each consist of a popup shell widget and a child    *
* label widget.  They are "popped" up and down as requested by the user.  As they are    *
* shell widgets, it is possible to manipulate them in a manner which is independent of   *
* the main window.  This assumes of course that the user has access to a window manager. *
```

```
*
* The main window consists of a complicated hierarchy of widgets.  Essentially, it con-
* sists of a main form with several child forms, each of which present one major piece
* of information.  Each child form in turn controls several widgets, as required by the
* associated data.  The full hierarchy of widgets is summarized below:
*
*      top -----> form --+--> form (Client   ) --+--> label
*                        |         (ID       )   +--> command (exit client)
*                        |
*                        +--> form (Username) --+--> label
*                        |                      +--> command (initialize monitor window)
*                        |                      +--> text    (actual data)
*                        |
*                        +--> form (Operate ) --+--> label
*                        |         (Mode     )   +--> command (initialize monitor window)
*                        |                      +--> text    (actual data)
*                        |
*                        +--> form (Operate ) --+--> label
*                        |         (Classif  )   +--> command (initialize monitor window)
*                        |                      +--> text    (actual data)
*                        |
*                        +--> form (Current ) --+--> label
*                        |         (Host     )   +--> command (initialize monitor window)
*                        |                      +--> text    (actual data)
*                        |
*                        +--> form (Current ) --+--> label
*                        |         (Flight   )   +--> command (initialize monitor window)
*                        |                      +--> text    (actual data)
*                        |
*                        +--> form (Host    ) --+--> label
*                        |         (List     )   +--> text    (data list with scrollbar)
*                        |
*                        +--> form (Realtime) --+--> label
*                        |         (List     )   +--> text    (data list with scrollbar)
*                        |
*                        +--> form (Flight  ) --+--> label
*                                  (List     )   +--> text    (data list with scrollbar)
*
* Each of the forms used is offset from other forms to maintain a consistent layout of
* information.  The widgets with each form are in turn offset from one another in the
* same way.  This insures that homogenous widgets remain in close proximity and in a
* sensible arrangement.
*
* Once this function calls XtMainLoop, there are a number of callback, timer, and action
* events which may be executed.  These functions, the command widgets/timer/action event
* to which they are aried, and the operations they perform are as follows:
*
*      function            event              operation
*      --------            -----              ---------
*      cbr_info_terminate  exit               terminate h_info client
*      cbr_monitor_init    init monitor       pop up or down a monitor window
*      act_monitor_update  leave field        update monitor data for host or flight
*      tmr_monitor_update  timer              update host, rts, or flight list
*
* For more information on these callback, timer, and action functions, refer to the
* appropriate source code file.
*
*
* SPECIFICATION DOCUMENTS:
*
*      /hisde/req/requirements
*      /hisde/design/design
*
*
```

```
* EXECUTION SEQUENCE:                                                          *
*                                                                             *
*      h_info [-interval value]                                               *
*                                                                             *
*      -interval value - optional argument which allows the user to specify the inter-  *
*                        val (in seconds) used to update the host, realtime data source, *
*                        and flight lists.  The value must be in the range of 10 to 300  *
*                        seconds.  If not specified, the default of 30 seconds will be   *
*                        used.                                                *
*                                                                             *
*                                                                             *
* SYSTEM RESOURCES USED:                                                      *
*                                                                             *
* This client indirectly accesses and updates the HISDE shared memory segment via rou-  *
* tines in the HISDE library.                                                 *
*                                                                             *
*                                                                             *
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)       *
*                                                                             *
*      This routine initializes all declared widget variables, the strings which contain *
*      the HISDE information, and the timer value.                            *
*                                                                             *
*                                                                             *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                         *
*                                                                             *
*      Mark D. Collier - Software Engineering Section                         *
*                        Data System Science and Technology Department        *
*                        Automation and Data Systems Division                 *
*                        Southwest Research Institute                         *
*****************************************************************************/


#include <stdio.h>
#include <X11/IntrinsicP.h>
#include <X11/StringDefs.h>
#include <X11/Cardinals.h>
#include <X11/Shell.h>
#include <Xm/MainW.h>
#include <Xm/RowColumn.h>
#include <Xm/Text.h>
#include <Xm/List.h>
#include <hisde.h>
#include <h_user_inter.h>
#include <h_info.h>
#include <h_info.bit>


/*
 *  Declare the variable used to contain the timer value.  It is made external to allow
 *  it to be used in the function which is executed upon completion of the timer.  By
 *  default, it is initialized to 30 seconds, but may be changed by the user.
 */

unsigned int    timer_value = DEFAULT_TIMER_VALUE;

/*
 *  Declare all widgets which will be used by this client.  Again, this data is made
 *  external to allow simple access in callback, timer, and action functions.
 */

Widget          top, m_main, mb_main, mp_file, widget, form,
                t_username,
                t_mode,
                t_class,
```

```
                t_host,
                t_flight,
                t_rts,
                t_lhosts,
                t_lflights,
                t_lrts;

/*
 *  Declare all callback, action, and timer functions used by this client.
 */

extern  XtCallbackProc      cbr_info_terminate(),
                            cbr_select        ();

extern  XtTimerCallbackProc tmr_monitor_update();


main ( argc, argv )
    int     argc;
    char    **argv;
{
/*
 *  Define the application-specific resources allowed by this client.  These values
 *  may be set previously (in server or .Xdefaults) or in the command line.
 */

    static XrmOptionDescRec options[ ] = {
        { "-interval", "Interval",  XrmoptionSepArg, NULL   }
    };

/*
 *  Specify the variables which will be updated if any of the application-specific
 *  resources were specified.  Note that if any of the monitor window resources
 *  are included, the appropriate boolean variable (mon1, mon2, mon3, mon4, or mon5)
 *  will be set to TRUE.  If an update interval is specified, the (timer_value)
 *  variable will be set.
 */

    static Boolean      mon1, mon2, mon3, mon4, mon5;

    static XtResource   resources[ ] = {
        { "interval",  "Interval",  XtRInt,     sizeof(int),     (Cardinal)&timer_value,
          XtRInt,    (caddr_t)&timer_value }
    };

/*
 *  Initialize the callback list required for the routine which terminates this client.
 *  This callback occurs when the user selects the "exit" command.
 */

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

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

/*
 *  Declare all information items which are presented by this client.  This data is
 *  external, as this greatly simplifies its use in callback functions.
 */
```

```
    char    username    [SIZE_USERNAME     + 1 ],
            mode        [SIZE_MODE         + 1 ],
            class       [SIZE_CLASS        + 1 ],
            host        [SIZE_HOSTNAME     + 1 ],
            flight      [SIZE_FLIGHT       + 1 ],
            rts         [SIZE_HOSTNAME     + 1 ],
            flight_list[SIZE_FLIGHT_LIST + 1 ],
            host_list   [SIZE_HOST_LIST    + 1 ],
            rts_list    [SIZE_RTS_LIST     + 1 ];

    Arg     icon_arg,                   /* Define the argument used to initialize the
                                         * graphic icon for this client.
                                         */
            args[ 1 ];

/*
 *  Use HISDE library routines to retrieve the initial values from shared memory.  If
 *  any call returns an error (nonzero), output a message and exit from this client.
 */

    if ( h_get_username          ( username    ) ||
         h_get_mode              ( mode        ) ||
         h_get_class             ( class       ) ||
         h_get_host              ( host        ) ||
         h_get_flight            ( flight      ) ||
         h_get_realtime_host     ( rts         ) ||
         h_list_hosts            ( host_list   ) ||
         h_list_flight           ( flight_list ) ||
         h_list_realtime_hosts   ( rts_list    )    ) {

        h_message ( MSG_ERROR, "Could not retrieve system information values" );
        exit ( 1 );
    }

/*
 *  Initialize the Xtoolkit, parse command line, and return the root widget which will be
 *  the parent of the main window.  Note that this call also parses all application
 *  specific resources.
 */

    top = XtInitialize ( NAME_SHELL, NAME_APLIC, options, XtNumber(options), &argc, argv )
;

/*
 *  If there were arguments on the command line which could not be parsed, call the
 *  function (bad_syntax) to report the error, display the correct syntax, and exit from
 *  the client.
 */

    if ( argc > 1 )
        bad_syntax ( "h_info [-interval value]" );

/*
 *  Initialize the icon bitmap for this client.
 */

    XtSetArg ( icon_arg, XtNiconPixmap,
                    XCreateBitmapFromData ( XtDisplay(top), XtScreen(top)->root,
                                            h_info_bits, h_info_width, h_info_height ) );

    XtSetValues ( top, &icon_arg, ONE );

/*
```

```
     *  Retrieve any application-specific resources which were initialized previously or in
     *    the command line.  This includes both initialization of monitor windows and the
     *    data update interval.
     *  Upon return, check if the user has specified an invalid timer value.  The timer value
     *    used must be in the range of 10 to 300 seconds.  If an invalid value was specified,
     *    output a message and set the value to the default (30 seconds or 30000 milli-
     *    seconds).  Otherwise, multiply the specified value by 1000 to get it into mill-
     *    seconds.
     */

        XtGetApplicationResources( top, (caddr_t)NULL, resources, XtNumber(resources),
                                   NULL, ZERO );

        if ( timer_value < MIN_TIMER_VALUE || timer_value > MAX_TIMER_VALUE ) {
            h_message ( MSG_WARNING, "Invalid timer value specified - Default will be used" );
            timer_value = DEFAULT_TIMER_VALUE * 1000;
        } else
            timer_value = timer_value * 1000;

    /*
     *  Create the main window widget and the menu bar which will contain all commands.
     */

        m_main = XmCreateMainWindow ( top, "", NULL, 0 );
        XtManageChild ( m_main );

        mb_main = XmCreateMenuBar ( m_main, "", NULL, 0 );
        XtManageChild ( mb_main );

    /*
     *  Create pulldown for file commands.
     */

        mp_file = XmCreatePulldownMenu ( mb_main, "", NULL, 0 );
        create_cascade ( "", mb_main, mp_file, LABEL_FILE                );
        create_command ( "", mp_file,          LABEL_EXIT, cb_terminate );

    /*
     *  Create the help cascade.
     */

        widget = create_cascade ( "", mb_main, NULL, LABEL_HELP );
        XtSetArg ( args[ 0 ], XmNmenuHelpWidget, widget );
        XtSetValues ( mb_main, args, 1 );

    /*
     *  Create the form which is used for the main information window.  This form will be
     *  the parent to all widgets except those used for the monitor windows.
     */

        form = create_form ( "", m_main );

    /*
     *  Initialize all single line fields.
     */

                    create_label ( W_L_USER_M, form, LABEL_USERNAME );
       t_username = create_text  ( W_T_USER_M, form, username, 0, XmSINGLE_LINE_EDIT, 0);

                create_label ( W_L_MODE_M, form, LABEL_MODE );
       t_mode = create_text  ( W_T_MODE_M, form, mode, 0, XmSINGLE_LINE_EDIT, 0 );

                 create_label ( W_L_CLASS_M, form, LABEL_CLASS );
       t_class = create_text  ( W_T_CLASS_M, form, class, 0, XmSINGLE_LINE_EDIT, 0 );
```

```
            create_label ( W_L_HOST_M, form, LABEL_HOST  );
    t_host = create_text  ( W_T_HOST_M, form, host, 0, XmSINGLE_LINE_EDIT, 0 );

              create_label ( W_L_FLIGHT_M, form, LABEL_FLIGHT );
    t_flight = create_text  ( W_T_FLIGHT_M, form, flight, 0, XmSINGLE_LINE_EDIT, 0 );

              create_label ( W_L_RTS_M, form, LABEL_RTS );
    t_rts = create_text  ( W_T_RTS_M, form, rts, 0, XmSINGLE_LINE_EDIT, 0 );

/*
 *  Create all list labels.
 */

    create_label ( W_L_LHOSTS_M,   form, LABEL_LHOSTS   );
    create_label ( W_L_LFLIGHTS_M, form, LABEL_LFLIGHTS );
    create_label ( W_L_LRTS_M,     form, LABEL_LRTS     );

/*
 *  Create the scrolled lists. Each has a callback initialized to process selection of
 *  an entry in a list.
 */

    cb_select[ 0 ].closure  = (caddr_t)CB_HOST;
    XtSetArg ( args[ 0 ], XmNbrowseSelectionCallback, cb_select );
    XtManageChild ( t_lhosts  = XmCreateScrolledList ( form, W_S_LHOSTS_M,   args, 1 ) );

    cb_select[ 0 ].closure  = (caddr_t)CB_FLIGHT;
    XtSetArg ( args[ 0 ], XmNbrowseSelectionCallback, cb_select );
    XtManageChild ( t_lflights = XmCreateScrolledList ( form, W_S_LFLIGHTS_M, args, 1 ) );

    cb_select[ 0 ].closure  = (caddr_t)CB_RTS;
    XtSetArg ( args[ 0 ], XmNbrowseSelectionCallback, cb_select );
    XtManageChild ( t_lrts     = XmCreateScrolledList ( form, W_S_LRTS_M,     args, 1 ) );

/*
 *  Initialize each list.
 */

    init_list ( t_lhosts,   host_list   );
    init_list ( t_lflights, flight_list );
    init_list ( t_lrts,     rts_list    );

/*
 *  Initialize the first iteration of the timer.  This will cause the tmr_data_update
 *  callback routine to be executed.  This routine in turn will re-initialize each
 *  timer event, as they are deinitialized once they occur.
 */

    XtAddTimeOut ( timer_value, tmr_monitor_update, NULL );

/*
 *  Realize the top level widget.  This causes the main form of this client to be
 *  displayed.  Note that if the user included the "-iconic" parameter in the command
 *  line, the form will be displayed as an icon.
 */

    XtRealizeWidget ( top );

/*
 *  Enter the normal Xtoolkit main loop, which coordinates processing of the various
 *  widget events.  This loop will terminate normally when the user selects the
 *  "Exit" command, which in turn causes the cbr_info_terminate callback routine to be
 *  executed.
```

```
    */

        XtMainLoop ( );
    }
```

```
/*****************************************************************************
* MODULE NAME AND FUNCTION ( tmr_monitor_update )                            *
*                                                                            *
* This is a callback function which is activated at a defined interval.  By default,  *
* this interval is 30 seconds, but may be set by the user at execution time within the *
* range of 10 to 300 seconds.  The interval is stored in the external (timer_value).  It *
* is never updated once this client is running.                              *
*                                                                            *
* This function is activated in order to update those fields which contain dynamic data. *
* These include:                                                             *
*                                                                            *
*      o Host list           - list of all active hosts.                     *
*      o Flight list         - list of all active flights.                   *
*      o Realtime data list  - list of all active realtime data sources.     *
*                                                                            *
* For more information on these fields and the data they present, refer to the main *
* module header.                                                             *
*                                                                            *
*                                                                            *
* SPECIFICATION DOCUMENTS:                                                   *
*                                                                            *
*      /hisde/req/requirements                                               *
*      /hisde/design/design                                                  *
*                                                                            *
*                                                                            *
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)      *
*                                                                            *
*      timer_value (unsigned) (I)   - The timer value which determines the interval be- *
*                                     tween calls to this function.          *
*                                                                            *
*      t_lhosts    (Widget)   (I)   - Pointer to the text widget containing the list of *
*                                     active hosts.                          *
*                                                                            *
*      t_lrts      (Widget)   (I)   - Pointer to the text widget containing the list of *
*                                     of realtime data sources.              *
*                                                                            *
*      t_lflights  (Widget)   (I)   - Pointer to the text widget containing the list of *
*                                     active flights.                        *
*                                                                            *
*                                                                            *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                        *
*                                                                            *
*      Mark D. Collier - Software Engineering Section                        *
*                        Data System Science and Technology Department       *
*                        Automation and Data Systems Division                *
*                        Southwest Research Institute                        *
*****************************************************************************/


#include <X11/Intrinsic.h>
#include <hisde.h>
#include <h_info.h>


extern  unsigned    timer_value;

extern  Widget      t_lhosts,
                    t_lrts,
                    t_lflights;


XtTimerCallbackProc tmr_monitor_update ( client_data, id )

    caddr_t         client_data;    /* Character data passed to this callback function.
```

```
                                    * It is currently unused by this function.
                                    */

    XtIntervalId    *id;            /* Identifies the timer which caused this function to
                                     * be activated.
                                     */
{
    static char
    host_list_t  [ SIZE_HOST_LIST   + 1 ],
                                    /* Temporary buffer used to get the most recent host
                                     * list.  It is compared to the current list to de-
                                     * termine if it needs to be updated.
                                     */
    flight_list_t[ SIZE_FLIGHT_LIST + 1 ],
                                    /* Temporary buffer used to get the most recent
                                     * flight list.
                                     */
    rts_list_t   [ SIZE_RTS_LIST    + 1 ];
                                    /* Temporary buffer used to get the most recent
                                     * real-time data sources list.
                                     */

/*
 *  Update the list of hosts. Note that in the unlikely event that the h_list_hosts
 *  function fails, output a message and exit.
 *

    if ( h_list_hosts ( host_list_t ) == 0 )
        update_text_widget ( t_lhosts, host_list_t );
    else {
        h_message ( MSG_ERROR, "Could not retrieve the list of current hosts" );
        exit ( 1 );
    }

*
 *  As described for the host list widget, update the list of real-time data sources.
 *

    if ( h_list_realtime_hosts ( rts_list_t ) == 0 )
        update_text_widget ( t_lrts, rts_list_t );
    else {
        h_message ( MSG_ERROR, "Could not retrieve the list of current realtime sources" )
;
        exit ( 1 );
    }

*
 *  As described for the host list widget, update the current flights list widget.
 *

    if ( h_list_flight ( flight_list_t ) == 0 )
        update_text_widget ( t_lflights, flight_list_t );
    else {
        h_message ( MSG_ERROR, "Could not retrieve the list of current flights" );
        exit ( 1 );
    }

*
 *  Reinitialize the timer to cause this function to be executed at the next interval.
 *  It is necessary to perform this each time as the interval is deinitialized after
 *  it completes (indicated by execution of this function).
 */

    XtAddTimeOut ( timer_value, tmr_monitor_update, NULL );
```

```
}
```

```
/*******************************************************************************
* MODULE NAME AND FUNCTION ( cbr_info_terminate )                              *
*                                                                              *
* This callback function is activated when the user selects the exit command widget.  It *
* is responsible for normal termination of the h_info client.  It simply destroys the *
* top level widget, which in turn causes all subordinate widgets (including the popup *
* shells to be destroyed.                                                      *
*                                                                              *
*                                                                              *
* SPECIFICATION DOCUMENTS:                                                     *
*                                                                              *
*      /hisde/req/requirements                                                 *
*      /hisde/design/design                                                    *
*                                                                              *
*                                                                              *
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)        *
*                                                                              *
*      top (Widget) (I) - Pointer to the root widget of the main window.       *
*                                                                              *
*                                                                              *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                          *
*                                                                              *
*      Mark D. Collier - Software Engineering Section                          *
*                        Data System Science and Technology Department         *
*                        Automation and Data Systems Division                  *
*                        Southwest Research Institute                          *
*******************************************************************************/


#include <X11/Intrinsic.h>


extern  Widget  top;


XtCallbackProc cbr_info_terminate ( 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.
                                     */
            calldata;               /* Specifies any callback-specific data the widget
                                     * needs to pass to the client.  This parameter is
                                     * is not used by this function.
                                     */
{
    XEvent  event;                  /* Event structure needed to make the calls to the
                                     * XtNextEvent and XtDispatchEvent functions.
                                     */

/*
 *  Destroy the root application shell widget and thereby, all subordinate widgets which
 *  make up the window and any popup windows used for monitors.
 */

    XtDestroyWidget ( top );

/*
 *  Determine if any events have been queued.  These will normally be events which
 *  cause the widgets destroy callback to be executed.  Waiting and then processing
 *  the events insures that all data structures initialized by the widgets are
```

```
 *  properly deallocated.
 */

    XtNextEvent     ( &event );
    XtDispatchEvent ( &event );

/*
 *  Close the display to deallocate any connections set up by X Windows.  Next
 *  exit from the client.
 */

    XCloseDisplay ( XtDisplay ( top ) );
    exit ( 0 );
}
```

```
/*****************************************************************************
* MODULE NAME AND FUNCTION:  cbr_select                                      *
*                                                                            *
*       This callback is executed when the user selects a string from either the hosts,
*       flights, or rts lists. It will automatically updates the corresponding text
*       widget.                                                              *
*                                                                            *
*                                                                            *
* SPECIFICATION DOCUMENTS:                                                   *
*                                                                            *
*      /hisde/req/requirements                                               *
*      /hisde/design/design                                                  *
*                                                                            *
*                                                                            *
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)      *
*                                                                            *
*      t_host    (Widget) (O) - Text widget for the host field.              *
*                                                                            *
*      t_flight  (Widget) (O) - Text widget for the flight field.            *
*                                                                            *
*      t_rts     (Widget) (O) - Text widget for the realtime data host.      *
*                                                                            *
*                                                                            *
* ORIGINAL AUTHOR AND INDENTIFICATION:                                       *
*                                                                            *
*      Mark D. Collier - Software Engineering Section                        *
*                        Data System Science and Technology Department       *
*                        Automation and Data Systems Division                *
*                        Southwest Research Institute                        *
*****************************************************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/List.h>
#include <hisde.h>
#include <h_info.h>


extern Widget   t_host, t_flight, t_rts;


cbr_select ( widget, closure, calldata )

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

    caddr_t closure,                /* Callback specific data.  This parameter will be
                                     * be set to a value which identifies the selected
                                     * command.
                                     */
            calldata;               /* Specifies any callback-specific data the widget
                                     * needs to pass to the client.  This parameter is
                                     * is not used by this function.
                                     */
{
    XmListCallbackStruct    *ptr;   /* Structure type returned by the (calldata)
                                     * parameter. The selection text will be retrieved
                                     * from it.
                                     */

    char    *p;                     /* Updated to point to the actual text selection.
                                     */
```

```
/*
 *  Set (ptr) to the structure pointer passed in (calldata).
 */

    ptr = (XmListCallbackStruct *)calldata;

/*
 *  Extract the actual string from the compound string in the returned structure. If
 *  this function fails, generate a message and return.
 */

    if ( XmStringGetLtoR ( ptr->item, XmSTRING_DEFAULT_CHARSET, &p ) == FALSE ) {
        display_message ( MSG_ERROR, "Could not convert selection string" );
        return;
    }

/*
 *  Based on which list generated the callback, update the appropriate text widget.
 */

    if (      (int)closure == CB_HOST   )
        update_text_widget ( t_host,   p );
    else if ( (int)closure == CB_FLIGHT )
        update_text_widget ( t_flight, p );
    else if ( (int)closure == CB_RTS    )
        update_text_widget ( t_rts ,   p );
}
```

```
#############################################################################
# Makefile for HISDE user interface client (h_info_a).
#############################################################################

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

TARGET      = h_info_a

#
# Initialize include and library search paths to include current directory and the
# HISDE directories.  Note that the library path also includes the user interface
# library.
#

BINDIR          = /hisde/bin
INCDIR          = /hisde/src/include
INCDIRS         = -I. -I$(INCDIR)

#
# Define the libraries to search.  This includes the HISDE library, the local user
# interface library, and all required X libraries.
#

LIBRARIES       = -lui -lhisde -lXm -lXt -lX11

#
# Define the compiler and linker flags.
#

CFLAGS          = -O $(INCDIRS)
LDFLAGS         = -O $(EXTRAFLAGS)

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

OBJS        =\
            cbr_list_int.o\
            cbr_info_a_t.o\
            h_info_a.o

#
# Define all header files required.
#

HDRS        =\
            $(INCDIR)/h_info_a.h\
            $(INCDIR)/h_info_a.bit\
            $(INCDIR)/h_user_inter.h\
            $(INCDIR)/hisde.h

#
# Make the target.
#

all:        $(TARGET)

$(TARGET):  $(OBJS)
    $(CC) -o $@ $(OBJS) $(LIBRARIES) $(LDFLAGS)
    strip $(TARGET)
    mv $(TARGET) $(BINDIR)
```

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

```
/*****************************************************************************
* MODULE NAME AND FUNCTION: ( h_info_a )                                     *
*                                                                            *
* This client is used to provide the user with all HISDE-specific information.  It is
* functionally identical to the client (h_info), except that it presents its information *
* in a different format.  This format is similar to that presented by the (info) appli- *
* cation in the WEX software system.  As the two clients are functionally equivalent,  *
* the user may use whichever best suits his needs.                           *
*                                                                            *
* This client presents all data which is unique to the HISDE system.  This includes the *
* following items of information.                                            *
*                                                                            *
*      Username - the name of the user currently logged into the workstation.*
*                                                                            *
*      Operation mode - the mode of operation on the current workstation (development,
*                       simulation, or flight).                              *
*                                                                            *
*      Classification - the classification mode of the current workstation (non-classi-
*                       fied, secret, or classified).                        *
*                                                                            *
*      Current Host - the host system to which unqualified communications requests will
*                     be routed.  Note that the user may update this item if desired.
*                                                                            *
*      Current Flight - the current flight being accessed and/or controlled by the HISDE
*                       system.  Note that the user may update this item if desired.
*                                                                            *
*      Host list - a list of all host and workstation systems which are accessible from
*                  the current workstation.                                  *
*                                                                            *
*      Realtime data sources list - a list of all realtime data sources which are cur-
*                                   rently being accessed by processes on this work-
*                                   station.                                 *
*                                                                            *
*      Flight list - a list of all flights which are currently active.       *
*                                                                            *
* Note that the username, the operation mode, the classification, the current host, and *
* the current flight are constantly displayed in the information window.  The remaining *
* three items are lists of information which require more screen space to display.  For *
* this reason, they are accessible as 'popup' windows, which may be displayed and re-  *
* moved as required by the user.  To activate one of these windows, the user need simply *
* select the appropriate command button.  This command button will toggle on and off   *
* display of the window.  Note that it is possible to leave any or all of the lists dis- *
* played on the screen.  They will be updated if any changes occur to the data lists.  *
*                                                                            *
* All list windows will include a scrollbar to allow the user to page through data which *
* is too large to fit at once in the window.                                 *
*                                                                            *
* Note that the user is allowed to update the values in the current host and current   *
* flight fields.  In this case, the user may enter the desired field and simply enter  *
* the new value.  Alternatively, he may 'cut' a host or flight string from the appro-  *
* riate list window and 'paste' it into the text field.  In any case, the value will be *
* checked and updated when the user removes the cursor from the field.  Note that if the *
* user enters an invalid value, he will receive an error message and no update will take *
* place.                                                                     *
*                                                                            *
* To exit from this client, the exit command in the upper right corner may be selected. *
* This causes both the main window and all list windows to be removed.       *
*                                                                            *
*                                                                            *
* DESCRIPTION OF MAIN FUNCTION:                                              *
*                                                                            *
* This is the main function of the h_info_a client.  It is responsibile for initializa- *
* tion of the resource database and all widgets which make up the main window.  Once all *
* widgets and their associated callbacks are initialized and realized, this routine *
```

```
* calls the Xtoolkit intrinsic (XtMainLoop) to process all incoming events.    *
*                                                                              *
* This routine initializes the application-specific resources (or options) allowed by *
* this client.  These resources may be set in the server, in an .Xdefaults file, or on *
* the command line.  For a complete listing of these resources, refer to the main header *
* block.                                                                       *
*                                                                              *
* This routine initializes 4 distinct hierarchies of widgets to present the main and *
* popup windows.  The popup windows each consist of a popup shell widget and a child *
* text widget.  Note that although they appear to be 'popped' up and down, they are act- *
* ually destroyed and recreated.  This insures that they will be placed at the correct *
* screen position in case the user uses the window manager to move them to another part *
* of the screen.                                                               *
*                                                                              *
* The main window consists of a complicated hierarchy of widgets.  Essentially, it con- *
* sists of a main form with several child forms, each of which present one major piece *
* of information.  Each child form in turn controls several widgets, as required by the *
* associated data.  The full hierarchy of widgets is summarized below:         *
*                                                                              *
*      top -----> form --+--> form (Username) --+--> label                     *
*                        |                      +--> text    (actual data)     *
*                        |                                                     *
*                        +--> form (Operate ) --+--> label                     *
*                        |          (Mode    )  +--> text    (actual data)     *
*                        |                                                     *
*                        +--> form (Operate ) --+--> label                     *
*                        |          (Classif )  +--> text    (actual data)     *
*                        |                                                     *
*                        +--> form (Current ) --+--> label                     *
*                        |          (Host    )  +--> text    (actual data)     *
*                        |                      +--> command (initialize list window) *
*                        |                                                     *
*                        +--> form (Current ) --+--> label                     *
*                        |          (Flight  )  +--> text    (actual data)     *
*                        |                      +--> command (initialize list window) *
*                        |                                                     *
*                        +--> form (Realtime) --+--> label                     *
*                        |          (Data    )  +--> command (initialize list window) *
*                        |                                                     *
*                        +--> form (Exit    ) --+--> command (exit client)     *
*                                                                              *
* Note that the realtime data sources does not have a current value (text widget).  This *
* is because there is no current value and the user is not allowed to change the data. *
*                                                                              *
* Each of the forms used is offset from other forms to maintain a consistent layout of *
* information.  The widgets with each form are in turn offset from one another in the *
* same way.  This insures that homogenous widgets remain in close proximity and in a *
* sensible arrangement.                                                        *
*                                                                              *
* Once this function calls XtMainLoop, there are a number of callback, timer, and action *
* events which may be executed.  These functions, the command widgets/timer/action event *
* to which they are aried, and the operations they perform are as follows:     *
*                                                                              *
*      function              event              operation                     *
*      --------              -----              ---------                     *
*      cbr_info_a_terminate  exit               terminate h_info_a client     *
*      cbr_list_init         init popup list    pop up or down a list window  *
*      act_list_update       leave field        update system data for host/flight *
*                                                                              *
* For more information on these callback, timer, and action functions, refer to the *
* appropriate source code file.                                                *
*                                                                              *
*                                                                              *
* SPECIFICATION DOCUMENTS:                                                     *
```

```
*
*       /hisde/req/requirements
*       /hisde/design/design
*
*
* EXECUTION SEQUENCE:
*
*       h_info_a [-interval value]
*
*       -interval value - optional argument which allows the user to specify the inter-
*                         val (in seconds) used to update the host, realtime data source,
*                         and flight lists.  The value must be in the range of 10 to 300
*                         seconds.  If not specified, the default of 30 seconds will be
*                         used.
*
*
* SYSTEM RESOURCES USED:
*
* This client indirectly accesses and updates the HISDE shared memory segment via rou-
* tines in the HISDE library.
*
*
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)
*
*       This routine initializes all declared widget variables and the strings which con-
*       tain the HISDE information.
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*       Mark D. Collier - Software Engineering Section
*                         Data System Science and Technology Department
*                         Automation and Data Systems Division
*                         Southwest Research Institute
*******************************************************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Cardinals.h>
#include <X11/Shell.h>
#include <X11/MwmUtil.h>
#include <Xm/MainW.h>
#include <Xm/RowColumn.h>
#include <Xm/Form.h>
#include <hisde.h>
#include <h_user_inter.h>
#include <h_info_a.h>
#include <h_info_a.bit>


/*
 *  Declare all widgets which will be used by this client.  Again, this data is made
 *  external to allow simple access in callback and action functions.  Note that the
 *  the (popup_shells) and (popup_text) arrays are used to maintain the popup list
 *  windows.
 */

Widget          widget, top, m_main, mb_main, mp_file, form, rc_info,
                t_rts, t_host, t_flight;

/*
 *  Declare all callback, action, and timer functions used by this client.
```

```
 */

extern  XtCallbackProc      cbr_list_init       (),
                            cbr_info_a_terminate();


main ( argc, argv )
    int     argc;
    char    **argv;
{
/*
 *  Initialize the callback list required for the routine which terminates this client.
 *  This callback occurs when the user selects the "exit" command.
 */

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

/*
 *  Initialize the callback list required for the function which initializes the list
 *  windows.  For each command widget, the callback list will be initialized with the
 *  appropriate (closure) parameter.
 */

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

/*
 *  Declare all information items which are presented by this client.  This data is
 *  external, as this greatly simplifies its use in callback, action, and timer functions.
 */

    char        username    [ SIZE_USERNAME    + 1 ],
                mode        [ SIZE_MODE        + 1 ],
                class       [ SIZE_CLASS       + 1 ],
                host        [ SIZE_HOSTNAME    + 1 ],
                flight      [ SIZE_FLIGHT      + 1 ],
                rts         [ SIZE_HOSTNAME    + 1 ];

    Arg icon_arg,                   /* Argument used to initialize the graphic icon
                                     * for this client.
                                     */
        args[ 1 ];                  /* Argument list used to initialize various widget
                                     * resources.
                                     */


/*
 *  Use HISDE library routines to retrieve the initial values from shared memory.  If
 *  any call returns an error (nonzero), output a message and exit from this client.
 */

    if ( h_get_username      ( username    ) ||
         h_get_mode          ( mode        ) ||
         h_get_class         ( class       ) ||
         h_get_host          ( host        ) ||
         h_get_flight        ( flight      ) ||
         h_get_realtime_host ( rts         )    ) {

        h_message ( MSG_ERROR, "Could not retrieve system information values" );
        exit ( 1 );
```

```
    }

/*
 *  Initialize the Xtoolkit, parse command line, and return the root widget which will be
 *  the parent of the main window.  Note that this call also parses all application
 *  specific resources.
 */

    top = XtInitialize ( NAME_SHELL, NAME_APLIC, NULL, 0, &argc, argv );

/*
 *  If there were arguments on the command line which could not be parsed, call the
 *  function (bad_syntax) to report the error, display the correct syntax, and exit from
 *  the client.
 */

    if ( argc > 1 )
        bad_syntax (
            "h_info_a [-interval value] [-username] [-mode] [-class] [-host] [-flight]" );

/*
 *  Initialize the icon bitmap for this client.
 */

    XtSetArg ( icon_arg, XtNiconPixmap,
                    XCreateBitmapFromData ( XtDisplay(top), XtScreen(top)->root,
                                        h_info_a_bits, h_info_a_width, h_info_a_height ) )
;

    XtSetValues ( top, &icon_arg, ONE );

/*
 *  Create the main window widget and the menu bar which will contain all commands.
 */

    m_main = XmCreateMainWindow ( top, "", NULL, 0 );
    XtManageChild ( m_main );

    mb_main = XmCreateMenuBar ( m_main, "", NULL, 0 );
    XtManageChild ( mb_main );

/*
 *  Create pulldown for file commands.
 */

    mp_file = XmCreatePulldownMenu ( mb_main, "", NULL, 0 );
    create_cascade ( "", mb_main, mp_file, LABEL_FILE                    );
    create_command ( "", mp_file,          LABEL_EXIT,     cb_terminate );

/*
 *  Create the help cascade.
 */

    widget = create_cascade ( "", mb_main, NULL, LABEL_HELP );
    XtSetArg ( args[ 0 ], XmNmenuHelpWidget, widget );
    XtSetValues ( mb_main, args, 1 );

/*
 *  Create a RowColumn widget to contain all widgets.
 */

    XtSetArg ( args[ 0 ], XmNorientation, XmHORIZONTAL );
    rc_info = XmCreateRowColumn ( m_main, "", args, 1 );
    XtManageChild ( rc_info );
```

```
/*
 *  Initialize the widgets which will contain the username information.
 */

    create_label ( "", rc_info, LABEL_USERNAME );
    create_text  ( "", rc_info, username, 0, XmSINGLE_LINE_EDIT, 0 );

/*
 *  Initialize the widgets which will contain the mode information.
 */

    create_label ( "", rc_info, LABEL_MODE );
    create_text  ( "", rc_info, mode, 0, XmSINGLE_LINE_EDIT, 0 );

/*
 *  Initialize the widgets which will contain the classification information.
 */

    create_label ( "", rc_info, LABEL_CLASS );
    create_text  ( "", rc_info, class, 0, XmSINGLE_LINE_EDIT, 0 );

/*
 *  Initialize the widgets which will contain the host information.  This includes
 *  a pushbutton with a callbackto the (cbr_list_init) function.
 */

    cb_list_init[ 0 ].closure = (caddr_t)CB_HOST;
            create_command ( "", rc_info, LABEL_HOST, cb_list_init );
    t_host = create_text    ( "", rc_info, host, 0, XmSINGLE_LINE_EDIT, 0 );

/*
 *  Initialize the widgets which will contain the flight information.  This includes
 *  a pushbutton with a callbackto the (cbr_list_init) function.
 */

    cb_list_init[ 0 ].closure = (caddr_t)CB_FLIGHT;
              create_command ( "", rc_info, LABEL_FLIGHT, cb_list_init );
    t_flight = create_text    ( "", rc_info, flight, 0, XmSINGLE_LINE_EDIT, 0 );

/*
 *  Initialize the widgets which will contain the realtime information.  This includes
 *  a pushbutton with a callbackto the (cbr_list_init) function.
 */

    cb_list_init[ 0 ].closure = (caddr_t)CB_RTS;
           create_command ( "", rc_info, LABEL_RTS, cb_list_init );
    t_rts = create_text    ( "", rc_info, "", 0, XmSINGLE_LINE_EDIT, 0 );

/*
 *  Define the areas which constitute the main window widget.
 */

    XmMainWindowSetAreas ( m_main, mb_main, NULL, NULL, NULL, rc_info );

/*
 *  Realize the top level widget.  This causes the main form of this client to be
 *  displayed.  Note that if the user included the "-iconic" parameter in the command
 *  line, the form will be displayed as an icon.
 */

    XtRealizeWidget ( top );

/*
```

```
     *  Enter the normal Xtoolkit main loop, which coordinates processing of
     *  the various widget events.  This loop will terminate normally when the user selects
     *  the "Exit" command, which in turn causes the cbr_info_a_terminate callback routine
     *  to be executed.
     */

        XtMainLoop ( );
    }
```

```
/*****************************************************************************
* MODULE NAME AND FUNCTION ( cbr_info_a_terminate )                          *
*                                                                            *
* This callback function is activated when the user selects the exit command widget.  It *
* is responsible for normal termination of the h_info_a client.  It simply destroys the  *
* top level widget, which in turn causes all subordinate (including the popup shells) to *
* be destroyed.                                                              *
*                                                                            *
*                                                                            *
* SPECIFICATION DOCUMENTS:                                                   *
*                                                                            *
*      /hisde/req/requirements                                               *
*      /hisde/design/design                                                  *
*                                                                            *
*                                                                            *
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)      *
*                                                                            *
*      top (Widget) (I) - Pointer to the root widget of the main window.     *
*                                                                            *
*                                                                            *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                        *
*                                                                            *
*      Mark D. Collier - Software Engineering Section                        *
*                        Data System Science and Technology Department       *
*                        Automation and Data Systems Division                *
*                        Southwest Research Institute                        *
*****************************************************************************/


#include <X11/Intrinsic.h>


extern  Widget  top;


XtCallbackProc cbr_info_a_terminate ( 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.
                                     */
            calldata;               /* Specifies any callback-specific data the widget
                                     * needs to pass to the client.  This parameter is
                                     * is not used by this function.
                                     */
{
    XEvent  event;                  /* Event structure needed to make the calls to the
                                     * XtNextEvent and XtDispatchEvent functions.
                                     */

/*
 *  Destroy the root application shell widget and thereby, all subordinate widgets which
 *  make up the window.
 */

    XtDestroyWidget ( top );

/*
 *  Determine if any events have been queued.  These will normally be events which
 *  cause the widgets destroy callback to be executed.  Waiting and then processing
 *  the events insures that all data structures initialized by the widgets are
```

```
     *  properly deallocated.
     */

        XtNextEvent     ( &event );
        XtDispatchEvent ( &event );

    /*
     *  Close the display to deallocate any connections set up by X Windows.  Next
     *  exit from the client.
     */

        XCloseDisplay ( XtDisplay ( top ) );
        exit ( 0 );
    }
```

```
/*****************************************************************************
* MODULE NAME AND FUNCTION ( cbr_list_init )                                 *
*                                                                            *
* This callback function is activated when the user selects the host, flight, or real-  *
* time data host push buttons in the main window. This function will determine the which *
* button and will display the corresponding list of available entries. This popup will   *
* remain displayed until the user selects an entry or cancels. If an entry is selected,  *
* it will become the current value (and displayed in the main window) if valid.          *
*                                                                            *
*                                                                            *
* SPECIFICATION DOCUMENTS:                                                   *
*                                                                            *
*      /hisde/req/requirements                                               *
*      /hisde/design/design                                                  *
*                                                                            *
*                                                                            *
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)      *
*                                                                            *
*      top       (Widget) (I) - Pointer to the root widget of the main window.  *
*                                                                            *
*      t_host    (Widget) (I) - Pointer to text widget containing the current host.  *
*                                                                            *
*      t_flight  (Widget) (I) - Pointer to text widget containing the current flight.  *
*                                                                            *
*      t_rts     (Widget) (I) - Pointer to text widget containing the current realtime  *
*                               host.                                        *
*                                                                            *
*                                                                            *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                        *
*                                                                            *
*      Mark D. Collier - Software Engineering Section                        *
*                        Data System Science and Technology Department       *
*                        Automation and Data Systems Division                *
*                        Southwest Research Institute                        *
*****************************************************************************/


#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Cardinals.h>
#include <X11/MwmUtil.h>
#include <Xm/SelectioB.h>
#include <hisde.h>
#include <h_user_inter.h>
#include <h_info_a.h>
#include <h_info_a.bit>


/*
 *  Define the required external widget pointers.
 */

extern Widget top, t_host, t_flight, t_rts;


XtCallbackProc cbr_list_init ( widget, closure, calldata )

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

    caddr_t closure,                /* Set to an integer which indicates the callback:
                                     *
                                     *   CB_HOST
```

```
                                    *   CB_FLIGHT
                                    *   CB_RTS
                                    *   CB_OK      (popup)
                                    *   CB_CANCEL  (popup)
                                    */
            calldata;               /* Specifies any callback-specific data the widget
                                     * needs to pass to the client.  This parameter is
                                     * is not used by this function.
                                     */
{
/*
 *  Initialize the callback lists required to return control to this function from the
 *  popup.
 */

    static XtCallbackRec cb_list1[ ] = {
        { (XtCallbackProc)cbr_list_init, (caddr_t)CB_OK },
        { (XtCallbackProc)NULL,          (caddr_t)NULL }
    };

    static XtCallbackRec cb_list2[ ] = {
        { (XtCallbackProc)cbr_list_init, (caddr_t)CB_CANCEL },
        { (XtCallbackProc)NULL,          (caddr_t)NULL      }
    };

    static  Widget  f_popup,        /* Widget used for the main selection popup form.
                                     * It is static to allow use from call to call.
                                     */
                    t_popup;        /* Widget used for the selection text.
                                     */

    static int      save_cmd;       /* Variable used to save the command which initiated
                                     * the popup display.
                                     */

    Arg             args[ 4 ];      /* Argument list used to initialize the popup
                                     * widget resources.
                                     */

    XmString    list[ MAX_HOSTS + 1 ];
                                    /* List of string pointers which will be initialized
                                     * with the selection list items (hosts, flights).
                                     */

    char        data_list[ SIZE_HOST_LIST + 1 ],
                                    /* Buffer used to contain the list as returned from
                                     * shared memory. This is a physical string with
                                     * items separated by newlines. It is used for all
                                     * types of lists, SIZE_HOST_LIST is largest of 3.
                                     */
                temp[ SIZE_HOSTNAME  + 1 ],
                                    /* String used to contain the current entry as parsed
                                     * from the (data_list). This value will be converted
                                     * to an XmString and saved in (list).
                                     */
                *p;                 /* Pointer used to step through the (data_list). Also
                                     * used to point to selection text.
                                     */

    int         cmd,                /* Set to the current value of the (closure) param-
                                     * eter.
                                     */
                count = 0,          /* Used to maintain number of items parsed from the
                                     * (datalist).
```

```
                                        */
                status;                 /* Maintains the status of the (h_set_xxx) call.
                                        *.

/*
 *  Save the (closure) value in a more convienient form.
 */

    cmd = (int)closure;

/*
 *  Determine if called from main window. This requires initialization of the popup.
 *  First save the command so that later, it can be determined what initiated display
 *  of the popup.
 */

    if ( cmd == CB_HOST || cmd == CB_FLIGHT || cmd == CB_RTS ) {
        save_cmd = cmd;

/*
 *      Based on the callback type, retrieve the appropriate data from shared memory.
 */

        if      ( cmd == CB_HOST )
                h_list_hosts          ( data_list );
        else if ( cmd == CB_FLIGHT )
                h_list_flight         ( data_list );
        else if ( cmd == CB_RTS )
                h_list_realtime_hosts ( data_list );

/*
 *      Scan the list and create XmStrings for placement in the selection box. Note that
 *      (data_list) includes a number of logical strings terminated by newlines. The
 *      physical strings is terminated by a newline. Note that the list is terminated by
 *      a NULL entry.
 */

        p = data_list;
        while ( *p ) {
            sscanf ( p, "%s", temp );
            list[ count ] = XmStringCreateLtoR ( temp, XmSTRING_DEFAULT_CHARSET );
            p += strlen ( temp ) + 1;
            count++;
        }
        list[ count ] = NULL;

/*
 *      Initialize the list, the number of items, and the callbacks for the OK and
 *      CANCEL pushbuttons.
 */

        XtSetArg ( args[ 0 ], XmNlistItems,      list     );
        XtSetArg ( args[ 1 ], XmNlistItemCount,  count    );
        XtSetArg ( args[ 2 ], XmNokCallback,     cb_list1 );
        XtSetArg ( args[ 3 ], XmNcancelCallback, cb_list2 );

/*
 *      Create the popup and save the widget pointer for the selection text (it will be
 *      needed later to retrieve the value).
 */

        f_popup = XmCreateSelectionDialog ( top, "", args, 4 );
        t_popup = XmSelectionBoxGetChild ( f_popup, XmDIALOG_TEXT );
```

```
    /*
     *      Set the popup to be application modal. This will disable input to the main
     *      window while the popup is displayed.
     */

            XtSetArg ( args[ 0 ], XmNmwmInputMode, MWM_INPUT_APPLICATION_MODAL );
            XtSetValues ( XtParent ( f_popup ), args, 1 );

            XtManageChild ( f_popup );

    /*
     *  Otherwise, the callback came from the popped up window (OK or CANCEL pushbutton).
     */

        } else {

    /*
     *      Retrieve the text from the selection widget. If the command was OK and if a
     *      string was specified, attempt to update the current host, flight, or realtime
     *      data host as indicated by the initial call to this function (Note that the
     *      set functions return a non-zero value if an illegal value is specified). If
     *      the call is successful, update the value in the main window text widget.
     */

            p = get_text_widget ( t_popup );
            if ( cmd == CB_OK && *p ) {
                if ( save_cmd == CB_HOST ) {
                    if ( ( status = h_set_host          ( p ) ) == 0 )
                        update_text_widget ( t_host,   p );
                } else if ( save_cmd == CB_FLIGHT ) {
                    if ( ( status = h_set_flight        ( p ) ) == 0 )
                        update_text_widget ( t_flight, p );
                } else if ( save_cmd == CB_RTS ) {
                    if ( ( status = h_set_realtime_host ( p ) ) == 0 )
                        update_text_widget ( t_rts,    p );
                }

    /*
     *          If the update failed, output a warning message.
     */

                if ( status )
                    display_message ( MSG_WARNING, "Cannot set to a non-existant value" );
            }

    /*
     *      In both cases (OK and CANCEL) free the memory allocated for the selection and
     *      then destroy the popup widget. Note that the widget is created and destroyed,
     *      as this is the only way it seemed to work.
     */

            XtFree ( p );
            XtDestroyWidget ( f_popup );
        }
    }
```

```
################################################################################
# Makefile for HISDE user interface client h_help.                             #
################################################################################

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

TARGET      = h_help

#
# Initialize include and library search paths to include current directory and the
# HISDE directories.  Note that the library path also includes the user interface
# library.
#

BINDIR          = /hisde/bin
INCDIR          = /hisde/src/include
INCDIRS         = -I. -I$(INCDIR)

#
# Define the libraries to search.  This includes the HISDE library, the local user
# interface library, and all required X libraries.
#

LIBRARIES       = -lui -lhisde -lXm -lXt -lXt -lX11

#
# Define the compiler and linker flags.
#

CFLAGS          = -O $(INCDIRS)
LDFLAGS         = -O $(EXTRAFLAGS)

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

OBJS        =\
            cbr_exit_com.o\
            cbr_help.o\
            h_help.o

#
# Define all header files required.
#

HDRS        =\
            $(INCDIR)/h_help.h\
            $(INCDIR)/h_help.bit\
            $(INCDIR)/hisde.h

#
# Make the target.
#

all:        $(TARGET)

$(TARGET):  $(OBJS)
    $(CC) -o $@ $(OBJS) $(LIBRARIES) $(LDFLAGS)
    strip $(TARGET)
    mv $(TARGET) $(BINDIR)

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

```
/*****************************************************************************
* MODULE NAME AND FUNCTION: ( h_help )
*
*       The h_help client provides the user with the help window for the
*       HISDE system.  It allows the user to view a 'man' page for a selected
*       topic.
*
*       This client displays the help 'man' page in a scroll window which
*       allows the user to view all informatin for the requested topic.
*
*       The user may either specify the desired help topic on the command line
*       with the -topic option, or enter it in the help window when h_help is
*       executed.
*
*
* DESCRIPTION OF MAIN FUNCTION:
*
*       This is the main driver for the h_help client of the HISDE system.  It
*       initializes the X Windows system and then creates the widgets
*       necesssary for the h_help window.   The window created contains
*       a label for the help window, an exit command button, a topic input field,
*       a clear button, and a scroll window for the display of the requested
*       help.  If the user did not specify a help topic on the command line,
*       h_help will prompt the user for a topic in the h_help window.
*
*       When a topic has been received from either the command line or user input,
*       h_help will execute the man function on the topic and pipe the output
*       into a temporary file.  This temporary file will then be displayed in
*       a scroll window.  The user will be allowed to scroll through the information
*       until he requests to exit the h_help client by selecting the exit command
*       button or clears the information by selecting the clear command button.
*
*
* SPECIFICATION DOCUMENTS:
*
*      /hisde/req/requirements
*      /hisde/design/design
*
*
* EXECUTION SEQUENCE:
*
*       h_help [-topic name]
*
*       In addition to the X Windows options which may be used when
*       running h_help, the following options are defined:
*
*       -topic [name] - indicates the topic for which h_help will display
*                       the 'man' page.
*
*
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)
*
*      This function initializes all declared widget variables.
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*      Nancy L. Martin - Software Engineering Section
*                        Data System Science and Technology Department
*                        Automation and Data Systems Division
*                        Southwest Research Institute
*****************************************************************************/
```

```
#include <stdio.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Cardinals.h>
#include <X11/Shell.h>
#include <X11/MwmUtil.h>
#include <Xm/MainW.h>
#include <Xm/Form.h>
#include <Xm/RowColumn.h>
#include <Xm/DialogS.h>
#include <hisde.h>
#include <h_user_inter.h>
#include <h_help.h>
#include <h_help.bit>


/*
 *  Declare all external widgets to be used by the h_help application.
 *  This is required for their use in the callback and action routines.
 */

Widget  top, m_main, mb_main, mp_file, form,
        t_topic,
        f_popup, c_popup, t_popup;

/*
 *  Declare the callback procedures to be executed when a command button is selected.
 */

extern  XtCallbackProc  cbr_help_terminate(),
                        cbr_help          ();

/*
 *  Declare the update function to call if a topic was passed on the command line.
 *  This function will also be called whenever a user leaves the input topic
 *  text widget.
 */

extern  char    *malloc();


main ( argc, argv )
    int     argc;
    char    **argv;
{
/*
 *  Declare the character string to contain the topic name passed in on
 *  the command line, if one is specified.
 */

    static  char    *topic            = NULL,
                    topic_name[ 100 ] = "";

/*
 *  Declare the application-specific resources allowed by this client.  The
 *  resource which may be set is the interval desired for updating the scroll
 *  window.
 */

    static XrmOptionDescRec options[] = {
        {"-topic",  "Topic", XrmoptionSepArg,   NULL  }
    };

    static  XtResource  resources[] = {
```

```
        { "topic", "Topic", XtRString, sizeof(char *), (Cardinal)&topic, NULL,
          (caddr_t)NULL }
    };

/*
 *  Declare the different argument lists.
 */

    Arg     icon_arg,
            args[ 1 ];

/*
 *  Initialize the callback lists required for the clear fields, exit client, and CM
 *  manager commands functions.  These callbacks occur when the user selects one of the
 *  associated command widgets.
 */

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

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

/*
 *  Initialize the X Windows system and create the top level widget for the
 *  help screen.
 */

    top = XtInitialize ( HELP_SHELL,HELP_CLASS, options, XtNumber(options), &argc, argv );

/*
 *  If there were invalid arguments on the command line which could not be parsed,
 *  call the function, bad syntax, to display the correct syntax and exit from
 *  the client.
 */

    if ( argc > 1 )
        bad_syntax ( "h_help [-topic name]" );

/*
 *  Initialize the icon bitmap for this client.
 */

    XtSetArg ( icon_arg, XtNiconPixmap,
               XCreateBitmapFromData ( XtDisplay(top), XtScreen(top) -> root,
               h_help_bits, h_help_width, h_help_height ) );

    XtSetValues ( top, &icon_arg, ONE );

/*
 *  Retrieve any application-specific resources which were initialized previously or
 *  in the command line.  This includes the help topic to be displayed.
 */

    XtGetApplicationResources ( top, (caddr_t)NULL, resources, XtNumber(resources),
                                NULL, ZERO );
    if ( topic )
        strcpy ( topic_name, topic );

/*
```

```
 *  Create the main window widget and the menu bar which will contain all commands.
 */

    m_main = XmCreateMainWindow ( top, "", NULL, 0 );
    XtManageChild ( m_main );

    mb_main = XmCreateMenuBar ( m_main, "", NULL, 0 );
    XtManageChild ( mb_main );

/*
 *  Create pulldown for file commands.
 */

    mp_file = XmCreatePulldownMenu ( mb_main, "", NULL, 0 );
    create_cascade ( "", mb_main, mp_file, LABEL_FILE                    );
    cb_help[ 0 ].closure = (caddr_t)CB_NEW;
    create_command ( "", mp_file,          LABEL_NEW,  cb_help           );
    create_command ( "", mp_file,          LABEL_EXIT, cb_help_terminate );

/*
 *  Create the main form.
 */

    form = create_form ( W_F_TOPIC_M, m_main );

/*
 *  Create the text field for the help display.
 */

    t_topic = create_text ( W_T_TOPIC_M, form, "", 1, XmMULTI_LINE_EDIT, 0 );

/*
 *  Define the areas which constitute the main window widget.
 */

    XmMainWindowSetAreas ( m_main, mb_main, NULL, NULL, NULL, form );

/*
 *  Create the dialog shell used for the popup. Note setting the MODAL flag on the
 *  widget returned by (XmCreateDialogShell) does not work. Therefore we get the
 *  parent of the form and set the value on it.
 */

    f_popup = XmCreateFormDialog ( top, W_F_POPUP_S, NULL, 0 );
    XtSetArg ( args[ 0 ], XmNmwmInputMode, MWM_INPUT_APPLICATION_MODAL );
    XtSetValues ( XtParent ( f_popup ), args, 1 );

/*
 *  Create the label, commands, and text widgets in the popup.
 */

    create_label ( W_L_POPUP_S, f_popup, "Enter Help Topic:" );

    cb_help[ 0 ].closure = (caddr_t)CB_OK;
    c_popup = create_command ( W_C1_POPUP_S, f_popup, LABEL_OK,     cb_help );
    cb_help[ 0 ].closure = (caddr_t)CB_CANCEL;
              create_command ( W_C2_POPUP_S, f_popup, LABEL_CANCEL, cb_help );
    cb_help[ 0 ].closure = (caddr_t)CB_HELP;
              create_command ( W_C3_POPUP_S, f_popup, LABEL_HELP,   cb_help );

    t_popup = create_text ( W_T_POPUP_S, f_popup, topic_name, 0, XmSINGLE_LINE_EDIT, 1 );

/*
 *  Set argument on first command to indicate that it is the default.
```

```
 */

    XtSetArg ( args[ 0 ], XmNshowAsDefault, TRUE );
    XtSetValues ( c_popup, args, 1 );

/*
 *  Call XtRealizeWidget on the top level widget to display the h_help window.
 *  If the user did not specify a topic then only the label, exit button, and
 *  the topic name prompt will be displayed.  Otherwise, the specified help
 *  information will be displayed.
 */

    XtRealizeWidget ( top );

/*
 *  Call get_topic() to display information if a topic was passed on the command
 *  line.
 */

    if ( *topic_name )
        cbr_help ( NULL, (caddr_t)CB_MAIN, NULL );

/*
 *  Enter the Xtoolkit main loop to coordinate processing of all widget events.
 *  If the topic prompt is displayed, then the get_topic function will be called
 *  once the user has entered a topic.
 *
 *  This loop is terminated when the user selects the exit command button and
 *  the associated callback procedure is executed to terminate this client.
 */

    XtMainLoop ( );
}
```

```
/*****************************************************************************
* MODULE NAME AND FUNCTION: cbr_help_terminate                               *
*                                                                            *
*       This function is a callback procedure attached to the exit command button of the *
*       client.  This function causes the client to terminate naturally when the user *
*       selects the exit button.                                             *
*                                                                            *
*                                                                            *
* SPECIFICATION DOCUMENTS:                                                   *
*                                                                            *
*      /hisde/req/requirements                                               *
*      /hisde/design/design                                                  *
*                                                                            *
*                                                                            *
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)      *
*                                                                            *
*       top (Widget) (I) - The top level form widget for the h_help client.  *
*                                                                            *
*                                                                            *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                        *
*                                                                            *
*      Nancy L. Martin - Software Engineering Section                        *
*                        Data System Science and Technology Department       *
*                        Automation and Data Systems Division                *
*                        Southwest Research Institute                        *
*****************************************************************************/


#include <X11/Intrinsic.h>
#include <h_help.h>

/*
 *  Declare the top level widget for the h_help client.
 */

extern  Widget  top;


cbr_help_terminate ( 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.
                                     */
            calldata;               /* Specifies any callback-specific data the widget
                                     * needs to pass to the client.  This parameter is
                                     * is not used by this function.
                                     */

{
/*
 *  Remove the top level widget and then close the h_help display.
 */

    XtUnmapWidget ( top );
    XCloseDisplay ( XtDisplay(top) );

/*
 *  Remove the temporary help file created to display the requested information,
 *  if there was one created.
 */
```

```
    unlink ( HELPFILE );

    exit ( 0 );
}
```

```
/*****************************************************************************
* MODULE NAME AND FUNCTION:  cbr_help                                        *
*                                                                            *
*       This is a callback function which is called if the user did not specify a help   *
*       topic on the command line. This function displays a popup which allows the user  *
*       to enter the topic.                                                  *
*                                                                            *
*       This function will create the command to run 'man' on the entered topic. The     *
*       'sed' stream editor must be used to strip out all underline control characters.  *
*       When the command is executed, a temporary help file containing the stripped      *
*       'man' page will be created and displayed in a scroll window created for this     *
*       purpose.                                                             *
*                                                                            *
*       The command created will be as follows with the input topic in the               *
*       appropriate place:                                                   *
*                                                                            *
*           man [topic name] 2>/dev/null | sed -f /hisde/.filter_man > /hisde/.help_tmp   *
*                                                                            *
*       This command will run 'man' on the entered topic name and pipe it to             *
*       the stream editor, sed.  'sed' will then use the file, /hisde/.filter_man         *
*       to strip out unwanted control sequences and then direct the output to            *
*       the temporary help file, /hisde/.help_tmp.                           *
*                                                                            *
*                                                                            *
* SPECIFICATION DOCUMENTS:                                                   *
*                                                                            *
*      /hisde/req/requirements                                               *
*      /hisde/design/design                                                  *
*                                                                            *
*                                                                            *
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)      *
*                                                                            *
*       t_topic (char) (O) - The widget used to contain the help text.       *
*                                                                            *
*       t_popup (Widget) (I) - The text widget containing the help topic string.  *
*                                                                            *
*       f_popup (Widget) (I) - The form widget containing the popup fields.  *
*                                                                            *
*                                                                            *
* FILES USED AND APPLICATION DEFINED FORMATS:                                *
*                                                                            *
*      /hisde/.filter_man - This file is used with the stream editor, sed, to strip  *
*                           out the escape sequences for underlining in the 'man'    *
*                           pages.  This file contains one line with the following   *
*                           substitute command:                              *
*                                                                            *
*                               s/_^H//g                                     *
*                                                                            *
*                           This command means substitute nothing for an underline   *
*                           followed by a backspace in all lines.            *
*                                                                            *
*                                                                            *
* ORIGINAL AUTHOR AND INDENTIFICATION:                                       *
*                                                                            *
*      Nancy L. Martin - Software Engineering Section                        *
*                        Data System Science and Technology Department       *
*                        Automation and Data Systems Division                *
*                        Southwest Research Institute                        *
*****************************************************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <hisde.h>
```

```
#include <h_user_inter.h>
#include <h_help.h>


/*
 *  Declare the widgets which may be updated by this function.
 */

extern  Widget  t_topic, t_popup, f_popup;

/*
 *  Declare the widget needed to create the file text widget used to
 *  display the topic information.
 */

extern  char    *malloc();


XtCallbackProc cbr_help ( 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.
                                     */
            calldata;               /* Specifies any callback-specific data the widget
                                     * needs to pass to the client.  This parameter is
                                     * is not used by this function.
                                     */
{
    static  int in_popup = FALSE;   /* Static variable used to indicate whether or not a
                                     * popup is displayed.
                                     */

    char    *command,               /* The pointer to the character string which will
                                     * contain command to execute.
                                     */
            *topic_name;            /* Used to contain the help topic entered by the user.
                                     */

/*
 *  If not in popup and if not called from main (called from menu), set popup to
 *  TRUE and display the popup.
 */

    if ( in_popup == FALSE  && (int)closure != CB_MAIN ) {
        in_popup = TRUE;
        XtManageChild ( f_popup );

/*
 *  Otherwise, remove the popup if displayed (it will not be in the call from
 *  the main function.
 */

    } else {
        if ( in_popup ) {
            XtUnmanageChild ( f_popup );
            in_popup = FALSE;
        }

/*
 *      If the command was from main or the OK from popup, get the string from the
```

```
     *      popup text widget. Note that the string is placed in the text widget if
     *      passed on the command line.
     */

          if ( ( (int)closure == CB_MAIN || (int)closure == CB_OK ) &&
               ( topic_name = get_text_widget ( t_popup )          )     )

     /*
      *          Attempt to malloc memory for the help command. If this fails, output a
      *          message.
      */

               if ( (command = malloc(strlen(MAN_FILTER_COMMAND) + strlen(topic_name)
                                      + strlen(HELPFILE))) == NULL )
                   display_message ( MSG_ERROR, "Cannot malloc for command string" );

     /*
      *          Otherwise format and execute the command. When complete free all memory
      *          allocated and update the text widget with the help text. Note that if the
      *          'man' call fails, the message will be displayed in the text widget.
      */

               else {
                   sprintf ( command, MAN_FILTER_COMMAND, topic_name, HELPFILE );
                   system ( command );

                   free   ( command    );
                   XtFree ( topic_name );

                   clear_text_widget ( t_topic );
                   load_text_widget  ( HELPFILE, t_topic, 0 );
               }
          }
     }
```

```
#############################################################################
# Makefile for HISDE user interface client h_login.
#############################################################################

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

TARGET      = h_login

#
# Initialize include and library search paths to include current directory and the
# HISDE directories.  Note that the library path also includes the user interface
# library.
#

BINDIR          = /hisde/bin
INCDIR          = /hisde/src/include
INCDIRS         = -I. -I$(INCDIR)

#
# Define the libraries to search.  This includes the HISDE library, the local user
# interface library, and all required X libraries.
#

LIBRARIES       = -lui -lhisde -lXm -lXt -lX11 $(EXTRALIB)

#
# Define the compiler and linker flags.
#

CFLAGS          = -O $(INCDIRS)
LDFLAGS         = -O $(EXTRAFLAGS)

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

OBJS        =\
            cbr_help_com.o\
            cbr_sel_com.o\
            cbr_ent_com.o\
            cbr_file_com.o\
            h_log_amodes.o\
            h_log_verify.o\
            h_login.o

#
# Define all header files required.
#

HDRS        =\
            $(INCDIR)/h_login.h\
            $(INCDIR)/hisde.h

#
# Make the target.
#

all:        $(TARGET)

$(TARGET):  $(OBJS)
    $(CC) -o $@ $(OBJS) $(LIBRARIES) $(LDFLAGS)
    strip $(TARGET)
```

```
	mv $(TARGET) $(BINDIR)

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

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( h_login )
 *
 *       The h_login client provides the user with the login screen for the HISDE
 *       system.  It allows the user to enter the user name, the password, the
 *       default host, the flight number, the classification, the mode of access
 *       and the name of the initialization file to be used.  It also provides the
 *       user with a command  button for each input which has an associated list
 *       of options - active flight numbers, available hosts/workstations, and
 *       initialization files.  When a command button is selected, a list will
 *       pop-up from which the user may cut and paste his selection into the input
 *       field.  When the command button is selected again, the list will be
 *       popped-down.  If the user enters an invalid selection, a message will
 *       appear in the message window at the bottom of the form, informing the
 *       user that the input is invalid.  This client will also provide a command
 *       button  for 'help' which when selected will give the user information on
 *       how to log into the HISDE system.
 *
 *
 * DESCRIPTION OF MAIN FUNCTION:
 *
 *       This is the main driver for the h_login client of the HISDE system.  It
 *       will initialize the X Windows system and then create the widgets
 *       necesssary for the h_login screen.   The screen which is created contains
 *       a label for the login screen, a command button for help, labels and text
 *       input fields for each piece of information which may be entered by the
 *       user, command buttons for listing valid selections for some of the input
 *       fields, pop-up windows containing the lists of valid selections,
 *       and a message window for the display of information to the user.
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *      /hisde/req/requirements
 *      /hisde/design/design
 *
 *
 *
 * EXECUTION SEQUENCE:
 *
 *      h_login
 *
 *
 * FILES USED AND APPLICATION DEFINED FORMATS:
 *
 *       user-defined initialization file -
 *                   used by this application to initialize the workstation
 *                   for the current user.
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *      Nancy L. Martin - Software Engineering Section
 *                        Data System Science and Technology Department
 *                        Automation and Data Systems Division
 *                        Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Cardinals.h>
#include <X11/Shell.h>
```

```
#include <X11/MwmUtil.h>
#include <Xm/MainW.h>
#include <Xm/RowColumn.h>
#include <Xm/Text.h>
#include <Xm/List.h>
#include <Xm/FileSB.h>
#include <hisde.h>
#include <h_user_inter.h>
#include <h_login.h>


/*
 *  Declare all external widgets to be used by the h_login application.
 *  This is required for their use in the callback and action routines.
 */

Widget  top, m_main, mb_main, mp_file, form, widget, f_popup,
        amode_txt,
        class_txt,
        flight_txt,
        host_txt,
        ifile_txt,
        pass_txt,
        term_txt,
        user_txt,
        sl_modes, sl_flights, sl_hosts;

/*
 *  Declare the callback procedures to be executed when a command button is selected.
 */

extern  XtCallbackProc  enter_command  (),
                        help_command   (),
                        select_command (),
                        file_command   ();


main ( argc, argv )
    int     argc;
    char    **argv;
{
/*
 *  Declare the callback list array to be used when creating command widgets.
 *  This array will contain the routines to be executed when the associated
 *  command button is selected.
 */

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

    static  XtCallbackRec   command_callbacks1[] = {
        { (XtCallbackProc)NULL, (caddr_t)NULL   },
        { (XtCallbackProc)NULL, (caddr_t)NULL   }
    };

/*
 *  Define buffers for display and list data which will be presented in fields.
 */

    char        list_hosts    [ SIZE_HOST_LIST   + 1 ],
                list_flights  [ SIZE_FLIGHT_LIST + 1 ],
                classification[ SIZE_CLASS       + 1 ],
```

```
    list_modes      [ 50                    ];

    Arg args[ 2 ];                      /* Argument list used to initialize various
                                         * widget resources.
                                         */

/*
 *  Initialize the message queue used by the advisory system.
 */

    if ( h_advisory_initialize() != 0 )  {
        fprintf ( stderr, "h_login:  Cannot initialize system resources.\n" );
        exit(-1);
    }

/*
 *  Retreive the current classification mode.
 */

    if ( h_get_class ( classification ) < 0 ) {
        fprintf ( stderr, "h_login:  Cannot get classification.\n" );
        exit(-1);
    }

/*
 *  Initialize the X Windows system and create the top level widget for the login
 *  screen.
 */

    top = XtInitialize ( LOGIN_SHELL, LOGIN_CLASS, NULL, 0, &argc, argv );

/*
 *  Create the main window widget and the menu bar which will contain all commands.
 */

    m_main = XmCreateMainWindow ( top, "", NULL, 0 );
    XtManageChild ( m_main );

    mb_main = XmCreateMenuBar ( m_main, "", NULL, 0 );
    XtManageChild ( mb_main );

/*
 *  Create pulldown for file commands.
 */

    mp_file = XmCreatePulldownMenu ( mb_main, "", NULL, 0 );
    create_cascade ( "", mb_main, mp_file, LABEL_FILE                    );

    command_callbacks[ 0 ].callback = (XtCallbackProc)enter_command;
    create_command ( "", mp_file,          LABEL_LOGIN, command_callbacks );

/*
 *  Create the help cascade.
 */

    widget = create_cascade ( "", mb_main, NULL, LABEL_HELP );
    command_callbacks[ 0 ].callback = (XtCallbackProc)help_command;
    XtAddCallbacks ( widget, XmNactivateCallback, command_callbacks );

    XtSetArg ( args[ 0 ], XmNmenuHelpWidget, widget );
    XtSetValues ( mb_main, args, 1 );

/*
 *  Create the main form.
```

```
 */

    form = create_form ( W_F_LOGIN, m_main );

/*
 *  Create the label which identifies the client. This is unique, as no window manager
 *  is running during login.
 */

    create_label ( W_L_LOGIN, form, LABEL_ID );

/*
 *  Create all labels and text widgets needed for the eight entry fields. The fields
 *  will be arranged (via defaults) in two columns with required on the left and optional
 *  on the right. Note that the initialization file field has a command which causes
 *  display of a file selection widget (via the file_command callback).
 */

                create_label ( W_L_USERNAME,  form, LABEL_USERNAME );
    user_txt  = create_text  ( W_T_USERNAME,  form, "", 0, XmSINGLE_LINE_EDIT, 1 );
    XmAddTabGroup ( user_txt );

                create_label ( W_L_PASSWORD,  form, LABEL_PASSWORD );
    pass_txt  = create_text  ( W_T_PASSWORD,  form, "", 0, XmSINGLE_LINE_EDIT, 1 );
    XmAddTabGroup ( pass_txt );

                create_label ( W_L_MODE,      form, LABEL_MODE     );
    amode_txt = create_text  ( W_T_MODE,      form, "", 0, XmSINGLE_LINE_EDIT, 1 );
    XmAddTabGroup ( amode_txt );

                create_label ( W_L_FLIGHT,    form, LABEL_FLIGHT   );
    flight_txt = create_text ( W_T_FLIGHT,    form, "", 0, XmSINGLE_LINE_EDIT, 1 );
    XmAddTabGroup ( flight_txt );

                create_label ( W_L_CLASS,     form, LABEL_CLASS    );
    class_txt = create_text  ( W_T_CLASS,     form, classification, 0,
                               XmSINGLE_LINE_EDIT, 1 );
    XmAddTabGroup ( class_txt );

                create_label ( W_L_HOST,      form, LABEL_HOST     );
    host_txt  = create_text  ( W_T_HOST,      form, "", 0, XmSINGLE_LINE_EDIT, 1 );
    XmAddTabGroup ( host_txt );

    command_callbacks[ 0 ].callback = (XtCallbackProc)file_command;
    command_callbacks[ 0 ].closure  = (caddr_t)CB_NEW;
                create_command ( W_L_INITFILE,  form, LABEL_FILE, command_callbacks     )
;
    ifile_txt  = create_text  ( W_T_INITFILE,  form, "", 0, XmSINGLE_LINE_EDIT, 1 );
    XmAddTabGroup ( ifile_txt );

                create_label ( W_L_TERMINAL,  form, LABEL_TERMINAL );
    term_txt  = create_text  ( W_T_TERMINAL,  form, "", 0, XmSINGLE_LINE_EDIT, 1 );
    XmAddTabGroup ( term_txt );

/*
 *  Initialize the labels which identify the four scrolled lists.
 */

    create_label ( W_L_LISTMODES,   form, LABEL_LIST_MODES   );
    create_label ( W_L_LISTFLIGHTS, form, LABEL_LIST_FLIGHTS );
    create_label ( W_L_LISTHOSTS,   form, LABEL_LIST_HOSTS   );
    create_label ( W_L_LISTFILES,   form, LABEL_LIST_FILES   );

/*
```

```
    *  Create the scrolled lists. Each has a callback initialized to process selection of
    *  an entry in a list.
    */

       command_callbacks[ 0 ].callback = (XtCallbackProc)select_command;
       command_callbacks[ 0 ].closure  = (caddr_t)CB_SELECT_MODE;
       XtSetArg ( args[ 0 ], XmNbrowseSelectionCallback, command_callbacks );
       XtManageChild ( sl_modes   = XmCreateScrolledList ( form, W_S_LISTMODES,   args, 1 ) )
;

       command_callbacks[ 0 ].callback = (XtCallbackProc)select_command;
       command_callbacks[ 0 ].closure  = (caddr_t)CB_SELECT_FLIGHT;
       XtSetArg ( args[ 0 ], XmNbrowseSelectionCallback, command_callbacks );
       XtManageChild ( sl_flights = XmCreateScrolledList ( form, W_S_LISTFLIGHTS, args, 1 ) )
;

       command_callbacks[ 0 ].callback = (XtCallbackProc)select_command;
       command_callbacks[ 0 ].closure  = (caddr_t)CB_SELECT_HOST;
       XtSetArg ( args[ 0 ], XmNbrowseSelectionCallback, command_callbacks );
       XtManageChild ( sl_hosts   = XmCreateScrolledList ( form, W_S_LISTHOSTS,   args, 1 ) )
;

    /*
     *  Initialize the available modes, flights, and hosts.
     */

       h_list_amodes ( list_modes   );
       h_list_flight ( list_flights );
       h_list_hosts  ( list_hosts   );

    /*
     *  Transfer the data in the buffers to the selection list widgets.
     */

       init_list ( sl_modes,   list_modes   );
       init_list ( sl_flights, list_flights );
       init_list ( sl_hosts,   list_hosts   );

    /*
     *  Create the popup file selection widget. This includes attaching callbacks to the OK
     *  and CANCEL pushbuttons
     */

       command_callbacks [ 0 ].callback = (XtCallbackProc)file_command;
       command_callbacks [ 0 ].closure  = (caddr_t)CB_OK;
       XtSetArg ( args[ 0 ], XmNokCallback,     command_callbacks  );
       command_callbacks1[ 0 ].callback = (XtCallbackProc)file_command;
       command_callbacks1[ 0 ].closure  = (caddr_t)CB_CANCEL;
       XtSetArg ( args[ 1 ], XmNcancelCallback, command_callbacks1 );

       f_popup = XmCreateFileSelectionDialog ( top, "", args, 2 );

       XtSetArg ( args[ 0 ], XmNmwmInputMode, MWM_INPUT_APPLICATION_MODAL );
       XtSetValues ( XtParent ( f_popup ), args, 1 );

    /*
     *  Call XtRealizeWidget on the top level widget to display the h_login screen.
     *  Next, enter the XtMainLoop routine to process events and actions.  This client
     *  will be terminated in a callback routine when the user has successfully entered
     *  all required information.
     */

       XtRealizeWidget ( top );
       XtMainLoop ( );
```

}

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: h_log_amodes                                    *
 *                                                                           *
 *       This function sets up the list of available access modes for the HISDE *
 *       system.  This system has three access modes defined:                *
 *                                                                           *
 *               Development - Used to develop the software for a mission.   *
 *               Simulation  - Used to simulate a mission flight and test the *
 *                             developed software.                           *
 *               Flight      - Used for execution of the approved software during *
 *                             an actual flight.  This mode has a restricted filesystem. *
 *                                                                           *
 *       This list is used in a popup window to give the user the available access modes. *
 *                                                                           *
 *                                                                           *
 * SPECIFICATION DOCUMENTS:                                                  *
 *                                                                           *
 *      /hisde/req/requirements                                              *
 *      /hisde/design/design                                                 *
 *                                                                           *
 *                                                                           *
 * ORIGINAL AUTHOR AND INDENTIFICATION:                                      *
 *                                                                           *
 *      Nancy L. Martin - Software Engineering Section                       *
 *                        Data System Science and Technology Department      *
 *                        Automation and Data Systems Division               *
 *                        Southwest Research Institute                       *
 *****************************************************************************/


#include <stdio.h>
#include <hisde.h>


h_list_amodes ( amodes )

    char    *amodes;                /* A pointer which will be updated with the list of
                                     * access modes.
                                     */
{
/*
 *  Set up the list of access modes.  The constants: DEVELOPMENT, SIMULATION, and
 *  FLIGHT, are located in the hisde.h include file.  Each mode is separated by
 *  a newline for parsing purposes.
 */

    sprintf ( amodes, "%s\n%s\n%s", DEVELOPMENT, SIMULATION, FLIGHT );
}
```

```
/*****************************************************************************
* MODULE NAME AND FUNCTION:  help_command                                    *
*                                                                            *
*       This is a callback function executed whenever the user selects the HELP
*       command button.  Since help is not available for login yet, a message is
*       displayed informing the user of this.                                *
*                                                                            *
*                                                                            *
* SPECIFICATION DOCUMENTS:                                                   *
*                                                                            *
*      /hisde/req/requirements                                               *
*      /hisde/design/design                                                  *
*                                                                            *
*                                                                            *
* ORIGINAL AUTHOR AND INDENTIFICATION:                                       *
*                                                                            *
*      Nancy L. Martin - Software Engineering Section                        *
*                        Data System Science and Technology Department       *
*                        Automation and Data Systems Division                *
*                        Southwest Research Institute                        *
*****************************************************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <hisde.h>


help_command ( widget, closure, calldata )

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

    caddr_t closure,                /* Callback specific data.  This parameter will be
                                     * be set to a value which identifies the selected
                                     * command.
                                     */
            calldata;               /* Specifies any callback-specific data the widget
                                     * needs to pass to the client.  This parameter is
                                     * is not used by this function.
                                     */
{
    display_message ( MSG_WARNING, "There is no help available at this time."  );
}
```

```
/*****************************************************************************************
* MODULE NAME AND FUNCTION:  verify_login_parms                                          *
*                                                                                        *
*      This function is called to verify that all of the user's inputs are valid.       *
*                                                                                        *
*                                                                                        *
* SPECIFICATION DOCUMENTS:                                                               *
*                                                                                        *
*      /hisde/req/requirements                                                           *
*      /hisde/design/design                                                              *
*                                                                                        *
*                                                                                        *
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)                  *
*                                                                                        *
*      user_txt    (Widget) (I) - Text widget for the username field.  Needed to retrieve *
*                                 current value.                                         *
*                                                                                        *
*      pass_txt    (Widget) (I) - Text widget for the password field.                    *
*                                                                                        *
*      amode_txt   (Widget) (I) - Text widget for the mode field.                        *
*                                                                                        *
*      flight_txt  (Widget) (I) - Text widget for the flight field.                      *
*                                                                                        *
*      host_txt    (Widget) (I) - Text widget for the host field.                        *
*                                                                                        *
*      ifile_txt   (Widget) (I) - Text widget for the initialization field.              *
*                                                                                        *
*                                                                                        *
* ORIGINAL AUTHOR AND INDENTIFICATION:                                                   *
*                                                                                        *
*      Nancy L. Martin - Software Engineering Section                                    *
*                        Data System Science and Technology Department                   *
*                        Automation and Data Systems Division                            *
*                        Southwest Research Institute                                    *
*****************************************************************************************/

#include <stdio.h>
#include <X11/Intrinsic.h>
#include <X11/Cardinals.h>
#include <hisde.h>
#include <h_user_inter.h>
#include <h_login.h>


/*
 *  Declare the text widget needed to retrieve the current values.
 */

extern  Widget  user_txt, pass_txt, amode_txt, flight_txt, host_txt, ifile_txt;


verify_login_parms ( )
{
    int     login_err = ZERO;       /*The flag used to indicate that there are fields
                                     * in error and an message needs to be displayed.
                                     */

    char    error_message[MAX_MESSAGE_LENGTH],
                                    /*  The message being built to indicate which
                                     * fields are in error.
                                     */
            *username,              /* Set to the value of the current username.
                                     */
            *mode,                  /* Set to the value of the current mode.
```

```
                                        */
            *string;                   /* Temporary variable used to point to the data
                                        * currently being evaluated.
                                        */

    /*
     *  Initialize the error message string with the lead-in for the list of
     *  invalid fields.
     */

        strcpy ( error_message, "Cannot login.  The following fields are invalid: " );

    /*
     *  Check if the username and password combination is valid.  If it is not,
     *  concatenate USER onto the error message and set the error flag.
     */

        username = get_text_widget ( user_txt );
        string   = get_text_widget ( pass_txt );
        if ( h_set_username ( username, string ) < 0 ) {
            strcat ( error_message, " [ User ]" );
            login_err = ONE;
        }
        XtFree ( string );

    /*
     *  Check for a valid flight number if the selected access mode is either
     *  simulation (S) or flight (F).  If the flight number is invalid,
     *  concatenate FLIGHT onto the error message and set the error flag.
     */

        mode   = get_text_widget ( amode_txt );
        string = get_text_widget ( flight_txt );
        if ( ( *mode == 'S' ) || ( *mode == 'F' ) ) {
            if ( h_set_flight ( string ) < 0 ) {
                strcat ( error_message, " [ Flight ]" );
                login_err = ONE;
            }
        }
        XtFree ( string );

    /*
     *  If a host name was entered, check for it being valid.  If it is invalid,
     *  concatenate HOST onto the error message and set the error flag.
     */

        string = get_text_widget ( host_txt );
        if (*string ) {
            if ( h_set_host ( string ) < 0 ) {
                strcat ( error_message, " [ Host ]" );
                login_err = ONE;
            }
        }
        XtFree ( string );

    /*
     *  Check if the entered access mode is valid.  If it is not, concatenate,
     *  concatenate ACCESS MODE onto the error message and set the error flag.
     */

        if ( h_set_mode ( mode ) < 0 ) {
            strcat ( error_message, " [ Access Mode ]" );
            login_err = ONE;
        }
```

```
    XtFree ( string );

/*
 *  If an initialization file was entered, check that it is valid.  If it is
 *  not, concatenate INITIALIZATION FILE onto the error message and set the
 *  error flag.
 */

    string = get_text_widget ( ifile_txt );
    if ( *string ) {
        if ( h_check_init ( username, string ) < 0 ) {
            strcat ( error_message, " [ Initialization File ]" );
            login_err = ONE;
        }
    }
    XtFree ( string );

/*
 *  If any of the entered fields were in error, call update_status to display
 *  the created error message in the status message window.
 */

    if ( login_err == ONE )
        display_message ( MSG_ERROR, error_message );

/*
 *  Return the error flag to indicate whether the HISDE system may be started
 *  up.
 */

    return ( login_err );
}
```

```
/*****************************************************************************
* MODULE NAME AND FUNCTION:  enter_command                                   *
*                                                                            *
*       This is the callback function which is executed when the ENTER command button *
*       is selected.  This function will verify the user's inputs and then clear the *
*       screen of the login window.  After the login form and popups have been *
*       removed, the HISDE system is started up.  This program spawns a child which *
*       will initiate the user's login and spawn another child process which *
*       overlays the current process with the HISDE startup program.  The parent *
*       process will go into a wait state waiting for the child to die.  When  the *
*       child dies (logout selected by the user), the child processes will die and *
*       the parent process will kill all processes invoked during the login process, *
*       as well as any additional process invoked by the user.  This client will *
*       then exit to insure that the system is completely reinitialized.     *
*                                                                            *
*                                                                            *
* SPECIFICATION DOCUMENTS:                                                   *
*                                                                            *
*      /hisde/req/requirements                                               *
*      /hisde/design/design                                                  *
*                                                                            *
*                                                                            *
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)      *
*                                                                            *
*       top (Widget) (I) - The top level widget for the h_login client.      *
*                                                                            *
* ORIGINAL AUTHOR AND INDENTIFICATION:                                       *
*                                                                            *
*      Nancy L. Martin - Software Engineering Section                        *
*                        Data System Science and Technology Department       *
*                        Automation and Data Systems Division                *
*                        Southwest Research Institute                        *
*****************************************************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <X11/Shell.h>
#include <signal.h>
#include <hisde.h>
#include <h_user_inter.h>
#include <h_login.h>


/*
 *  Declare the top level shell widget.
 */

extern  Widget  top, ifile_txt;


enter_command ( widget, closure, calldata )

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

    caddr_t closure,                /* Callback specific data.  This parameter will be
                                     * be set to a value which identifies the selected
                                     * command.
                                     */
            calldata;               /* Specifies any callback-specific data the widget
                                     * needs to pass to the client.  This parameter is
                                     * is not used by this function.
```

```
                                        */
{
    int     process_id;                 /*  The id of the process spawned by a parent.
                                        */

    char    *file;                      /* Will point to the initialization file entered
                                        * by the user.
                                        */

/*
 *  Save the initialization filename.
 */

    file = get_text_widget ( ifile_txt );

/*
 *  Verify all entered login parameters.  If all parameters are valid, unmap the top
 *  level widget (and all children) and flush the X buffer.
 */

    if ( verify_login_parms ( ) == 0 ) {
        XtUnmapWidget (top);
        XFlush ( XtDisplay ( top ) );

/*
 *      Spawn off a child process to initiate the users login.  The parent process
 *      goes into a wait state waiting for the child to die (which occurs when the
 *      logout client is selected by the user).
 */

        if ( ( process_id = fork() ) == 0 )

/*
 *          In the child process of the login client, the HISDE startup program
 *          should be run as well as the HISDE logout client.
 */

            if ( fork() == 0 ) {

/*
 *              In the grandchild of the HISDE login client, overlay the current
 *              process with the HISDE startup program.  If the user specified an
 *              initialization file during the login process, pass it as a
 *              parameter to the HISDE startup program.
 */

                if ( *file ) {
                    execl ( HISDE_STARTUP, HISDE_STARTUP, file, NULL );
                    exit (-1);
                } else {
                    execl ( HISDE_STARTUP, HISDE_STARTUP, NULL );
                    exit (-1);
                }

/*
 *          Execute the HISDE logout client.  When the client terminates (only
 *          when the user selects the widget), exit the process (which will
 *          then cause the parent process to terminate all siblings.
 */

            }  else {
                system ( EXIT_HISDE );
                exit ( 0 );
            }
```

```
        else
            while ( wait( 0 ) != process_id );

    /*
     *  Send out a SIGKILL to all process associated with the current process group.
     *  The command will kill all processes invoked during the login process plus
     *  any additional processes invoked by the user.
     */

        kill ( 0, SIGKILL );

    /*
     *  Exit the login client, exit is utilized to assure that the system is
     *  completely and accurately reinitialized (this is as opposed to a large
     *  loop in the login client which continually executes).
     */

        exit( 0 );
    }
}
```

```
/*****************************************************************************
* MODULE NAME AND FUNCTION: ifile_command                                    *
*                                                                            *
*       This is a callback function executed when the user selects the command button  *
*       associated with the initialization file input field.  This function will       *
*       check if a list of initialization files is already popped up.  If the list     *
*       is already displayed, it is popped down.  Otherwise, this routine will check   *
*       for a valid user and access mode before calling the function, h_list_directory,*
*       to get the list of initialization files for this user.  When h_list_directory  *
*       returns, the list of files is updated and popped up.                           *
*                                                                                      *
*       If there is not a valid username or access mode, a message will be             *
*       a valid username or access mode, a message will be displayed informing the     *
*       user that no files can be listed until these fields are entered.               *
*                                                                                      *
*                                                                                      *
* SPECIFICATION DOCUMENTS:                                                             *
*                                                                                      *
*      /hisde/req/requirements                                                         *
*      /hisde/design/design                                                            *
*                                                                                      *
*                                                                                      *
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)                *
*                                                                                      *
*      f_popup    (Widget) (I) - File selection widget. Needed to display/remove popup.*
*                                                                                      *
*      user_txt   (Widget) (I) - Text widget for the username field.  Needed to retrieve*
*                                current value.                                        *
*                                                                                      *
*      pass_txt   (Widget) (I) - Text widget for the password field.                   *
*                                                                                      *
*      amode_txt  (Widget) (I) - Text widget for the mode field.                       *
*                                                                                      *
*      ifile_txt  (Widget) (I) - Text widget for the initialization file field.        *
*                                                                                      *
*                                                                                      *
* ORIGINAL AUTHOR AND INDENTIFICATION:                                                 *
*                                                                                      *
*                                                                                      *
*      Nancy L. Martin - Software Engineering Section                                  *
*                        Data System Science and Technology Department                 *
*                        Automation and Data Systems Division                          *
*                        Southwest Research Institute                                  *
*****************************************************************************/


#include <unistd.h>
#include <sys/file.h>
#include <pwd.h>
#include <X11/Intrinsic.h>
#include <X11/Shell.h>
#include <Xm/FileSB.h>
#include <hisde.h>
#include <h_user_inter.h>
#include <h_login.h>


extern Widget   f_popup, user_txt, pass_txt, amode_txt, ifile_txt;


file_command ( widget, closure, calldata )

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

```
                                        */

    caddr_t closure,                    /* Callback specific data.  This parameter will be
                                         * be set to a value which identifies the selected
                                         * command.
                                         */

            calldata;                   /* Specifies any callback-specific data the widget
                                         * needs to pass to the client.  This parameter is
                                         * is not used by this function.
                                         */

{
    XmString    filter;                 /* Will point to compound string formatted from the
                                         * directory.
                                         */

    Arg     args[ 1 ];                  /* Argument list used to set the directory on the
                                         * the file selection widget.
                                         */

    struct passwd   *passwd_ptr;        /* Structure used to contain user information (for
                                         * home directory.
                                         */

    char    temp[ SIZE_FILENAME ],      /* Temporary string used to format the users home
                                         * directory, the location of the startup files, and
                                         * the mask.
                                         */

            *file,                      /* Will point to string entered in the file selec-
                                         * tion text widget.
                                         */

            *mode,                      /* Will point to currently entered mode.
                                         */

            *password,                  /* Will point to currently entered password.
                                         */

            *username;                  /* Will point to currently entered username.
                                         */

/*
 *  If called from the main window, first retrieve the contents of the mode and username
 *  text widgets.
 */

    if ( (int)closure == CB_NEW ) {
        username = get_text_widget ( user_txt  );
        password = get_text_widget ( pass_txt  );
        mode     = get_text_widget ( amode_txt );

/*
 *      Verify that a valid username and password have been entered. If not, return.
 */

        if ( h_set_username ( username, password ) || h_set_mode ( mode ) ) {
            display_message ( MSG_WARNING, "Username/password/mode must be entered" );
            XtFree ( username );
            XtFree ( password );
            XtFree ( mode     );
            return;
        }

/*
 *      Free space allocated for password and mode.
 */

        XtFree ( password );
```

```
        XtFree ( mode     );

/*
 *      Retrieve a pointer to the user's password entry (to get home directory). If this
 *      fails, return.
 */

        if ( ( passwd_ptr = getpwnam ( username ) ) == NULL ) {
            display_message ( MSG_ERROR, "Unable to determine user's home directory" );
            XtFree ( username );
        }

/*
 *      Format the user's home directory (with .hisde subdirectory and mask) and create
 *      a compound string.
 */

        sprintf ( temp, "%s/%s/*", passwd_ptr->pw_dir, HISDE_STARTUP_FILES );
        filter = XmStringLtoRCreate ( temp, XmSTRING_DEFAULT_CHARSET );

/*
 *      Set the file selection widget directory mask argument and display the widget.
 */

        XtSetArg ( args[ 0 ], XmNdirMask, filter );
        XtSetValues ( f_popup, args, 1 );
        XtManageChild ( f_popup );

        XtFree ( username );

/*
 *  Otherwise, the callback originated from the popup. So first remove the popup.
 */

    } else {
        XtUnmanageChild ( f_popup );

/*
 *      If callback is from OK, get the widget for the selection text and use to re-
 *      trieve the selection. If a selection was entered, use it to update the main
 *      text field. Note that if the callback was from CANCEL, no action takes place.
 */

        if ( (int)closure == CB_OK ) {
            widget = XmFileSelectionBoxGetChild ( f_popup, XmDIALOG_TEXT );
            if ( file = get_text_widget ( widget ) )
                update_text_widget ( ifile_txt, file );
            XtFree ( file );
        }
    }
}
```

```
/*****************************************************************************
* MODULE NAME AND FUNCTION:  select_command                                  *
*                                                                            *
*       This callback is executed when the user selects a string from either the modes,  *
*       flights, or hosts lists. It will automatically updates the corresponding text    *
*       widget.                                                              *
*                                                                            *
*                                                                            *
* SPECIFICATION DOCUMENTS:                                                   *
*                                                                            *
*      /hisde/req/requirements                                               *
*      /hisde/design/design                                                  *
*                                                                            *
*                                                                            *
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)      *
*                                                                            *
*      flight_txt (Widget) (O) - Text widget for the flight field. Used to update the  *
*                                current value if one was selected from list.*
*                                                                            *
*      host_txt   (Widget) (O) - Text widget for the host field.             *
*                                                                            *
*      amode_txt  (Widget) (O) - Text widget for the mode field.             *
*                                                                            *
*                                                                            *
* ORIGINAL AUTHOR AND INDENTIFICATION:                                       *
*                                                                            *
*      Mark D. Collier - Software Engineering Section                        *
*                        Data System Science and Technology Department       *
*                        Automation and Data Systems Division                *
*                        Southwest Research Institute                        *
*****************************************************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/List.h>
#include <hisde.h>
#include <h_login.h>


extern Widget   amode_txt,
                flight_txt,
                host_txt;


select_command ( widget, closure, calldata )

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

    caddr_t closure,                /* Callback specific data.  This parameter will be
                                     * be set to a value which identifies the selected
                                     * command.
                                     */

            calldata;               /* Specifies any callback-specific data the widget
                                     * needs to pass to the client.  This parameter is
                                     * is not used by this function.
                                     */

{
    XmListCallbackStruct    *ptr;   /* Structure type returned by the (calldata)
                                     * parameter. The selection text will be retrieved
                                     * from it.
                                     */
```

```
    char    *p;                     /* Updated to point to the actual text selection.
                                     */


/*
 *  Set (ptr) to the structure pointer passed in (calldata).
 */

    ptr = (XmListCallbackStruct *)calldata;

/*
 *  Extract the actual string from the compound string in the returned structure. If
 *  this function fails, generate a message and return.
 */

    if ( XmStringGetLtoR ( ptr->item, XmSTRING_DEFAULT_CHARSET, &p ) == FALSE ) {
        display_message ( MSG_ERROR, "Could not convert selection string" );
        return;
    }

/*
 *  Based on which list generated the callback, update the appropriate text widget.
 */

    if (      (int)closure == CB_SELECT_MODE   )
        update_text_widget ( amode_txt,  p );
    else if ( (int)closure == CB_SELECT_FLIGHT )
        update_text_widget ( flight_txt, p );
    else if ( (int)closure == CB_SELECT_HOST   )
        update_text_widget ( host_txt,   p );
}
```

```
##############################################################################
# Makefile for HISDE user interface client (h_logout).                       #
##############################################################################

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

TARGET      = h_logout

#
# Initialize include and library search paths to include current directory and the
# HISDE directories.  Note that the library path also includes the user interface
# library.
#

BINDIR          = /hisde/bin
INCDIR          = /hisde/src/include
INCDIRS         = -I. -I$(INCDIR)

#
# Define the libraries to search.  This includes the HISDE library, the local user
# interface library, and all required X libraries.
#

LIBRARIES       = -lui -lhisde -lXm -lXt -lX11

#
# Define the compiler and linker flags.
#

CFLAGS          = -O $(INCDIRS)
LDFLAGS         = -O $(EXTRAFLAGS)

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

OBJS        =\
            cbr_log_trm.o\
            h_logout.o

#
# Define all header files required.
#

HDRS        =\
            $(INCDIR)/h_user_inter.h\
            $(INCDIR)/h_logout.h\
            $(INCDIR)/h_logout.bit

#
# Make the target.
#

all:        $(TARGET)

$(TARGET):  $(OBJS)
    $(CC) -o $@ $(OBJS) $(LIBRARIES) $(LDFLAGS)
    strip $(TARGET)
    mv $(TARGET) $(BINDIR)

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

```
/*****************************************************************************
* MODULE NAME AND FUNCTION: ( h_logout )
*
* This client is used to present the means by which the user exits from the HISDE sys-  *
* tem.  This client initializes a single command widget, which when selected, causes all *
* active HISDE clients to be terminated.  At this point, the HISDE login client will    *
* be in control.                                                                        *
*                                                                                       *
*                                                                                       *
* DESCRIPTION OF MAIN FUNCTION:                                                         *
*                                                                                       *
* This is the main function of the h_logout client.  It is responsibile for initializing *
* the resource database and the exit command widget.  Once this widget and its associ-  *
* ated callbacks are initialized and realized, this function calls the Xtoolkit intrin- *
* ic (XtMainLoop) to process all incoming events.                                       *
*                                                                                       *
* The window presented by this client consists of a simple hierarchy of widgets.  These *
* widgets are summarized below:                                                         *
*                                                                                       *
*      top -----> form --+--> command (exit HISDE)                                      *
*                                                                                       *
* Once this function calls XtMainLoop, a callback will occur when the exit command is   *
* selected.  In this case, the (cbr_logout_terminate) function will be called.          *
*                                                                                       *
*                                                                                       *
* SPECIFICATION DOCUMENTS:                                                              *
*                                                                                       *
*      /hisde/req/requirements                                                          *
*      /hisde/design/design                                                             *
*                                                                                       *
*                                                                                       *
* EXECUTION SEQUENCE:                                                                   *
*                                                                                       *
*      h_logout                                                                         *
*                                                                                       *
*                                                                                       *
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)                 *
*                                                                                       *
*      This routine initializes all declared widget variables.                          *
*                                                                                       *
*                                                                                       *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                                   *
*                                                                                       *
*      Mark D. Collier - Software Engineering Section                                   *
*                        Data System Science and Technology Department                  *
*                        Automation and Data Systems Division                           *
*                        Southwest Research Institute                                   *
*****************************************************************************/


#include <stdio.h>
#include <X11/IntrinsicP.h>
#include <X11/StringDefs.h>
#include <X11/Cardinals.h>
#include <X11/Shell.h>
#include <Xm/RowColumn.h>
#include <h_user_inter.h>
#include <h_logout.h>
#include <h_logout.bit>


/*
 *  Declare all widgets which will be used by this client.  This data is made
 *  external to allow simple access in callback functions.
```

```
 */

Widget  top, mb_main, mp_file, widget;

/*
 *  Declare all callback functions.
 */

extern  XtCallbackProc      cbr_logout_terminate();


main ( argc, argv )
    int     argc;
    char    **argv;
{
/*
 *  Initialize each of the callback lists used for the commands generating this type of
 *  event.
 */

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

    Arg icon_arg,                       /* Argument used to initialize the graphic icon
                                         * used for this client.
                                         */

        args[ 1 ];                      /* Argument list used to initialize widgets.
                                         */


/*
 *  Initialize the Xtoolkit, parse command line, and return the root widget which will be
 *  the parent of the window.
 */

    top = XtInitialize ( NAME_SHELL, NAME_APLIC, NULL, ZERO, &argc, argv );

/*
 *  If there were arguments on the command line which could not be parsed, call the
 *  function (bad_syntax) to report the error, display the correct syntax, and exit from
 *  the client.
 */

    if ( argc > 1 )
        bad_syntax ( "h_logout" );

/*
 *  Initialize the icon bitmap for this client.
 */

    XtSetArg ( icon_arg, XtNiconPixmap,
               XCreateBitmapFromData ( XtDisplay(top), XtScreen(top)->root,
                                       h_logout_bits, h_logout_width, h_logout_height ) )

;

    XtSetValues ( top, &icon_arg, ONE );

/*
 *  Create the menu bar which will contain all commands.
 */

    mb_main = XmCreateMenuBar ( top, "", NULL, 0 );
    XtManageChild ( mb_main );
```

```
/*
 *  Create the pulldown for the file commands.
 */

    mp_file = XmCreatePulldownMenu ( mb_main, "" , NULL, 0 );
    create_cascade ( "", mb_main, mp_file, LABEL_FILE                );
    create_command ( "", mp_file,          LABEL_EXIT, cb_terminate );

/*
 *  Create the help cascade.
 */

    widget = create_cascade ( "", mb_main, NULL, LABEL_HELP );
    XtSetArg ( args[ 0 ], XmNmenuHelpWidget, widget );
    XtSetValues ( mb_main, args, 1 );

/*
 *  Realize the top level widget.  This causes the main form of this client to be
 *  displayed.
 */

    XtRealizeWidget ( top );

/*
 *  Enter the normal Xtoolkit main loop, which coordinates processing of the various
 *  widget events.  This loop will terminate normally when the user selects the
 *  "Exit" command.
 */

    XtMainLoop ( );
}
```

```
/*****************************************************************************
* MODULE NAME AND FUNCTION ( cbr_logout_terminate )                          *
*                                                                            *
* This callback function is activated when the user selects the exit command widget.  It *
* is responsible for terminating the HISDE system.  It simply destroys the top level    *
* widget, which in turn causes all subordinate widgets to be removed.        *
*                                                                            *
*                                                                            *
* SPECIFICATION DOCUMENTS:                                                   *
*                                                                            *
*      /hisde/req/requirements                                               *
*      /hisde/design/design                                                  *
*                                                                            *
*                                                                            *
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)      *
*                                                                            *
*      top (Widget) (I) - Pointer to the root widget of the main window.     *
*                                                                            *
*                                                                            *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                        *
*                                                                            *
*      Mark D. Collier - Software Engineering Section                        *
*                        Data System Science and Technology Department       *
*                        Automation and Data Systems Division                *
*                        Southwest Research Institute                        *
*****************************************************************************/


#include <X11/Intrinsic.h>
#include <X11/Shell.h>


extern  Widget  top;


XtCallbackProc cbr_logout_terminate ( 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.
                                     */
            calldata;               /* Specifies any callback-specific data the widget
                                     * needs to pass to the client.  This parameter is
                                     * is not used by this function.
                                     */
{
    XEvent  event;                  /* Event structure needed to make the calls to the
                                     * XtNextEvent and XtDispatchEvent functions.
                                     */

/*
 *  Destroy the root application shell widget and thereby, all subordinate widgets which
 *  make up the window.
 */

    XtDestroyWidget ( top );

/*
 *  Determine if any events have been queued.  These will normally be events which
 *  cause the widgets destroy callback to be executed.  Waiting and then processing
 *  the events insures that all data structures initialized by the widgets are
```

```
 *  properly deallocated.
 */

    XtNextEvent     ( &event );
    XtDispatchEvent ( &event );

/*
 *  Close the display to deallocate any connections set up by X Windows.  Next
 *  exit from the client.
 */

    XCloseDisplay ( XtDisplay ( top ) );
    exit ( 0 );
}
```

```
#############################################################################
# Makefile for HISDE user interface client (h_menu_edit)                    #
#############################################################################

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

TARGET      = h_menu_edit

#
# Initialize include and library search paths to include current directory and the
# HISDE directories.  Note that the library path also includes the user interface
# library.
#

BINDIR          = /hisde/bin
INCDIR          = /hisde/src/include
INCDIRS         = -I. -I$(INCDIR)

#
# Define the libraries to search.  This includes the HISDE library, the local user
# interface library, and all required X libraries.
#

LIBRARIES       = -lui -lhisde -lXm -lXt -lX11

#
# Define the compiler and linker flags.
#

CFLAGS          = -O $(INCDIRS)
LDFLAGS         = -O $(EXTRAFLAGS)

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

OBJS        =\
            save_menu.o\
            cbr_file.o\
            cbr_clear.o\
            cbr_edit_trm.o\
            h_menu_edit.o

#
# Define all header files required.
#

HDRS        =\
            $(INCDIR)/menu.h\
            $(INCDIR)/h_user_inter.h\
            $(INCDIR)/h_menu_edit.bit\
            $(INCDIR)/h_menu_edit.h\
            $(INCDIR)/hisde.h

#
# Make the target.
#

all:        $(TARGET)

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

```
	strip $(TARGET)
	mv $(TARGET) $(BINDIR)

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

```
/*****************************************************************************
* MODULE NAME AND FUNCTION: ( h_menu_edit )                                  *
*                                                                            *
* This HISDE client is provides a convienient means by which the user can build and edit *
* menus.  A menu is a list of labels and commands which is presented to the user for     *
* selection.  A menu may contain both normal commands (programs and options), direct-    *
* ories, and names of sub-menus.  This allows users to build hierarchial lists of menus  *
* which are grouped based on function.  For more information on menus, refer to the      *
* documentation on the (h_menu) client.                                      *
*                                                                            *
* All menus are simply ascii files which contain a logical record (string terminated by  *
* a newline) for each menu item.  Each item in turn consists of the following three      *
* parts:                                                                     *
*                                                                            *
*      Label - The string which appears on the menu item.  It may consist of any string  *
*              which is not blank, does not contain a separator character, and is not    *
*              longer than the maximum length.  It is recommended that only alphanumeric *
*              characters be used.                                           *
*                                                                            *
*      Separator - Separates the label from the command/directory/menu.  This character  *
*                  also indicates whether the next part is a command, directory, or      *
*                  menu.  The character ':' indicates a command, '@' indicates a com-    *
*                  mand which needs a window initialized, '$' indicates a directory, and *
*                  '|' indicates a menu.                                     *
*                                                                            *
*      Command/Menu - The command, directory, or menu which is used.  It may consist of  *
*      Directory      any string which is not blank and is not longer than a maximum      *
*                     size.                                                  *
*                                                                            *
* In addition, the menu file may contain comments and blank lines to aid readability for *
* the user.  Any lines beginning with a '#' character are considered to be comments.  A  *
* menu file may contain any number of lines, but may contain no more than 32 actual menu *
* items.  Note that this value (along with all other constants), may be changed if re-   *
* quired by the user.                                                        *
*                                                                            *
* When this client executes, it will present a window focused on a text widget (with     *
* scrollbar) which allows any menu file to be viewed and edited.  In addition, the user  *
* may select any of the following commands:                                  *
*                                                                            *
*      Exit - Exit from the menu edit client.                                *
*                                                                            *
*      Clear - Clear the menu text widget of all data.                       *
*                                                                            *
*      Load - Load the contents of a menu into the text widget.  Note that this erases   *
*             the current contents of this window.                           *
*                                                                            *
*      Merge - Merge the contents of a menu at the current text widget cursor position.  *
*              Note that the current contents of the window will not be removed.         *
*                                                                            *
*      Save - Save the contents of the menu to a file.  Note that this command will      *
*             verify the contents of the menu.  It will not allow a menu which contains  *
*             invalid data to be saved.                                      *
*                                                                            *
* Note that the load and merge commands do not check the validity of a menu.  This only  *
* occurs when the user attempts to save a file.  Note also the the user could alterna-   *
* tively use 'vi' to edit menus, however, this is discouraged as menu errors will not be *
* found.  The (h_menu) client performs syntax checking, but may still fail if an invalid *
* menu is displayed.                                                         *
*                                                                            *
* Note that in order to use the load, merge, or save commands, the user must first enter *
* a menu filename.  A menu filename is simply the name of the file.  It may be any nor-  *
* mal UNIX path name, which includes either a full pathname or a partial path.  Note     *
* that the filename itself is free format.  The user is free to choose his own naming    *
* standards and extensions.                                                  *
```

```
*
*
* DESCRIPTION OF MAIN FUNCTION:
*
* This is the main function of the h_menu_edit client.  It is responsibile for initial-
* ization of the resource database and all widgets which make up the client window.
* Once all widgets and their associated callbacks are initialized and realized, this
* routine calls the Xtoolkit intrinsic (XtMainLoop) to process all incoming events.
*
* The window presented by this client consists of a hierarchy of widgets.  Essentially,
* it consists of a main form with several child forms, each of which present one major
* function.  Each child form in turn controls several widgets.  The full hierarchy of
* widgets is summarized below:
*
*      top -----> form --+--> form (Client  ) --+--> label
*                        |         (ID      )   +--> command (clear menu)
*                        |                      +--> command (exit client)
*                        |
*                        +--> form (Menu    ) --+--> text    (actual menu data)
*                        |
*                        +--> form (Filename) --+--> label
*                                  (Function)   +--> text    (filename data)
*                                               +--> command (load command)
*                                               +--> command (merge command)
*                                               +--> command (save command)
*
* Each of the forms used is offset from other forms to maintain a consistent layout of
* information.  The widgets with each form are in turn offset from one another in the
* same way.  This insures that homogenous widgets remain in close proximity and in a
* sensible arrangement.
*
* Once this function calls XtMainLoop, there are a number of callback events which may
* be executed.  These functions, the command widgets to which they are tied, and the
* operations they perform are as follows:
*
*      function            event              operation
*      --------            -----              ---------
*      cbr_edit_terminate  exit               terminate h_menu_edit client
*      cbr_clear           clear              clear menu text
*      cbr_load            load               load a menu file
*      cbr_merge           merge              merge a menu file
*      cbr_save            save               save a menu file
*
* For more information on these callback functions, refer to the appropriate source
* code file.
*
*
* SPECIFICATION DOCUMENTS:
*
*      /hisde/req/requirements
*      /hisde/design/design
*
*
* EXECUTION SEQUENCE:
*
*      h_menu_edit [-menu menu_file]
*
*      -menu menu_file - optional parameter which allows the user to specify the initial
*                        menu file to be edited.  If an existing file is specified, it
*                        will be loaded and displayed.
*
*
* FILES USED AND APPLICATION DEFINED FORMATS:
*
```

```
*       Menu File                                                                    *
*                                                                                    *
*       A menu file is a normal UNIX ascii file which contains an arbitrary number of *
*       logical lines.  A logical line may in turn be a blank line, a comment line, or a *
*       menu item line.  A menu item line is defined as one of the following:        *
*                                                                                    *
*            'label'    ':'    'command'                                            *
*                                                                                    *
*                       OR                                                           *
*                                                                                    *
*            'label'    '@'    'command' (to initialize a window)                   *
*                                                                                    *
*                       OR                                                           *
*                                                                                    *
*            'label'    '$'    'directory'                                          *
*                                                                                    *
*                       OR                                                           *
*                                                                                    *
*            'label'    '|'    'menu'                                               *
*                                                                                    *
*       Note that a menu file must not contain more than 32 actual menu items.  It is *
*       possible to increase this value if necessary.                                *
*                                                                                    *
*                                                                                    *
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)              *
*                                                                                    *
*       This routine initializes all declared widget variables.                      *
*                                                                                    *
*                                                                                    *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                                *
*                                                                                    *
*       Mark D. Collier - Software Engineering Section                               *
*                         Data System Science and Technology Department              *
*                         Automation and Data Systems Division                       *
*                         Southwest Research Institute                               *
*************************************************************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Cardinals.h>
#include <X11/Shell.h>
#include <X11/MwmUtil.h>
#include <Xm/MainW.h>
#include <Xm/RowColumn.h>
#include <Xm/Form.h>
#include <Xm/FileSB.h>
#include <h_menu_edit.bit>
#include <hisde.h>
#include <h_user_inter.h>
#include <menu.h>
#include <h_menu_edit.h>


char            file[ SIZE_FILENAME + 1 ] = "";

/*
 *  Declare all widgets which will be used by this client.  Again, this data is made
 *  external to allow simple access in callback functions.
 */

Widget          top,    m_main, mb_main, mp_file, mp_edit, widget,
                f_menu, t_menu,
```

```
                f_file, l_file, t_file,
                f_popup, t_popup;

/*
 *  Declare all callback functions.
 */

extern  XtCallbackProc      cbr_edit_terminate(),
                            cbr_clear         (),
                            cbr_file          ();


main ( argc, argv )
    int     argc;
    char    **argv;
{
/*
 *  Define the application-specific resources allowed by this client.  The only resource
 *  which may be set is the initial menu to be edited.  Note that if the user specified
 *  a menu file, the pointer (fp) will be set to address it.
 */

    static XrmOptionDescRec options[] = {
        { "-menu", "Menu", XrmoptionSepArg, NULL }
    };

    static  char    *fp;

    static XtResource   resources[] = {
        { "menu", "Menu", XtRString, sizeof(char *), (Cardinal) &fp,
          NULL, (caddr_t)NULL }
    };

/*
 *  Initialize each of the callback lists used for the commands generating this type of
 *  event.  These include the exit, clear, and file commands.
 */

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

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

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

    static XtCallbackRec cb_file1[ ] = {
        { (XtCallbackProc)cbr_file,           (caddr_t)NULL },
        { (XtCallbackProc)NULL,               (caddr_t)NULL }
    };

    Arg icon_arg,                   /* Argument used to initialize the graphic icon for
                                     * this client.
                                     */
        args[ 2 ];                  /* Argument list used to initialize various widget
                                     * values.
                                     */
```

```
/*
 *  Initialize the Xtoolkit, parse command line, and return the root widget which will be
 *  the parent of the window.
 */

    top = XtInitialize ( NAME_SHELL, NAME_APLIC, options, XtNumber(options), &argc, argv )
;

/*
 *  If there were arguments on the command line which could not be parsed, call the
 *  function (bad_syntax) to report the error, display the correct syntax, and exit from
 *  the client.
 */

    if ( argc > 1 )
        bad_syntax ( "h_menu_edit [-menu menu_file]" );

/*
 *  Initialize the icon bitmap for this client.
 */

    XtSetArg ( icon_arg, XtNiconPixmap,
                    XCreateBitmapFromData ( XtDisplay(top), XtScreen(top)->root,
                            h_menu_edit_bits, h_menu_edit_width, h_menu_edit_height ) );

    XtSetValues ( top, &icon_arg, ONE );

/*
 *  Parse all application-specific resources.  The only resource which is present is
 *  the initial menu to edit.  If specified, the pointer (fp) will point to the specified
 *  menu file.
 */

    XtGetApplicationResources( top, (caddr_t)NULL, resources, XtNumber(resources),
                               NULL, ZERO );

/*
 *  Create the main window widget and the menu bar which will contain all commands.
 */

    m_main = XmCreateMainWindow ( top, "", NULL, 0 );
    XtManageChild ( m_main );

    mb_main = XmCreateMenuBar ( m_main, "", NULL, 0 );
    XtManageChild ( mb_main );

/*
 *  Create pulldown for file commands.
 */

    mp_file = XmCreatePulldownMenu ( mb_main, "", NULL, 0 );
    create_cascade ( "", mb_main, mp_file, LABEL_FILE                );

    cb_file[ 0 ].closure = (caddr_t)CB_NEW;
    create_command ( "", mp_file,          LABEL_NEW,     cb_file      );
    cb_file[ 0 ].closure = (caddr_t)CB_MERGE;
    create_command ( "", mp_file,          LABEL_MERGE,   cb_file      );
    cb_file[ 0 ].closure = (caddr_t)CB_SAVE;
    create_command ( "", mp_file,          LABEL_SAVE,    cb_file      );
    cb_file[ 0 ].closure = (caddr_t)CB_SAVEAS;
    create_command ( "", mp_file,          LABEL_SAVEAS,  cb_file      );

    create_command ( "", mp_file,          LABEL_EXIT,    cb_terminate );
```

```
/*
 *  Create pulldown for edit commands.
 */

    mp_edit = XmCreatePulldownMenu ( mb_main, "", NULL, 0 );
    create_cascade ( "", mb_main, mp_edit, LABEL_EDIT            );
    create_command ( "", mp_edit,          LABEL_CLEAR, cb_clear );

/*
 *  Create the help cascade.
 */

    widget = create_cascade ( "", mb_main, NULL, LABEL_HELP );
    XtSetArg ( args[ 0 ], XmNmenuHelpWidget, widget );
    XtSetValues ( mb_main, args, 1 );

/*
 *  Initialize the child form which will contain the actual menu viewing/editing area.
 *  Note that the text widget provides a scrollbar and is editable.
 */

    f_menu = create_form ( W_F_MENU_M, m_main );
    t_menu = create_text ( W_T_MENU_M, f_menu, "", 1, XmMULTI_LINE_EDIT, 1 );

/*
 *  Define the areas which constitute the main window widget.
 */

    XmMainWindowSetAreas ( m_main, mb_main, NULL, NULL, NULL, f_menu );

/*
 *  Create the popup file selection widget. This includes attaching callbacks to the OK
 *  and CANCEL pushbuttons
 */

    cb_file [ 0 ].closure = (caddr_t)CB_OK;
    XtSetArg ( args[ 0 ], XmNokCallback,     cb_file );
    cb_file1[ 0 ].closure = (caddr_t)CB_CANCEL;
    XtSetArg ( args[ 1 ], XmNcancelCallback, cb_file1 );
    f_popup = XmCreateFileSelectionDialog ( top, "", args, 2 );

    XtSetArg ( args[ 0 ], XmNmwmInputMode, MWM_INPUT_APPLICATION_MODAL );
    XtSetValues ( XtParent ( f_popup ), args, 1 );

/*
 *  Save the widget pointer for the text widget in the file selection box.
 */

    t_popup = XmFileSelectionBoxGetChild ( f_popup, XmDIALOG_TEXT );


/*
 *  Realize the top level widget.  This causes the main form of this client to be
 *  displayed.
 */

    XtRealizeWidget ( top );

/*
 *  If the user specified a menu file, load it into the text widget.
 */

    if ( fp ) {
```

```
        strcpy ( file, fp );
        cbr_file ( NULL, CB_MAIN, NULL );
    }

/*
 *  Enter the normal Xtoolkit main loop, which coordinates processing of
 *  the various widget events.  This loop will terminate normally when the user selects
 *  the "Exit" command.
 */

    XtMainLoop ( );
}
```

```
/*****************************************************************************
* MODULE NAME AND FUNCTION ( cbr_clear )                                     *
*                                                                            *
* The callback function is executed when the user selects the clear command widget.  It *
* simply clears the text widget which presents the current menu.             *
*                                                                            *
*                                                                            *
* SPECIFICATION DOCUMENTS:                                                   *
*                                                                            *
*      /hisde/req/requirements                                               *
*      /hisde/design/design                                                  *
*                                                                            *
*                                                                            *
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)      *
*                                                                            *
*      t_menu (Widget) (I) - Pointer to the text widget containing the menu. *
*                                                                            *
*                                                                            *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                        *
*                                                                            *
*      Mark D. Collier - Software Engineering Section                        *
*                        Data System Science and Technology Department       *
*                        Automation and Data Systems Division                *
*                        Southwest Research Institute                        *
*****************************************************************************/


#include <X11/Intrinsic.h>


extern  Widget  t_menu;


XtCallbackProc cbr_clear ( 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.
                                     */
            calldata;               /* Specifies any callback-specific data the widget
                                     * needs to pass to the client.  This parameter is
                                     * is not used by this function.
                                     */
{
/*
 *  Clear all text from the menu text widget.
 */

    clear_text_widget ( t_menu );
}
```

```
/*****************************************************************************
* MODULE NAME AND FUNCTION ( cbr_file )                                      *
*                                                                            *
* This callback function is executed when the user selects the load command widget.  It *
* opens the filename specified by the user, reads all contained data, and places it into *
* the menu text widget.  In doing so, any previously displayed menu will be removed.    *
*                                                                            *
*                                                                            *
* SPECIFICATION DOCUMENTS:                                                   *
*                                                                            *
*      /hisde/req/requirements                                               *
*      /hisde/design/design                                                  *
*                                                                            *
*                                                                            *
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)      *
*                                                                            *
*      t_menu ((Widget) (I)   - A pointer to the text widget used to display the menu.  *
*                                                                            *
*      f_popup (Widget) (I)   - A pointer to the form widget used for the popup.        *
*                                                                            *
*      t_popup (Widget) (I)   - A pointer to the text widget used for the popup.        *
*                                                                            *
*      file    (char[]) (I/O) - String containing the current filename.      *
*                                                                            *
*                                                                            *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                        *
*                                                                            *
*      Mark D. Collier - Software Engineering Section                        *
*                        Data System Science and Technology Department       *
*                        Automation and Data Systems Division                *
*                        Southwest Research Institute                        *
*****************************************************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/FileSB.h>
#include <hisde.h>
#include <h_user_inter.h>
#include <h_menu_edit.h>
#include <menu.h>


extern  char    file[ ];

extern  Widget  t_menu,
                f_popup, t_popup;


XtCallbackProc cbr_file ( 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.
                                     */
            calldata;               /* Specifies any callback-specific data the widget
                                     * needs to pass to the client.  This parameter is
                                     * is not used by this function.
                                     */
{
    static int      start_cmd = -1; /* Static value used to contain the command which
```

```
                                * initiated the popup display.
                                */

    register int    pos = -1,   /* Paramter for (load_text_widget). Causes new
                                * data to replace or be merged into the current
                                * menu text widget.
                                */
                    cmd,        /* Set to the command which initiated this callback.
                                */
                    status;     /* Used to save the status of calls made to load and
                                * save menus.
                                */

    char            *file_temp; /* Temporary string which will point to the filename
                                * specified in the popup.
                                */

/*
 *  Convert the (closure) parameter to a normal value to ease comparison.
 */

    cmd = (int)closure;

/*
 *  If the function was called from the main (command line argument), simulate a call
 *  after a normal popup. First save file in the popup text widget and then make it
 *  look like a new call after the popup.
 */

    if ( cmd == CB_MAIN ) {
        update_text_widget ( t_popup, file );
        start_cmd = CB_NEW;
        cmd       = CB_OK;
    }

/*
 *  If a menu command (instead of a popup), save the command and if no file yet
 *  specified or the command requires a new filename, display the popup with the current
 *  filename.
 */

    if ( cmd == CB_NEW || cmd == CB_MERGE || cmd == CB_SAVE || cmd == CB_SAVEAS ) {
        start_cmd = cmd;
        if ( file[ 0 ] == NULL || cmd != CB_SAVE ) {
            update_text_widget ( t_popup, file );
            XtManageChild ( f_popup );
            return;
        }
    }

/*
 *  At this point assume that popup is displayed and a file was entered, so remove
 *  the popup (this is unnessary if called from the main function).
 */

    XtUnmanageChild ( f_popup );

/*
 *  If user selected cancel or help commands on the popup, simply return.
 */

    if ( cmd == CB_CANCEL || cmd == CB_HELP )
        return;
```

```
    /*
     *  Get the text from the widget.
     */

        file_temp = get_text_widget ( t_popup );

    /*
     *  Process the commands. If new or merge, then if merge, get the current position of
     *  the cursor. Next attempt to load the file into the text widget at the appropriate
     *  position (beginning or at cursor position). Note that if (pos) is still -1, the
     *  new data will replace the old data.
     */

        if ( start_cmd == CB_NEW || start_cmd == CB_MERGE ) {
            if ( start_cmd == CB_MERGE )
                pos = get_text_insertion_widget ( t_menu );

            if ( status = load_text_widget ( file_temp, t_menu, pos ) )
                display_message ( MSG_WARNING, "Could not open the specified file" );

    /*
     *  Otherwise (save commands), save the menu to the file. Error messages are generated
     *  internally to this function.
     */

        } else
            status = save_menu ( file_temp, t_menu );

    /*
     *  If the command was not merge (which does not change the filename) and if a new
     *  command did not fail, update the filename with the new name.
     */

        if ( start_cmd != CB_MERGE && ! ( start_cmd == CB_NEW && status ) )
            strcpy ( file, file_temp );

        XtFree ( file_temp );
    }
```

```
/*******************************************************************************
* MODULE NAME AND FUNCTION ( cbr_edit_terminate )                              *
*                                                                              *
* This callback function is activated when the user selects the exit command widget.  It *
* is responsible for normal termination of the h_menu_edit client.  It simply destroys  *
* the top level widget, which in turn causes all subordinate widgets to be destroyed.   *
*                                                                              *
*                                                                              *
* SPECIFICATION DOCUMENTS:                                                     *
*                                                                              *
*      /hisde/req/requirements                                                 *
*      /hisde/design/design                                                    *
*                                                                              *
*                                                                              *
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)        *
*                                                                              *
*      top (Widget) (I) - Pointer to the root widget of the main window.       *
*                                                                              *
*                                                                              *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                          *
*                                                                              *
*      Mark D. Collier - Software Engineering Section                          *
*                        Data System Science and Technology Department         *
*                        Automation and Data Systems Division                  *
*                        Southwest Research Institute                          *
*******************************************************************************/


#include <X11/Intrinsic.h>


extern  Widget  top;


XtCallbackProc cbr_edit_terminate ( 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.
                                     */
            calldata;               /* Specifies any callback-specific data the widget
                                     * needs to pass to the client.  This parameter is
                                     * is not used by this function.
                                     */
{
    XEvent  event;                  /* Event structure needed to make the calls to the
                                     * XtNextEvent and XtDispatchEvent functions.
                                     */

/*
 *  Destroy the root application shell widget and thereby, all subordinate widgets which
 *  make up the window.
 */

    XtDestroyWidget ( top );

/*
 *  Determine if any events have been queued.  These will normally be events which
 *  cause the widgets destroy callback to be executed.  Waiting and then processing
 *  the events insures that all data structures initialized by the widgets are
 *  properly deallocated.
```

```
    */

        XtNextEvent     ( &event );
        XtDispatchEvent ( &event );

    /*
     *  Close the display to deallocate any connections set up by X Windows.  Next
     *  exit from the client.
     */

        XCloseDisplay ( XtDisplay ( top ) );
        exit ( 0 );
    }
```

```
/*****************************************************************************************
* MODULE NAME AND FUNCTION ( save_menu )                                                 *
*                                                                                        *
* The callback function is executed when the user selects the save command widget.  It   *
* verifies the current contents of the menu and if valid, saves them to the specified    *
* filename.  Note that this function will not let the user save a file which contains    *
* any type of invalid data.                                                              *
*                                                                                        *
*                                                                                        *
* SPECIFICATION DOCUMENTS:                                                               *
*                                                                                        *
*      /hisde/req/requirements                                                           *
*      /hisde/design/design                                                              *
*                                                                                        *
*                                                                                        *
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)                  *
*                                                                                        *
*                                                                                        *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                                    *
*                                                                                        *
*      Mark D. Collier - Software Engineering Section                                    *
*                        Data System Science and Technology Department                   *
*                        Automation and Data Systems Division                            *
*                        Southwest Research Institute                                    *
*****************************************************************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <hisde.h>
#include <h_user_inter.h>
#include <menu.h>
#include <string.h>


int save_menu ( file, t_menu )          /* Function returns a value indicating whether or not
                                         * the file could be saved:
                                         *
                                         *   0 - Success
                                         *  -1 - Failure
                                         */
    char            *file;              /* File to save the contents of the menu to.
                                         */

    Widget          t_menu;             /* Text widget containing the menu data.
                                         */
{
    FILE            *fp;                /* File pointer used to open and save the contents
                                         * of the menu.
                                         */

    register int    line  = 0,          /* Used to maintain the current menu line.
                                         */
                    count = 0,          /* Used to maintain the count of actual menu items.
                                         */
                    len;                /* Set to the length of the line which is currently
                                         * being examined.
                                         */

    register char   *p,                 /* Pointer to the separator character in the current
                                         * menu item line.
                                         */
                    *n,                 /* Pointer to the newline (end of line) in the current
                                         * menu item line.
```

```
                                */
                    *s,         /* Pointer to first character in the current menu
                                 * item line.
                                 */
                    *menu;      /* Pointer for menu text return from widget.
                                 */

    char            message[ 80 ];  /* Set to the message which will be output if a menu
                                     * syntax error is found.  It will include the line
                                     * on which the error occurred.
                                     */

/*
 *  Retrieve all text from the widget.
 */

    menu = get_text_widget ( t_menu );

/*
 *  Prepare the verify the format of the menu.  First change the terminating NULL to a
 *  newline to simplify the following processing.
 */

    len = strlen ( menu );
    if ( menu[len-1] != NEWLINE )
        menu[len] = NEWLINE;

/*
 *  Scan the menu to determine if all logical lines are valid.  This loop first breaks
 *  the menu into logical lines (terminated by a newline).  Then, if the line is not
 *  blank or a comment, scans the line for either of the separator characters.
 *
 *  Note that through this loop, (s) points to the start of the line, (n) points to
 *  the newline (end of the line), and (p) points to the separator character (if
 *  present).
 */

    p = menu;
    while ( n = strchr ( p, NEWLINE ) ) {
        line++;
        if ( n > p && *p != COMMENT_CHAR ) {
            count++;
            s = p;
            while ( p<n && *p != SEP_CHAR_CMD  && *p != SEP_CHAR_CMD_W &
                           *p != SEP_CHAR_MENU && *p != SEP_CHAR_DIR )
                p++;

/*
 *          Determine if the current menu line is invalid.  Errors include omitting a
 *          separator, omitting the label, omitting the command/menu, entering a label
 *          which is too large, or entering a command/menu which is too large.  In any
 *          of these cases, format a message (which includes the line number), output
 *          to the system message client, and return.
 */

            if ( p == n ) {
                sprintf ( message, "No separator character found on line %d", line );
                XtFree ( menu );
                return ( display_message ( MSG_WARNING, message ) );
            } else if ( s == p ) {
                sprintf ( message, "No label found on line %d", line );
                XtFree ( menu );
                return ( display_message ( MSG_WARNING, message ) );
            } else if ( p+1 == n ) {
```

```
                sprintf ( message, "No command found on line %d", line );
                XtFree ( menu );
                return ( display_message ( MSG_WARNING, message ) );
            } else if ( s+SIZE_MENU_LABEL-1 < p ) {
                sprintf ( message, "Label is too large on line %d", line );
                XtFree ( menu );
                return ( display_message ( MSG_WARNING, message ) );
            } else if ( p+SIZE_MENU_DATA < n ) {
                sprintf ( message, "Command is too large on line %d", line );
                XtFree ( menu );
                return ( display_message ( MSG_WARNING, message ) );
            }
        }

/*
 *  Set (p) to point to the next line (first character following the newline).
 */

        p = n + 1;
    } /* of while */

/*
 *  Set the last character in the menu buffer to NULL.  This is in case it was changed
 *  to a newline to simplify the error checking.
 */

    menu[ len ] = NULL;

/*
 *  Determine if the menu has too many actual items (not counting blank lines and
 *  comments). If so, format a message, output to the system message client, and return.
 */

    if ( count > NUM_MENU_ITEMS ) {
        sprintf ( message, "A menu must not have more than %d items", NUM_MENU_ITEMS );
        XtFree ( menu );
        return ( display_message ( MSG_WARNING, message ) );
    }

/*
 *  Open the file for write access.  If an error occurs, output a message to the
 *  system message client and return.
 */

    if ( ( fp = fopen ( file, "w" ) ) == NULL ) {
        XtFree ( menu );
        return ( display_message ( MSG_WARNING, "Could not open the specified file" ) );
    }

/*
 *  Write the contents of the menu to the file. When complete, free memory allocated for
 *  widget text.
 */

    p = menu;
    while ( *p )
        putc ( *p++, fp );

    XtFree ( menu );

/*
 *  Close the file.  If an error occurs while closing the file, output a message to the
 *  system message client.  Otherwise, inform the user that the file was successfully
 *  saved.
```

```
    */

        if ( fclose ( fp ) )
            return ( display_message ( MSG_ERROR, "Could not properly close the menu file" ) )
    ;

        display_message ( MSG_INFORMATION, "Menu file was successfully saved" );
        return ( 0 );
    }
```

```
##############################################################################
# Makefile for HISDE user interface client h_advisory.                       #
##############################################################################

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

TARGET      = h_msg_look

#
# Initialize include and library search paths to include current directory and the
# HISDE directories.  Note that the library path also includes the user interface
# library.
#

BINDIR          = /hisde/bin
INCDIR          = /hisde/src/include
INCDIRS         = -I. -I$(INCDIR)

#
# Define the libraries to search.  This includes the HISDE library, the local user
# interface library, and all required X libraries.
#

LIBRARIES       = -lui -lhisde -lXm -lXt -lX11

#
# Define the compiler and linker flags.
#

CFLAGS          = -O $(INCDIRS)
LDFLAGS         = -O $(EXTRAFLAGS)

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

OBJS        =\
            tmr_upd_win.o\
            cbr_exit_com.o\
            update_win.o\
            h_msg_look.o

#
# Define all header files required.
#

HDRS        =\
            $(INCDIR)/h_msg_look.h\
            $(INCDIR)/h_msg_look.bit\
            $(INCDIR)/hisde.h

#
# Make the target.
#

all:        $(TARGET)

$(TARGET):  $(OBJS)
    $(CC) -o $@ $(OBJS) $(LIBRARIES) $(LDFLAGS)
    strip $(TARGET)
    mv $(TARGET) $(BINDIR)
```

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

```
/*****************************************************************************
* MODULE NAME AND FUNCTION: ( h_msg_look )                                   *
*                                                                            *
*       The h_msg_look client provides the user with the logged message window for the *
*       HISDE system.  It allows the user to view messages which have been received.  *
*                                                                            *
*       This client displays the message log file in a scroll window which allows the *
*       user to view the last 500 logged messages.                           *
*                                                                            *
*       This client uses a timer routine to check if new messages have been logged.  The *
*       default timer value is 2 seconds.  If the user wants to change the interval, *
*       he/she may do so in the command line when running h_msg_look by using the *
*       '-interval' option.  Whenever the timer expires, the last position written to *
*       is read from the message log file and compared to the previous value read from *
*       the file.  If the value has changed it is an indication that new messages have *
*       been written to the file.                                            *
*                                                                            *
*                                                                            *
* DESCRIPTION OF MAIN FUNCTION:                                              *
*                                                                            *
*       This is the main driver for the h_msg_look client of the HISDE system.  It *
*       initializes the X Windows system and then creates the widgets necesssary for the *
*       h_msg_look window.   The window created contains a label for the message log *
*       window, an exit command button, and a scroll window for the display of the *
*       logged messages.                                                     *
*                                                                            *
*       This client will display the window and then enter the XtMainLoop routine *
*       and periodically update the display.  It will also handle the user selecting *
*       a command button.                                                    *
*                                                                            *
*       If the exit button is selected, the exit_command() function is executed and *
*       h_msg_look is terminated.                                            *
*                                                                            *
*       In order to periodically update the message log display, a timer is started *
*       before entering XtMainLoop.  When this timer expires, the update_msg_win() *
*       function is executed.  This function will access the message log file and *
*       redisplay the scroll window with its contents.  If any messages have been *
*       received since the last check, they will be displayed at the bottom of the *
*       current list.  Once the scroll window has been updated, the timer is started *
*       again.  This will continue until the user selects the exit button.   *
*                                                                            *
*                                                                            *
* SPECIFICATION DOCUMENTS:                                                   *
*                                                                            *
*      /hisde/req/requirements                                               *
*      /hisde/design/design                                                  *
*                                                                            *
*                                                                            *
* EXECUTION SEQUENCE:                                                        *
*                                                                            *
*       h_msg_look [-interval seconds]                                       *
*                                                                            *
*       In addition to the X Windows options which may be used when running h_msg_look, *
*       the following options are defined:                                   *
*                                                                            *
*       -interval [seconds]  - indicates the interval, in seconds, desired by the user. *
*                                                                            *
*                                                                            *
* FILES USED AND APPLICATION DEFINED FORMATS:                                *
*                                                                            *
*       /hisde/.msg_log - This file is used by the h_msg_look client to retreieve *
*                         all messages received in the message queue.  It is set up *
*                         as a circular file with a maximum number of messages. *
*                         Because it is a circular file, each message written *
```

```
*                       to this file must be of the same length.  Therefore,
*                       each message is read into a blank message buffer of
*                       the maximum message size possible.  In order to
*                       maintain this file, the last position written to
*                       in the file the last time a message was added is
*                       stored at the beginning of the file.  This maximum
*                       sizes for this file are defined in the h_logfiles.h
*                       header file.
*
*           struct .msg_log {
*               char[POSITION_OFFSET]               last_position;
*               char[MAX_NUM_MSG * MAX_MESSAGE]     messages;
*           }
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*      Nancy L. Martin - Software Engineering Section
*                        Data System Science and Technology Department
*                        Automation and Data Systems Division
*                        Southwest Research Institute
*******************************************************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Cardinals.h>
#include <X11/Shell.h>
#include <Xm/MainW.h>
#include <Xm/RowColumn.h>
#include <Xm/Form.h>
#include <hisde.h>
#include <h_user_inter.h>
#include <h_msg_look.h>
#include <h_msg_look.bit>
#include <h_logfiles.h>


/*
 *  Declare all external widgets to be used by the h_msg_look application.
 *  This is required for their use in the callback and action routines.
 */

Widget  top, m_main, mb_main, mp_file, form, widget, msg_scrll;

/*
 *  Declare the interval to be used for redisplaying the message log.
 *  It's default is 2 seconds.  This may be changed in the command line
 *  with the -interval parameter.
 */

unsigned    long    timer_interval = DEFAULT_INTERVAL;

/*
 *  Declare the callback procedures to be executed when a command button is selected.
 */

extern  XtCallbackProc          exit_command();

/*
 *  Declare the callback procedure to be executed when the timer value expires.
 */
```

```
extern  XtTimerCallbackProc     update_msg_win();


main ( argc, argv )
    int     argc;
    char    **argv;
{
/*
 *  Declare the application-specific resources allowed by this client.  The
 *  resource which may be set is the interval desired for updating the scroll
 *  window.
 */
    static XrmOptionDescRec options[] = {
        {"-interval",   "Interval", XrmoptionSepArg,    NULL  }
    };

    static  XtResource  resources[] = {
   { "interval",    "Interval", XtRInt,     sizeof(int),        (Cardinal)&timer_interval,
                    XtRInt,     (caddr_t)&timer_interval }
   };

/*
 *  Declare the callback list array to be used when creating command widgets.
 *  This array will contain the routines to be executed when the associated
 *  command button is selected.
 */
    static  XtCallbackRec   command_callbacks[] = {
        {(XtCallbackProc) NULL,             (caddr_t) NULL  },
        {(XtCallbackProc) NULL,             (caddr_t) NULL  }
    };

    Arg             icon_arg,       /* Argument used to initialize the icon.
                                     */
                    args[ 1 ];      /* Argument list used to initialize various
                                     * widget resources.
                                     */

    XtIntervalId    id;             /* The ID necessary for identifying the timer.
                                     */

/*
 *  Initialize the X Windows system and create the top level widget for the
 *  message log screen.
 */
    top = XtInitialize ( MESSAGE_LOG_SHELL, MESSAGE_LOG_CLASS, options, XtNumber(options),
                         &argc, argv );
/*
 *  If there were invalid arguments on the command line which could not be parsed,
 *  call the function, bad syntax, to display the correct syntax and exit from
 *  the client.
 */

    if ( argc > 1 )
        bad_syntax ( "h_msg_look [-interval time]" );

/*
 *  Initialize the icon bitmap for this client.
 */

    XtSetArg ( icon_arg, XtNiconPixmap,
               XCreateBitmapFromData ( XtDisplay(top), XtScreen(top) -> root,
               h_msg_look_bits, h_msg_look_width, h_msg_look_height ) );

    XtSetValues ( top, &icon_arg, ONE );
```

```
/*
 *  Retrieve any application-specific resources which were initialized previously or
 *  in the command line.  This includes the scroll window update interval.
 *  Multiply the specified interval by 1000 to convert in into milliseconds.
 */

    XtGetApplicationResources (top, (caddr_t)NULL, resources, XtNumber(resources),
                               NULL, ZERO );
    timer_interval = timer_interval *  1000;

/*
 *  Create the main window widget and the menu bar which will contain all commands.
 */

    m_main = XmCreateMainWindow ( top, "", NULL, 0 );
    XtManageChild ( m_main );

    mb_main = XmCreateMenuBar ( m_main, "", NULL, 0 );
    XtManageChild ( mb_main );

/*
 *  Create pulldown for file commands.
 */

    command_callbacks[ 0 ].callback = (XtCallbackProc)exit_command;
    mp_file = XmCreatePulldownMenu ( mb_main, "", NULL, 0 );
    create_cascade ( "", mb_main, mp_file, LABEL_FILE                    );
    create_command ( "", mp_file,          LABEL_EXIT, command_callbacks );

/*
 *  Create the help cascade.
 */

    widget = create_cascade ( "", mb_main, NULL, LABEL_HELP );
    XtSetArg ( args[ 0 ], XmNmenuHelpWidget, widget );
    XtSetValues ( mb_main, args, 1 );

/*
 *  Create the main form.
 */

    form = create_form ( "", m_main );

/*
 *  Create the text widget to be used as the message window.  It is created
 *  with a vertical scrollbar to allow the user to page through displayed
 *  messages.
 */

    msg_scrll = create_text ( W_T_MESS, form, "", 1, XmMULTI_LINE_EDIT, 0 );

/*
 *  Initialize the first iteration of the timer.  This will cause the update_msg_win
 *  callback routine to be executed.  This routine will reset the timer each time
 *  it completes its function.
 */

    id = XtAddTimeOut ( timer_interval, update_msg_win, NULL );

/*
 *  Call XtRealizeWidget on the top level widget to display the h_msg_look window.
 */
```

```
    XtRealizeWidget ( top );

/*
 *  Enter the Xtoolkit main loop to coordinate processing of all widget events.
 *  This loop is terminated when the user selects the exit command button and
 *  the associated callback procedure is executed to terminate this client.
 */

    XtMainLoop ( );
}
```

```
/*****************************************************************************
* MODULE NAME AND FUNCTION: exit_command()                                   *
*                                                                            *
*       The exit_command function is a callback procedure attached to the exit
*       command button of the h_msg_look client.  This function causes the client
*       to terminate naturally when the user selects the exit button.        *
*                                                                            *
*                                                                            *
* SPECIFICATION DOCUMENTS:                                                   *
*                                                                            *
*                                                                            *
*      /hisde/req/requirements                                               *
*      /hisde/design/design                                                  *
*                                                                            *
*                                                                            *
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)      *
*                                                                            *
*       top (Widget) (I) - The top level form widget for the h_msg_look client.
*                                                                            *
*                                                                            *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                        *
*                                                                            *
*      Nancy L. Martin - Software Engineering Section                        *
*                        Data System Science and Technology Department       *
*                        Automation and Data Systems Division                *
*                        Southwest Research Institute                        *
*****************************************************************************/


#include <X11/Intrinsic.h>


/*
 *  Declare the top level widget.
 */

extern  Widget  top;


XtCallbackProc exit_command ( 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.
                                     */
            calldata;               /* Specifies any callback-specific data the widget
                                     * needs to pass to the client.  This parameter is
                                     * is not used by this function.
                                     */

{
/*
 *  Remove the top level widget and then close the h_msg_look display.
 */

    XtUnmapWidget ( top );
    XCloseDisplay ( XtDisplay(top) );

/*
 *  Exit the h_msg_look client with a zero.
 */
```

```
    exit(0);
}
```

```
/*****************************************************************************
* MODULE NAME AND FUNCTION: update_msg_win()                                 *
*                                                                            *
*       This function is a timer callback procedure which is executed when the timer    *
*       interval expires.  This function updates the scroll window  with the contents    *
*       of the message log file if there have been messages added to the file.           *
*       (update_msg_win) determines whether there have been new messages added by        *
*       reading the position last written to from the beginning of the file and          *
*       comparing it to the value read from the file the last time an update was         *
*       necessary.  If these numbers are not the same then the file has been updated.    *
*                                                                                        *
*       Finally, update_msg_win reinitializes the timer value.  This will cause         *
*       update_msg_win to be called continually, at the specified interval, to update    *
*       the message log scroll window when necessary.                                    *
*                                                                                        *
*                                                                                        *
* SPECIFICATION DOCUMENTS:                                                               *
*                                                                                        *
*      /hisde/req/requirements                                                           *
*      /hisde/design/design                                                              *
*                                                                                        *
*                                                                                        *
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)                  *
*                                                                                        *
*       timer_interval (unsigned long) (I) -                                             *
*               The interval used to set the timer for checking message queues.  This    *
*               value is initialized to the the value defined as DEFAULT_INTERVAL in     *
*               the h_msg_look.h include file.  It may be changed in the command line    *
*               when executing this client.  This value should be given in seconds.      *
*               It will be converted to milliseconds programmatically.                   *
*                                                                                        *
*       msg_scrll (Widget) (I/O) -                                                       *
*               The file text widget created for the display of messages in the message  *
*               window.  It is created with a vertical scroll bar on the left hand side  *
*               to allow the user to page through displayed messages.                    *
*                                                                                        *
*                                                                                        *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                                    *
*                                                                                        *
*      Nancy L. Martin - Software Engineering Section                                    *
*                        Data System Science and Technology Department                   *
*                        Automation and Data Systems Division                            *
*                        Southwest Research Institute                                    *
*****************************************************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <hisde.h>
#include <h_msg_look.h>
#include <h_logfiles.h>


/*
 *  Declare the timer interval value for use in starting the timer back up.
 */

extern  long    timer_interval;

/*
 *  Declare the widgets which are accessed for the update.
 */
```

```
extern  Widget  msg_scrll;


XtTimerCallbackProc update_msg_win ( client_data, id )

    caddr_t         client_data;    /* Specifies the client date that was registered
                                     * registered for this procedure in XtAddTimeOut.
                                     */

    XtIntervalId    *id;            /* Specifies the ID returned from the corresponding
                                     * corresponding XtAddTimeOut call.
                                     */
{
    static  int     last_position = 0;
                                    /* The position value read from the file on the
                                     * previous update.
                                     */

    int             fd,             /* The file descriptor of the opened host bulletin
                                     * log file.
                                     */
                    new_position;   /* The value of the last position written to the
                                     * file.
                                     */

    char    position[POSITION_OFFSET + 1 ];
                                    /* The character string used to read in the last
                                     * position written to.
                                     */

/*
 *  Open the message log file for reading and read the value of
 *  the last position written to from the beginning of the file.
 */

    if ( ( fd = open ( HISDE_MSG_LOG, O_RDONLY ) ) <= NULL ) {
        h_message ( MSG_ERROR, "h_msg_look:  Cannot open message log file." );
        exit (-1);
    }

    if ( read ( fd, position, POSITION_OFFSET ) != POSITION_OFFSET ) {
        h_message ( MSG_ERROR, "h_msg_look:  Cannot read mesage log file position." );
        close (fd);
        exit (-1);
    }

/*
 *  Convert the character string read from the file to an integer and compare
 *  it to the value read from the file on the previous update.  If the
 *  value has changed, assign the new position offset to the static variable,
 *  last_position, for use in the next pass through this function.  Next, call
 *  update_window to read the messages from the file and update the message
 *  scroll window.
 */

    new_position = atoi ( position );
    if ( new_position != last_position ) {
        last_position = new_position;
        update_window ( fd, new_position );
    }

/*
 *  After the window has been updated, or if it did not need to be updated,
 *  close the message log file.
```

```
 */

    close (fd);

/*
 *  When the scroll window has been updated (if needed), reset the timer so that
 *  this routine will be called continually until the user selects to exit
 *  the h_msg_look client.
 */

    *id = XtAddTimeOut ( timer_interval, update_msg_win, NULL );
}
```

```
/*******************************************************************************
* MODULE NAME AND FUNCTION: update_window ( )                                  *
*                                                                              *
*       This function is called to read in the logged messages from the message log file *
*       starting with the oldest message.  As each message is read, it will be *
*       concatenated onto the end of the buffer to be written in the message scroll *
*       window.  When all messages have been read from the file, update_text_widget() *
*       is called with the buffer of logged messages to update the message scroll window *
*       with the new messages.  The cursor will then placed at the beginnning of the *
*       newly added messages.                                                  *
*                                                                              *
*       In order to determine where the first mesage is in the circular log file, *
*       update_window will attempt to read the first mesage past the last position *
*       written to in the file.  If there is a message in this position then the file is *
*       full and this message is the oldest message.  If there is not a message *
*       following the last position written to, the file is not yet full and the oldest *
*       message is the first message in the file.                              *
*                                                                              *
*                                                                              *
* SPECIFICATION DOCUMENTS:                                                     *
*                                                                              *
*      /hisde/req/requirements                                                 *
*      /hisde/design/design                                                    *
*                                                                              *
*                                                                              *
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)        *
*                                                                              *
*       msg_scrll (Widget) (I/O) - Text widget created for display of host messages. *
*                                                                              *
*                                                                              *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                          *
*                                                                              *
*      Nancy L. Martin - Software Engineering Section                          *
*                        Data System Science and Technology Department         *
*                        Automation and Data Systems Division                  *
*                        Southwest Research Institute                          *
*******************************************************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <hisde.h>
#include <h_logfiles.h>


extern Widget   msg_scrll;


update_window ( fd, new_position )

    int     fd,                     /* Specifies the file descriptor for the host
                                     * host bulletin log file.
                                     */

            new_position;           /* Specifies the last position written to the host
                                     * bulletin log file.
                                     */
{
    int     i,                      /* Used to initialize the message buffer to blanks.
                                     */
            position;               /* Maintains the current position in the file.
                                     */

    char    buffer      [ MAX_MESSAGE + 1  ];
```

```
                                        /* Used to read in each host message.
                                         */
    char    display_msg[ MAX_MSG_LOG + 1 ];
                                        /* Buffer which will contain all host messages.
                                         */

/*
 *  Initialize the scroll window buffer to blanks and assign the first position
 *  to be null for concatenation purposes.
 */

    for ( i = 0; i < MAX_MSG_LOG; i++ )
        display_msg[i] = BLANK;
    display_msg[0] = NULL;

/*
 *  Assign the last position written to as the position to seek to for reading.
 */

    position = new_position;

/*
 *  Try to read the next message ofter the most recently added message.  If
 *  the read fails, set the file position to the first message in the file
 *  past the position value, read that message, and assign the file position
 *  to be this message's starting point.
 *
 *  If neither  read is successful, call h_message to inform the user that
 *  the message log file cannot be read, close the file, and exit h_msg_look.
 */

    lseek ( fd, position, 0L );
    if ( read ( fd, buffer, MAX_MESSAGE ) <= 0 ) {
        lseek ( fd, POSITION_OFFSET, 0L );
        position = POSITION_OFFSET;
        if ( read ( fd, buffer, MAX_MESSAGE ) <= 0 ) {
            h_message ( MSG_ERROR, "h_msg_look: Cannot read first log file message." );
            close (fd);
            exit (-1);
        }
    }

/*
 *  If the oldest message was successfully read from the file, append it to the
 *  message buffer.  Update the file position pointer to point to the next message.
 *  Each message read from the file is the same size, MAX_MESSAGE.
 */

    strcat ( display_msg, buffer );
    position += MAX_MESSAGE;

/*
 *  If the new file position is greater than or equal to the maximum size of the
 *  message log file, wrap around to the first message in the file.  Note:
 *  the first message in the file is located after the value indicating the
 *  last position written to in the file.  This value is of the size,
 *  POSITION_OFFSET.
 */

    if ( position >= MAX_MSG_LOG )
        position = POSITION_OFFSET;

/*
 *  Loop through the file reading the next message until the end of file is reached
```

```
     *  or the file position returns to the oldest message.
     *
     *  For each message, the message is attached to the end of the message buffer.
     *  The file position is updated to point to the next message in the file each
     *  time.
     */

        while ( ( read ( fd, buffer, MAX_MESSAGE ) > 0 ) && ( position != new_position ) ) {
            strcat ( display_msg, buffer );
            position += MAX_MESSAGE;
            if ( position >= MAX_MSG_LOG ) {
                position = POSITION_OFFSET;
                lseek ( fd, position, 0L );
            }
        }

    /*
     *  Update the text widget.
     */

        update_text_widget ( msg_scrll, display_msg );
        XmTextSetInsertionPosition ( msg_scrll, strlen ( display_msg ) );
    }
```

```
#############################################################################################
# Makefile for HISDE user interface client (h_pbi_edit)                                     #
#############################################################################################

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

TARGET      = h_pbi_edit

#
# Initialize include and library search paths to include current directory and the
# HISDE directories.  Note that the library path also includes the user interface
# library.
#

BINDIR          = /hisde/bin
INCDIR          = /hisde/src/include
INCDIRS         = -I. -I$(INCDIR)

#
# Define the libraries to search.  This includes the HISDE library, the local user
# interface library, and all required X libraries.
#

LIBRARIES       = -lui -lhisde -lXm -lXt -lX11

#
# Define the compiler and linker flags.
#

CFLAGS          = -O $(INCDIRS)
LDFLAGS         = -O $(EXTRAFLAGS)

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

OBJS        =\
            save_pbi.o\
            cbr_file.o\
            cbr_clear.o\
            cbr_edit_trm.o\
            h_pbi_edit.o

#
# Define all header files required.
#

HDRS        =\
            $(INCDIR)/h_user_inter.h\
            $(INCDIR)/h_pbi_edit.bit\
            $(INCDIR)/h_pbi_edit.h\
            $(INCDIR)/hisde.h

#
# Make the target.
#

all:        $(TARGET)

$(TARGET):  $(OBJS)
    $(CC) -o $@ $(OBJS) $(LIBRARIES) $(LDFLAGS)
    strip $(TARGET)
```

```
	mv $(TARGET) $(BINDIR)

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

```
/*****************************************************************************************
* MODULE NAME AND FUNCTION: ( h_pbi_edit )                                               *
*                                                                                        *
* This HISDE client is provides a convienient means by which the user can build and edit *
* Push-Button Indicator (PBI) files.  A PBI file is used to display a grid of buttons    *
* which may be selected to initiate some type of event.  This is a rough emulation of    *
* the PBI machines present on the MCC floor.  For more information on PBI's, refer to    *
* the documentation on the (h_pbi) client.                                               *
*                                                                                        *
* All PBI files are simply ascii files which contain a logical record for each PBI item. *
* For a discussion of the format of these records, refer to the documentation in the     *
* (h_pbi) client.                                                                        *
*                                                                                        *
* When this client executes, it will present a window focused on a text widget (with     *
* scrollbar) which allows any PBI file to be viewed and edited.  In addition, the user   *
* may select any of the following commands:                                              *
*                                                                                        *
*      Exit - Exit from the this client.                                                 *
*                                                                                        *
*      Clear - Clear the text widget of all data.                                        *
*                                                                                        *
*      Load - Load the contents of a PBI into the text widget.  Note that this erases    *
*             the current contents of this window.                                       *
*                                                                                        *
*      Merge - Merge the contents of a PBI at the current text widget cursor position.   *
*              Note that the current contents of the window will not be removed.         *
*                                                                                        *
*      Save - Save the contents of the PBI to a file.  Note that the current contents of *
*             the window will not be removed.                                            *
*                                                                                        *
* Note that in order to use the load, merge, or save commands, the user must first enter *
* a PBI filename.  A PBI filename is simply the name of the file.  It may be any normal  *
* UNIX path name, which includes either a full pathname or a partial path.  Note that    *
* the filename itself is free format.  The user is free to choose his own naming stan-   *
* dards and extensions.                                                                  *
*                                                                                        *
*                                                                                        *
* DESCRIPTION OF MAIN FUNCTION:                                                          *
*                                                                                        *
* This is the main function of the h_pbi_edit client.  It is responsibile for initial-   *
* ization of the resource database and all widgets which make up the client window.      *
* Once all widgets and their associated callbacks are initialized and realized, this     *
* routine calls the Xtoolkit intrinsic (XtMainLoop) to process all incoming events.      *
*                                                                                        *
* The window presented by this client consists of a hierarchy of widgets.  Essentially,  *
* it consists of a main form with several child forms, each of which present one major   *
* function.  Each child form in turn controls several widgets.  The full hierarchy of    *
* widgets is summarized below:                                                           *
*                                                                                        *
*      top -----> form --+--> form (Client  ) --+--> label                               *
*                        |         (ID      )   +--> command (clear PBI)                 *
*                        |                      +--> command (exit client)              *
*                        |                                                              *
*                        +--> form (PBI     ) --+--> text    (actual PBI data)          *
*                        |                                                              *
*                        +--> form (Filename) --+--> label                              *
*                                  (Function)   +--> text    (filename data)            *
*                                               +--> command (load command)             *
*                                               +--> command (merge command)            *
*                                               +--> command (save command)             *
*                                                                                        *
* Each of the forms used is offset from other forms to maintain a consistent layout of   *
* information.  The widgets with each form are in turn offset from one another in the    *
* same way.  This insures that homogenous widgets remain in close proximity and in a     *
```

```
* sensible arrangement.                                                        *
*                                                                             *
* Once this function calls XtMainLoop, there are a number of callback events which may
* be executed.  These functions, the command widgets to which they are tied, and the
* operations they perform are as follows:                                     *
*                                                                             *
*      function            event              operation                     *
*      --------            -----              ---------                     *
*      cbr_edit_terminate  exit               terminate h_pbi_edit client   *
*      cbr_clear           clear              clear PBI text                *
*      cbr_load            load               load a PBI file               *
*      cbr_merge           merge              merge a PBI file              *
*      cbr_save            save               save a PBI file               *
*                                                                             *
* For more information on these callback functions, refer to the appropriate source
* code file.                                                                  *
*                                                                             *
*                                                                             *
* SPECIFICATION DOCUMENTS:                                                    *
*                                                                             *
*      /hisde/req/requirements                                                *
*      /hisde/design/design                                                   *
*                                                                             *
*                                                                             *
* EXECUTION SEQUENCE:                                                         *
*                                                                             *
*      h_pbi_edit [-pbi pbi_file]                                             *
*                                                                             *
*      -pbi pbi_file - optional parameter which allows the user to specify the initial
*                      pbi file to be edited.  If an existing file is specified, it will
*                      be loaded and displayed.                               *
*                                                                             *
*                                                                             *
* FILES USED AND APPLICATION DEFINED FORMATS:                                 *
*                                                                             *
*      PBI File                                                               *
*                                                                             *
*      A PBI file is a normal UNIX ascii file which contains an arbitrary number of
*      logical lines.  Refer to the documentation in the (h_pbi) client for a descrip-
*      tion of valid PBI lines.                                               *
*                                                                             *
*                                                                             *
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)       *
*                                                                             *
*      This routine initializes all declared widget variables.                *
*                                                                             *
*                                                                             *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                         *
*                                                                             *
*      Mark D. Collier - Software Engineering Section                         *
*                        Data System Science and Technology Department        *
*                        Automation and Data Systems Division                 *
*                        Southwest Research Institute                         *
*******************************************************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Cardinals.h>
#include <X11/Shell.h>
#include <X11/MwmUtil.h>
#include <Xm/MainW.h>
#include <Xm/RowColumn.h>
```

```
#include <Xm/Form.h>
#include <Xm/FileSB.h>
#include <h_pbi_edit.bit>
#include <hisde.h>
#include <h_user_inter.h>
#include <h_pbi_edit.h>


char            file[ SIZE_FILENAME + 1 ] = "";

/*
 *  Declare all widgets which will be used by this client.  Again, this data is made
 *  external to allow simple access in callback functions.
 */

Widget          top,    m_main, mb_main, mp_file, mp_edit, widget,
                f_pbi, t_pbi,
                f_file, l_file, t_file,
                f_popup, t_popup;

/*
 *  Declare all callback functions.
 */

extern  XtCallbackProc      cbr_edit_terminate(),
                            cbr_clear         (),
                            cbr_file          ();


main ( argc, argv )
    int     argc;
    char    **argv;
{
/*
 *  Define the application-specific resources allowed by this client.  The only resource
 *  which may be set is the initial PBI to be edited.  Note that if the user specified
 *  a PBI file, the pointer (fp) will be set to address it.
 */

    static XrmOptionDescRec options[] = {
        { "-pbi", "Pbi", XrmoptionSepArg, NULL }
    };

    static  char    *fp;

    static XtResource   resources[] = {
        { "pbi", "Pbi", XtRString, sizeof(char *), (Cardinal) &fp,
          NULL, (caddr_t)NULL }
    };

/*
 *  Initialize each of the callback lists used for the commands generating this type of
 *  event.  These include the exit, clear, and file commands.
 */

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

    static XtCallbackRec cb_clear[ ] = {
        { (XtCallbackProc)cbr_clear,          (caddr_t)NULL },
        { (XtCallbackProc)NULL,               (caddr_t)NULL }
    };
```

```
    static XtCallbackRec cb_file[ ] = {
        { (XtCallbackProc)cbr_file,          (caddr_t)NULL },
        { (XtCallbackProc)NULL,              (caddr_t)NULL }
    };

    static XtCallbackRec cb_file1[ ] = {
        { (XtCallbackProc)cbr_file,          (caddr_t)NULL },
        { (XtCallbackProc)NULL,              (caddr_t)NULL }
    };

    Arg icon_arg,                    /* Argument used to initialize the graphic icon for
                                      * this client.
                                      */
        args[ 2 ];                   /* Argument list used to initialize various widget
                                      * values.
                                      */

/*
 *  Initialize the Xtoolkit, parse command line, and return the root widget which will be
 *  the parent of the window.
 */

    top = XtInitialize ( NAME_SHELL, NAME_APLIC, options, XtNumber(options), &argc, argv )
;

/*
 *  If there were arguments on the command line which could not be parsed, call the
 *  function (bad_syntax) to report the error, display the correct syntax, and exit from
 *  the client.
 */

    if ( argc > 1 )
        bad_syntax ( "h_pbi_edit [-pbi pbi_file]" );

/*
 *  Initialize the icon bitmap for this client.
 */

    XtSetArg ( icon_arg, XtNiconPixmap,
                    XCreateBitmapFromData ( XtDisplay(top), XtScreen(top)->root,
                            h_pbi_edit_bits, h_pbi_edit_width, h_pbi_edit_height ) );

    XtSetValues ( top, &icon_arg, ONE );

/*
 *  Parse all application-specific resources.  The only resource which is present is
 *  the initial PBI to edit.  If specified, the pointer (fp) will point to the specified
 *  PBI file.
 */

    XtGetApplicationResources( top, (caddr_t)NULL, resources, XtNumber(resources),
                               NULL, ZERO );

/*
 *  Create the main window widget and the menu bar which will contain all commands.
 */

    m_main = XmCreateMainWindow ( top, "", NULL, 0 );
    XtManageChild ( m_main );

    mb_main = XmCreateMenuBar ( m_main, "", NULL, 0 );
    XtManageChild ( mb_main );
```

```
/*
 *  Create pulldown for file commands.
 */

    mp_file = XmCreatePulldownMenu ( mb_main, "", NULL, 0 );
    create_cascade ( "", mb_main, mp_file, LABEL_FILE                    );

    cb_file[ 0 ].closure = (caddr_t)CB_NEW;
    create_command ( "", mp_file,          LABEL_NEW,    cb_file      );
    cb_file[ 0 ].closure = (caddr_t)CB_MERGE;
    create_command ( "", mp_file,          LABEL_MERGE,  cb_file      );
    cb_file[ 0 ].closure = (caddr_t)CB_SAVE;
    create_command ( "", mp_file,          LABEL_SAVE,   cb_file      );
    cb_file[ 0 ].closure = (caddr_t)CB_SAVEAS;
    create_command ( "", mp_file,          LABEL_SAVEAS, cb_file      );

    create_command ( "", mp_file,          LABEL_EXIT,   cb_terminate );

/*
 *  Create pulldown for edit commands.
 */

    mp_edit = XmCreatePulldownMenu ( mb_main, "", NULL, 0 );
    create_cascade ( "", mb_main, mp_edit, LABEL_EDIT            );
    create_command ( "", mp_edit,          LABEL_CLEAR, cb_clear );

/*
 *  Create the help cascade.
 */

    widget = create_cascade ( "", mb_main, NULL, LABEL_HELP );
    XtSetArg ( args[ 0 ], XmNmenuHelpWidget, widget );
    XtSetValues ( mb_main, args, 1 );

/*
 *  Initialize the child form which will contain the actual PBI viewing/editing area.
 *  Note that the text widget provides a scrollbar and is editable.
 */

    f_pbi = create_form ( W_F_PBI_M, m_main );
    t_pbi = create_text ( W_T_PBI_M, f_pbi, "", 1, XmMULTI_LINE_EDIT, 1 );

/*
 *  Define the areas which constitute the main window widget.
 */

    XmMainWindowSetAreas ( m_main, mb_main, NULL, NULL, NULL, f_pbi );

/*
 *  Create the popup file selection widget. This includes attaching callbacks to the OK
 *  and CANCEL pushbuttons
 */

    cb_file [ 0 ].closure = (caddr_t)CB_OK;
    XtSetArg ( args[ 0 ], XmNokCallback,     cb_file );
    cb_file1[ 0 ].closure = (caddr_t)CB_CANCEL;
    XtSetArg ( args[ 1 ], XmNcancelCallback, cb_file1 );
    f_popup = XmCreateFileSelectionDialog ( top, "", args, 2 );

    XtSetArg ( args[ 0 ], XmNmwmInputMode, MWM_INPUT_APPLICATION_MODAL );
    XtSetValues ( XtParent ( f_popup ), args, 1 );

/*
 *  Save the widget pointer for the text widget in the file selection box.
```

```
 */

    t_popup = XmFileSelectionBoxGetChild ( f_popup, XmDIALOG_TEXT );


/*
 *  Realize the top level widget.  This causes the main form of this client to be
 *  displayed.
 */

    XtRealizeWidget ( top );

/*
 *  If the user specified a PBI file, load it into the text widget.
 */

    if ( fp ) {
        strcpy ( file, fp );
        cbr_file ( NULL, CB_MAIN, NULL );
    }

/*
 *  Enter the normal Xtoolkit main loop, which coordinates processing of
 *  the various widget events.  This loop will terminate normally when the user selects
 *  the "Exit" command.
 */

    XtMainLoop ( );
}
```

```
/*****************************************************************************
* MODULE NAME AND FUNCTION ( cbr_clear )                                     *
*                                                                            *
* The callback function is executed when the user selects the clear command widget.  It *
* simply clears the text widget which presents the current PBI.              *
*                                                                            *
*                                                                            *
* SPECIFICATION DOCUMENTS:                                                   *
*                                                                            *
*      /hisde/req/requirements                                               *
*      /hisde/design/design                                                  *
*                                                                            *
*                                                                            *
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)      *
*                                                                            *
*      t_pbi (Widget) (I) - Pointer to the text widget containing the PBI.   *
*                                                                            *
*                                                                            *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                        *
*                                                                            *
*      Mark D. Collier - Software Engineering Section                        *
*                        Data System Science and Technology Department       *
*                        Automation and Data Systems Division                *
*                        Southwest Research Institute                        *
*****************************************************************************/


#include <X11/Intrinsic.h>


extern  Widget  t_pbi;


XtCallbackProc cbr_clear ( 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.
                                     */
            calldata;               /* Specifies any callback-specific data the widget
                                     * needs to pass to the client.  This parameter is
                                     * is not used by this function.
                                     */
{
/*
 *  Clear all text from the PBI text widget.
 */

    clear_text_widget ( t_pbi );
}
```

```
/*******************************************************************************
* MODULE NAME AND FUNCTION ( cbr_edit_terminate )                              *
*                                                                              *
* This callback function is activated when the user selects the exit command widget.  It *
* is responsible for normal termination of the h_pbi_edit client.  It simply destroys  *
* the top level widget, which in turn causes all subordinate widgets to be destroyed.  *
*                                                                              *
*                                                                              *
* SPECIFICATION DOCUMENTS:                                                     *
*                                                                              *
*      /hisde/req/requirements                                                 *
*      /hisde/design/design                                                    *
*                                                                              *
*                                                                              *
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)        *
*                                                                              *
*      top (Widget) (I) - Pointer to the root widget of the main window.       *
*                                                                              *
*                                                                              *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                          *
*                                                                              *
*      Mark D. Collier - Software Engineering Section                          *
*                        Data System Science and Technology Department         *
*                        Automation and Data Systems Division                  *
*                        Southwest Research Institute                          *
*******************************************************************************/


#include <X11/Intrinsic.h>


extern  Widget  top;


XtCallbackProc cbr_edit_terminate ( 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.
                                     */
            calldata;               /* Specifies any callback-specific data the widget
                                     * needs to pass to the client.  This parameter is
                                     * is not used by this function.
                                     */
{
    XEvent  event;                  /* Event structure needed to make the calls to the
                                     * XtNextEvent and XtDispatchEvent functions.
                                     */

/*
 *  Destroy the root application shell widget and thereby, all subordinate widgets which
 *  make up the window.
 */

    XtDestroyWidget ( top );

/*
 *  Determine if any events have been queued.  These will normally be events which
 *  cause the widgets destroy callback to be executed.  Waiting and then processing
 *  the events insures that all data structures initialized by the widgets are
 *  properly deallocated.
```

```
    */

        XtNextEvent     ( &event );
        XtDispatchEvent ( &event );

    /*
     *  Close the display to deallocate any connections set up by X Windows.  Next
     *  exit from the client.
     */

        XCloseDisplay ( XtDisplay ( top ) );
        exit ( 0 );
    }
```

```
/*****************************************************************************************
* MODULE NAME AND FUNCTION ( cbr_save )                                                  *
*                                                                                        *
* The callback function is executed when the user selects the save command widget.  It   *
* verifies the current contents of the PBI and if valid, saves them to the specified     *
* filename.  Note that this function does not check in any way, the format of the PBI    *
* information.  This may be added later when the final format is better defined.         *
*                                                                                        *
*                                                                                        *
* SPECIFICATION DOCUMENTS:                                                               *
*                                                                                        *
*      /hisde/req/requirements                                                           *
*      /hisde/design/design                                                              *
*                                                                                        *
*                                                                                        *
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)                  *
*                                                                                        *
*      t_pbi (Widget) (I)   - A pointer to the text widget used to display the PBI.      *
*                                                                                        *
*      file  (char[]) (I)   - String containing the current filename.                    *
*                                                                                        *
*      pbi   (char[]) (I/O) - String containing the current PBI.                         *
*                                                                                        *
*                                                                                        *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                                    *
*                                                                                        *
*      Mark D. Collier - Software Engineering Section                                    *
*                        Data System Science and Technology Department                   *
*                        Automation and Data Systems Division                            *
*                        Southwest Research Institute                                    *
*****************************************************************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <hisde.h>
#include <h_user_inter.h>
#include <string.h>


int save_pbi ( file, t_pbi )          /* Function returns a value indicating whether or not
                                       * the file could be saved:
                                       *
                                       *   0 - Success
                                       *  -1 - Failure
                                       */
    char            *file;            /* File to save the contents of the PBI to.
                                       */

    Widget          t_pbi;            /* Text widget containing the PBI data.
                                       */
{
    FILE            *fp;              /* File pointer used to open and write into the
                                       * PBI file.
                                       */

    register char   *p,               /* Pointer used to step through the PBI and write
                                       * into file.
                                       */
                    *pbi;             /* Pointer to pbi text returned from widget.
                                       */

/*
 *  Retrieve all text from the widget.
```

```
 */

    pbi = get_text_widget ( t_pbi );

/*
 *  Open the file for write access.  If an error occurs, output a message to the
 *  system message client and return.
 */

    if ( ( fp = fopen ( file, "w" ) ) == NULL ) {
        h_message ( MSG_WARNING, "Could not open the specified file" );
        XtFree ( pbi );
        return;
    }

/*
 *  Write all PBI data to the file. When complete, free memory allocated for the widget
 *  text.
 */

    p = pbi;
    while ( *p )
        putc ( *p++, fp );

    XtFree ( pbi );

/*
 *  Close the file.  If an error occurs while closing the file, output a message to the
 *  system message client.  Otherwise, inform the user that the file was successfully
 *  saved.
 */

    if ( fclose ( fp ) )
        return ( display_message ( MSG_ERROR, "Could not properly close the PBI file" ) );

    display_message ( MSG_INFORMATION, "PBI file was successfully saved" );
    return ( 0 );
}
```

```
/*****************************************************************************
* MODULE NAME AND FUNCTION ( cbr_file )                                      *
*                                                                            *
* This callback function is executed when the user selects the load command  *
* opens the filename specified by the user, reads all contained data, and pl *
* the PBI text widget.  In doing so, any previously displayed PBI will be re *
*                                                                            *
*                                                                            *
* SPECIFICATION DOCUMENTS:                                                   *
*                                                                            *
*      /hisde/req/requirements                                               *
*      /hisde/design/design                                                  *
*                                                                            *
*                                                                            *
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)      *
*                                                                            *
*      t_pbi    (Widget) (I)   - A pointer to the text widget used to displ *
*                                                                            *
*      f_popup  (Widget) (I)   - A pointer to the form widget used for the p *
*                                                                            *
*      t_popup  (Widget) (I)   - A pointer to the text widget used for the p *
*                                                                            *
*      file     (char[]) (I/O) - String containing the current filename.     *
*                                                                            *
*                                                                            *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                        *
*                                                                            *
*      Mark D. Collier - Software Engineering Section                        *
*                        Data System Science and Technology Department       *
*                        Automation and Data Systems Division                *
*                        Southwest Research Institute                        *
*****************************************************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/FileSB.h>
#include <hisde.h>
#include <h_user_inter.h>
#include <h_pbi_edit.h>


extern  char    file[ ];

extern  Widget  t_pbi,
                f_popup, t_popup;


XtCallbackProc cbr_file ( 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.
                                     */
            calldata;               /* Specifies any callback-specific data the widget
                                     * needs to pass to the client.  This parameter is
                                     * is not used by this function.
                                     */
{
    static int      start_cmd = -1; /* Static value used to contain the command which
                                     * initiated the popup display.
```

```
                                        */

    register int    pos = -1,           /* Paramter for (load_text_widget). Causes new
                                         * data to replace or be merged into the current
                                         * PBI text widget.
                                         */
                    cmd,                /* Set to the command which initiated this callback.
                                         */
                    status;             /* Used to save the status of calls made to load and
                                         * save PBI's.
                                         */

    char            *file_temp;         /* Temporary string which will point to the filename
                                         * specified in the popup.
                                         */

/*
 *  Convert the (closure) parameter to a normal value to ease comparison.
 */

    cmd = (int)closure;

/*
 *  If the function was called from the main (command line argument), simulate a call
 *  after a normal popup. First save file in the popup text widget and then make it
 *  look like a new call after the popup.
 */

    if ( cmd == CB_MAIN ) {
        update_text_widget ( t_popup, file );
        start_cmd = CB_NEW;
        cmd       = CB_OK;
    }

/*
 *  If a PBI command (instead of a popup), save the command and if no file yet
 *  specified or the command requires a new filename, display the popup with the current
 *  filename.
 */

    if ( cmd == CB_NEW || cmd == CB_MERGE || cmd == CB_SAVE || cmd == CB_SAVEAS ) {
        start_cmd = cmd;
        if ( file[ 0 ] == NULL || cmd != CB_SAVE ) {
            update_text_widget ( t_popup, file );
            XtManageChild ( f_popup );
            return;
        }
    }

/*
 *  At this point assume that popup is displayed and a file was entered, so remove
 *  the popup (this is unnessary if called from the main function).
 */

    XtUnmanageChild ( f_popup );

/*
 *  If user selected cancel or help commands on the popup, simply return.
 */

    if ( cmd == CB_CANCEL || cmd == CB_HELP )
        return;

/*
```

```
     *  Get the text from the widget.
     */

        file_temp = get_text_widget ( t_popup );

    /*
     *  Process the commands. If new or merge, then if merge, get the current position of
     *  the cursor. Next attempt to load the file into the text widget at the appropriate
     *  position (beginning or at cursor position). Note that if (pos) is still -1, the
     *  new data will replace the old data.
     */

        if ( start_cmd == CB_NEW || start_cmd == CB_MERGE ) {
            if ( start_cmd == CB_MERGE )
                pos = get_text_insertion_widget ( t_pbi );

            if ( status = load_text_widget ( file_temp, t_pbi, pos ) )
                display_message ( MSG_WARNING, "Could not open the specified file" );

    /*
     *  Otherwise (save commands), save the pbi to the file. Error messages are generated
     *  internally to this function.
     */

        } else
            status = save_pbi ( file_temp, t_pbi );

    /*
     *  If the command was not merge (which does not change the filename) and if a new
     *  command did not fail, update the filename with the new name.
     */

        if ( start_cmd != CB_MERGE && ! ( start_cmd == CB_NEW && status ) )
            strcpy ( file, file_temp );

        XtFree ( file_temp );
    }
```

```
#############################################################################
# Makefile for HISDE user interface client (h_talk)                         #
#############################################################################

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

TARGET      = h_talk

#
# Initialize include and library search paths to include current directory and the
# HISDE directories.  Note that the library path also includes the user interface
# library.
#

BINDIR          = /hisde/bin
INCDIR          = /hisde/src/include
INCDIRS         = -I. -I$(INCDIR)

#
# Define the libraries to search.  This includes the HISDE library, the local user
# interface library, and all required X libraries.
#

LIBRARIES       = -lui -lhisde -lXm -lXt -lX11

#
# Define the compiler and linker flags.
#

CFLAGS          = -O $(INCDIRS)
LDFLAGS         = -O $(EXTRAFLAGS)

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

OBJS        =\
            cbr_node.o\
            cbr_clear.o\
            cbr_send.o\
            tmr_recv.o\
            cbr_talk_trm.o\
            h_talk.o

#
# Define all header files required.
#

HDRS        =\
            $(INCDIR)/h_user_inter.h\
            $(INCDIR)/h_talk.bit\
            $(INCDIR)/h_talk.h\
            $(INCDIR)/hisde.h

#
# Make the target.
#

all:        $(TARGET)

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

```
	strip $(TARGET)
	mv $(TARGET) $(BINDIR)

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

```
/*****************************************************************************
* MODULE NAME AND FUNCTION: ( h_talk )                                       *
*                                                                            *
* This client provides a simple means for a user to interactively communicate with a *
* user on a different workstation.  In this sense, it is a X Windows based version of *
* the UNIX (write) command.  In addition, this client uses the HISDE pipe manager for *
* all inter-node communication.  It therefore provides a good demonstration of this ap- *
* plications cababilities.                                                   *
*                                                                            *
* When this client executes, it will display a window which presents three basic fields. *
* These fields are as follows:                                               *
*                                                                            *
*     Node Name - This is the name of the node to which the user would like to communi- *
*                 cate.  It may be any existing node name in the network.    *
*                                                                            *
*     Send Text - This field allows the user to interactively enter the text which he *
*                 would like to send to the remote node.  This field includes commands *
*                 which allow the user to clear and send the text.           *
*                                                                            *
*     Receive Text - This field will be updated when new text is received.  It will ap- *
*                    pend the new text to the end of the existing text.  This field will *
*                    be cleared when it reaches the maximum allowable size.  It may also *
*                    be cleared by the user with the clear command.          *
*                                                                            *
* Note that this client must be executing on the node to which communcation is desired. *
* It is also necessary for the user on the remote system to specify the local user's *
* node.  This establishes a connection between the two nodes.  If these two steps are *
* not performed, no communications will be possible.                         *
*                                                                            *
*                                                                            *
* DESCRIPTION OF MAIN FUNCTION:                                              *
*                                                                            *
* This is the main function of the h_talk client.  It is responsible for initialization *
* of the resource database and all widgets which make up the client window.  Once all *
* widgets and their associated callbacks are initialized and realized, this function *
* calls the Xtoolkit intrinsic (XtMainLoop) to process all incoming events.  *
*                                                                            *
* The window presented by this client consists of a hierarchy of widgets.  Essentially, *
* it consists of a main form with several child forms, each of which present one major *
* function.  Each child form in turn controls several widgets.  The full hierarchy of *
* widgets is summarized below:                                               *
*                                                                            *
*      top -----> form --+--> form (Client  ) --+--> label                   *
*                        |         (ID      )   +--> command (exit client)   *
*                        |                                                  *
*                        +--> form (Nodename) --+--> label                   *
*                        |                      +--> command (change node)  *
*                        |                      +--> text    (nodename data)*
*                        |                                                  *
*                        +--> form (Send    ) --+--> label                   *
*                        |         (Window  )   +--> command (clear command)*
*                        |                      +--> command (send command) *
*                        |                      +--> text    (send data)    *
*                        |                                                  *
*                        +--> form (Receive ) --+--> label                   *
*                                               +--> command (clear command)*
*                                               +--> text    (receive data) *
*                                                                            *
* Each of the forms used is offset from other forms to maintain a consistent layout of *
* information.  The widgets with each form are in turn offset from one another in the *
* same way.  This insures that homogenous widgets remain in close proximity and in a *
* sensible arrangement.                                                      *
*                                                                            *
* Once this function calls XtMainLoop, there are a number of callback events which may *
```

```
 * be executed.  These functions, the command widgets to which they are tied, and the  *
 * operations they perform are as follows:                                             *
 *                                                                                     *
 *      function             event              operation                              *
 *      --------             -----              ---------                              *
 *      cbr_talk_terminate   exit command       terminate client                       *
 *      cbr_node             set node command   set the node with which to communicate *
 *      cbr_clear            clear command      clear send or receive text             *
 *      cbr_send             send command       send text to the node                  *
 *      tmr_recv             timer              check if data was received             *
 *                                                                                     *
 * For more information on these callback functions, refer to the appropriate source  *
 * code file.                                                                          *
 *                                                                                     *
 *                                                                                     *
 * SPECIFICATION DOCUMENTS:                                                            *
 *                                                                                     *
 *      /hisde/req/requirements                                                        *
 *      /hisde/design/design                                                           *
 *                                                                                     *
 *                                                                                     *
 * EXECUTION SEQUENCE:                                                                 *
 *                                                                                     *
 *      h_talk [-node nodename]                                                        *
 *                                                                                     *
 *      -node nodename - optional argument which if specified, will cause communications *
 *                       to the indicated node to be initiated.                        *
 *                                                                                     *
 *                                                                                     *
 * EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)               *
 *                                                                                     *
 *      This routine initializes all declared widget variables.                        *
 *                                                                                     *
 *                                                                                     *
 * ORIGINAL AUTHOR AND IDENTIFICATION:                                                 *
 *                                                                                     *
 *      Mark D. Collier - Software Engineering Section                                 *
 *                        Data System Science and Technology Department                *
 *                        Automation and Data Systems Division                         *
 *                        Southwest Research Institute                                 *
 ***************************************************************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Cardinals.h>
#include <X11/Shell.h>
#include <X11/MwmUtil.h>
#include <Xm/MainW.h>
#include <Xm/RowColumn.h>
#include <Xm/Text.h>
#include <Xm/Form.h>
#include <hisde.h>
#include <h_user_inter.h>
#include <h_talk.h>
#include <h_talk.bit>


/*
 *  Declare the variable which is set to the connection value needed to send and receive
 *  data from the connected workstation node.  This variable is initialized to -1 to
 *  indicate that it is not a connection number.  Note that this variable is external
 *  to allow its use in each of callback and timer functions which require it.
```

```
 */

int         pipe_num = -1;

/*
 *  Declare all widgets.  Again, they are made external to allow usage in any of the
 *  callback and timer functions.
 */

Widget      top, m_main, mb_main, mp_file, mp_edit, form, widget,
            t_send, t_recv,
            f_popup, c_popup, t_popup;

/*
 *  Declare the callback functions which are executed when the corresponding command
 *  widgets are selected.
 */

XtCallbackProc  cbr_talk_terminate(),
                cbr_node          (),
                cbr_clear         (),
                cbr_send          ();

/*
 *  Declare the timer function which is executed at a constant interval to determine if
 *  any data has been received.
 */

XtTimerCallbackProc     tmr_recv();


main ( argc, argv )
    int     argc;
    char    **argv;
{
/*
 *  Define the application-specific resources allowed by this client.  The only resource
 *  which may be set is the initial node with which to communicate.  Note that if the
 *  user specifies a node, the pointer (fp) will address the string.
 */

    static  char    *fp;

    static XrmOptionDescRec options[] = {
        { "-node", "Node", XrmoptionSepArg, NULL }
    };

    static XtResource   resources[] = {
        { "node", "Node", XtRString, sizeof(char *), (Cardinal)&fp,
          NULL, (caddr_t)NULL }
    };

/*
 *  Declare all the callback lists which are needed to initialize the various
 *  command widgets and callback functions.  These include the termination, node
 *  initialization, text clear, and text send functions.
 */

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

    static XtCallbackRec cb_node[] = {
```

```
        { (XtCallbackProc)cbr_node,             (caddr_t)NULL },
        { (XtCallbackProc)NULL,                 (caddr_t)NULL }
    };

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

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

    Arg icon_arg,
        args[ 1 ];

    static char     node[ SIZE_NODE + 1 ] = "";

/*
 *  Initialize the Xtoolkit, parse command line, and return the root widget which will be
 *  the parent of the window.
 */

    top = XtInitialize ( NAME_SHELL, NAME_APLIC, NULL, ZERO, &argc, argv );

/*
 *  If there were arguments on the command line which could not be parsed, call the
 *  function (bad_syntax) to report the error, display the correct syntax, and exit from
 *  the client.
 */

    if ( argc > 1 )
        bad_syntax ( "h_talk [-node nodename]" );

/*
 *  Initialize the icon bitmap for this client.
 */

    XtSetArg ( icon_arg, XtNiconPixmap,
                    XCreateBitmapFromData ( XtDisplay(top), XtScreen(top)->root,
                            h_talk_bits, h_talk_width, h_talk_height ) );

    XtSetValues ( top, &icon_arg, ONE );

/*
 *  Parse all application-specific resources.  The only resource which is present is
 *  the initial node to communicate with.  If specified, the pointer (fp) will address
 *  the specified node name.
 */

    XtGetApplicationResources( top, (caddr_t)NULL, resources, XtNumber(resources),
                               NULL, ZERO );

    if ( fp )
        strcpy ( node, fp );

/*
 *  Create the main window widget and the menu bar which will contain all commands.
 */

    m_main = XmCreateMainWindow ( top, "", NULL, 0 );
    XtManageChild ( m_main );
```

```
    mb_main = XmCreateMenuBar ( m_main, "", NULL, 0 );
    XtManageChild ( mb_main );

/*
 *  Create pulldown for file commands.
 */

    cb_node[ 0 ].closure = (caddr_t)CB_NODE;
    mp_file = XmCreatePulldownMenu ( mb_main, "", NULL, 0 );
    create_cascade ( "", mb_main, mp_file, LABEL_FILE                );
    create_command ( "", mp_file,          LABEL_NODE, cb_node       );
    create_command ( "", mp_file,          LABEL_SEND, cb_send       );
    create_command ( "", mp_file,          LABEL_EXIT, cb_terminate );

/*
 *  Create pulldown for clear commands.
 */

    mp_edit = XmCreatePulldownMenu ( mb_main, "", NULL, 0 );
    create_cascade ( "", mb_main, mp_edit, LABEL_EDIT            );
    create_command ( "", mp_edit,          LABEL_CLEAR, cb_clear );

/*
 *  Create the help cascade.
 */

    widget = create_cascade ( "", mb_main, NULL, LABEL_HELP );
    XtSetArg ( args[ 0 ], XmNmenuHelpWidget, widget );
    XtSetValues ( mb_main, args, 1 );

/*
 *  Create the form which is used for the main information window.  This form will be
 *  the parent to all widgets except those used for the monitor windows.
 */

    form = create_form ( "", m_main );

/*
 *  Create send text widget.
 */

            create_label ( W_L_SEND_M, form, LABEL_SEND_W );
    t_send = create_text  ( W_T_SEND_M, form, "", 1, XmMULTI_LINE_EDIT, 1 );

/*
 *  Create receive text widget.
 */

            create_label ( W_L_RECV_M, form, LABEL_RECV_W );
    t_recv = create_text  ( W_T_RECV_M, form, "", 1, XmMULTI_LINE_EDIT, 0 );

/*
 *  Create the dialog shell used for the popup. Note setting the MODAL flag on the
 *  widget returned by (XmCreateDialogShell) does not work. Therefore we get the
 *  parent of the form and set the value on it.
 */

    f_popup = XmCreateFormDialog ( top, W_F_POPUP_S, NULL, 0 );
    XtSetArg ( args[ 0 ], XmNmwmInputMode, MWM_INPUT_APPLICATION_MODAL );
    XtSetValues ( XtParent ( f_popup ), args, 1 );

/*
 *  Create the label, commands, and text widgets in the popup.
 */
```

```
    create_label ( W_L_POPUP_S, f_popup, "Enter Node:" );

    cb_node[ 0 ].closure = (caddr_t)CB_OK;
    c_popup = create_command ( W_C1_POPUP_S, f_popup, LABEL_OK,     cb_node );
    cb_node[ 0 ].closure = (caddr_t)CB_CANCEL;
              create_command ( W_C2_POPUP_S, f_popup, LABEL_CANCEL, cb_node );
    cb_node[ 0 ].closure = (caddr_t)CB_HELP;
              create_command ( W_C3_POPUP_S, f_popup, LABEL_HELP,   cb_node );

    t_popup = create_text ( W_T_POPUP_S, f_popup, node, 0, XmSINGLE_LINE_EDIT, 1 );

/*
 *  Set argument on first command to indicate that it is the default.
 */

    XtSetArg ( args[ 0 ], XmNshowAsDefault, TRUE );
    XtSetValues ( c_popup, args, 1 );

/*
 *  Initialize the first iteration of the timer.  This will cause the (tmr_recv)
 *  callback routine to be executed.
 */

    XtAddTimeOut ( TIMER_VALUE, tmr_recv, NULL );

/*
 *  Realize the top level widget.  This causes the main form of this client to be
 *  displayed.
 */

    XtRealizeWidget ( top );

/*
 *  If the user specified a node name, call the (cbr_node) function to attempt to
 *  initialize the node.
 */

    if ( fp )
        cbr_node ( NULL, CB_MAIN, NULL );

/*
 *  Enter the normal Xtoolkit main loop, which coordinates processing of the various
 *  widget events.  This loop will terminate normally when the user selects the
 *  "Exit" command.
 */

    XtMainLoop ( );
}
```

```
/*****************************************************************************
* MODULE NAME AND FUNCTION ( cbr_clear )                                     *
*                                                                            *
* This callback function is activated when the user selects the clear command.  This
* causes the send and receive message areas to be cleared.                   *
*                                                                            *
*                                                                            *
* SPECIFICATION DOCUMENTS:                                                   *
*                                                                            *
*      /hisde/req/requirements                                               *
*      /hisde/design/design                                                  *
*                                                                            *
*                                                                            *
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)      *
*                                                                            *
*      t_send (Widget) (I) - Pointer to the widget containing the send text. *
*                                                                            *
*      t_recv (Widget) (I) - Pointer to the widget containing the receive text.
*                                                                            *
*                                                                            *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                        *
*                                                                            *
*      Mark D. Collier - Software Engineering Section                        *
*                        Data System Science and Technology Department       *
*                        Automation and Data Systems Division                *
*                        Southwest Research Institute                        *
*****************************************************************************/


#include <X11/Intrinsic.h>


extern  Widget  t_send,
                t_recv;


XtCallbackProc cbr_clear ( widget, closure, calldata )

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

    caddr_t closure,                /* Callback specific data.  This value is set to the
                                     * widget which is to be cleared by this function.
                                     */
            calldata;               /* Specifies any callback-specific data the widget
                                     * needs to pass to the client.  This parameter is
                                     * is not used by this function.
                                     */
{
/*
 *  Clear the two widget.
 */

    clear_text_widget ( t_send );
    clear_text_widget ( t_recv );
}
```

```
/*****************************************************************************
* MODULE NAME AND FUNCTION ( cbr_node )
*
* This callback function is executed whenever the user enters a node name and selects *
* the node command widget.  This action indicates that the user wants to talk to a dif- *
* ferent workstation.  This function closes any existing connection and then attempts to *
* open a connection to the requested workstation.  If it completes successfully, the *
* external variable (pipe_num) will be set to the appropriate connection number needed *
* to send and receive information from the workstation.                              *
*                                                                                    *
*                                                                                    *
* SPECIFICATION DOCUMENTS:                                                           *
*                                                                                    *
*      /hisde/req/requirements                                                       *
*      /hisde/design/design                                                          *
*                                                                                    *
*                                                                                    *
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)              *
*                                                                                    *
*      pipe_num (int)     (I/O) - On input, this value is set to the current connection *
*                                 number (if any).  On exit, it will be set to the new *
*                                 number if a connection can be made.                *
*                                                                                    *
*      f_popup  (Widget) (I)    - Pointer to the popup form widget. Needed to manage and *
*                                 unmanage the popup.                                *
*                                                                                    *
*      t_popup  (Widget) (I)    - Pointer to the popup text widget. Needed to retrieve *
*                                 the specified node.                                *
*                                                                                    *
*                                                                                    *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                                *
*                                                                                    *
*      Mark D. Collier - Software Engineering Section                                *
*                        Data System Science and Technology Department               *
*                        Automation and Data Systems Division                        *
*                        Southwest Research Institute                                *
*****************************************************************************/

#include <X11/Intrinsic.h>
#include <pm_constants.h>
#include <hisde.h>
#include <h_user_inter.h>
#include <h_talk.h>


extern  Widget  f_popup, t_popup;

extern  int     pipe_num;


XtCallbackProc cbr_node ( 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.
                                     */
            calldata;               /* Specifies any callback-specific data the widget
                                     * needs to pass to the client.  This parameter is
                                     * is not used by this function.
                                     */
```

```
{
    static  int in_popup = FALSE;    /* Static variable used to indicate whether or not a
                                      * popup is displayed.
                                      */

    char            *node;
    extern int      errno;           /* UNIX error value which is set after system calls.
                                      */

/*
 *  If not in popup, then if not called from main (called from menu), set popup to
 *  TRUE and display the popup.
 */

    if ( in_popup == FALSE  && (int)closure != CB_MAIN ) {
        in_popup = TRUE;
        XtManageChild ( f_popup );

/*
 *  Otherwise, remove the popup if displayed (it will not be in the call from
 *  the main function.
 */

    } else {
        if ( in_popup ) {
            XtUnmanageChild ( f_popup );
            in_popup = FALSE;
        }

/*
 *      If the command was CANCEL or HELP, return.
 */

        if ( (int)closure == CB_CANCEL || (int)closure == CB_HELP )
            return;

/*
 *      Retrieve the text from the popup text widget. If blank, return.
 */

        node = get_text_widget ( t_popup );
        if ( *node == NULL ) {
            XtFree ( node );
            return;
        }

/*
 *      At this point, the callback will be CB_MAIN or CB_OK (and a node will have been
 *      entered).  If a connection is already open (user was communicating to another
 *      workstation), close the connection.  If this fails, output a warning to the
 *      system message window and return.
 */

        if ( pipe_num != -1 && h_close ( pipe_num ) == -1 ) {
            display_message ( MSG_ERROR, "Could not close the existing connection" );
            XtFree ( node );
            return;
        }

/*
 *      Attempt to open a connection to the requested workstation.  If the attempt fails
 *      because the (h_talk) client is not activeon the requested node, report that
 *      problem.  If any other error occurs (pipe_num = -1), output a warning.  Note
 *      that in both cases, the user will not be allowed to send (or receive) to the'
```

```
     *      workstation.
     */

            if ( ( ( pipe_num =  h_pipe ( node, APP1, APP2 ) ) == -1 ) &&
                 (   errno     == NO_SUCH_PROCESS                      )    )
                display_message ( MSG_WARNING, "Could not find (h_talk) on requested node" );
            else if ( pipe_num == -1 )
                display_message ( MSG_WARNING, "Could not communicate with requested node" );

    /*
     *      Otherwise, the connection was successfully opened, so output a message.
     */

            else
                display_message ( MSG_INFORMATION, "Connection successfully established" );

            XtFree ( node );
        }
    }
```

```
/*****************************************************************************
* MODULE NAME AND FUNCTION ( cbr_send )                                      *
*                                                                            *
* This callback function is executed when the user enters data into the send window and *
* then selects the send command widget.  This causes the entered data to be sent to the *
* workstation for which communications are currently established.  Note that this func- *
* tion will output a warning if the user has not yet connected to a remote node.  *
*                                                                            *
*                                                                            *
* SPECIFICATION DOCUMENTS:                                                   *
*                                                                            *
*      /hisde/req/requirements                                               *
*      /hisde/design/design                                                  *
*                                                                            *
*                                                                            *
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)      *
*                                                                            *
*      pipe_num (int)     (I) - Set to the current connection number needed to send and *
*                               receive data from the workstation.           *
*                                                                            *
*                                                                            *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                        *
*                                                                            *
*      Mark D. Collier - Software Engineering Section                        *
*                        Data System Science and Technology Department       *
*                        Automation and Data Systems Division                *
*                        Southwest Research Institute                        *
*****************************************************************************/


#include <X11/Intrinsic.h>
#include <hisde.h>
#include <h_user_inter.h>
#include <errno.h>


extern  Widget  t_send;
extern  int     pipe_num;


XtCallbackProc cbr_send ( 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.
                                     */
            calldata;               /* Specifies any callback-specific data the widget
                                     * needs to pass to the client.  This parameter is
                                     * is not used by this function.
                                     */
{
    char    *temp;                  /* Used to obtain the current contents of the send
                                     * window.
                                     */

/*
 *  If no connection has yet been established, output a warning to the system messsage
 *  client.
 */

    if ( pipe_num == -1 )
```

```
        display_message ( MSG_WARNING, "You have not yet specified a remote node" );

/*
 *  Otherwise, a connection has been made, so attempt to send the data to the remote
 *  node.  If this fails, output a message to the system message client.
 */

    else {
        temp = get_text_widget ( t_send );
        if ( h_write ( pipe_num, temp, strlen ( temp ) ) == -1 )
            display_message ( MSG_ERROR, "Could not write the requested data" );
    }
}
```

```
/*****************************************************************************
* MODULE NAME AND FUNCTION ( cbr_talk_terminate )                            *
*                                                                            *
* This callback function is activated when the user selects the exit command widget.  It *
* is responsible for normal termination of this client.  It simply destroys the top lev- *
* el widget, which in turn causes all subordinate widgets to be destroyed.  If neces-    *
* sary, it will also close the existing connection to the workstation.       *
*                                                                            *
*                                                                            *
* SPECIFICATION DOCUMENTS:                                                   *
*                                                                            *
*      /hisde/req/requirements                                               *
*      /hisde/design/design                                                  *
*                                                                            *
*                                                                            *
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)      *
*                                                                            *
*      pipe_num (int)     (I) - Set to the number required to communicate with the con-  *
*                               nected workstation.                          *
*                                                                            *
*      top       (Widget) (I) - Pointer to the root widget of the main window.          *
*                                                                            *
*                                                                            *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                        *
*                                                                            *
*      Mark D. Collier - Software Engineering Section                        *
*                        Data System Science and Technology Department       *
*                        Automation and Data Systems Division                *
*                        Southwest Research Institute                        *
*****************************************************************************/


#include <X11/Intrinsic.h>
#include <hisde.h>


extern  Widget  top;

extern  int     pipe_num;


XtCallbackProc cbr_talk_terminate ( 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.
                                     */
            calldata;               /* Specifies any callback-specific data the widget
                                     * needs to pass to the client.  This parameter is
                                     * is not used by this function.
                                     */
{
    XEvent  event;                  /* Event structure needed to make the calls to the
                                     * XtNextEvent and XtDispatchEvent functions.
                                     */

/*
 *  If a connection is open (pipe_num <> -1), attempt to close.  If this fails,
 *  output an error to the system message client.
 */
```

```
    if ( pipe_num != -1 && h_close ( pipe_num ) == -1 )
        display_message ( MSG_ERROR, "Could not close the established connection" );

/*
 *  Destroy the root application shell widget and thereby, all subordinate widgets which
 *  make up the window.
 */

    XtDestroyWidget ( top );

/*
 *  Determine if any events have been queued.  These will normally be events which
 *  cause the widgets destroy callback to be executed.  Waiting and then processing
 *  the events insures that all data structures initialized by the widgets are
 *  properly deallocated.
 */

    XtNextEvent     ( &event );
    XtDispatchEvent ( &event );

/*
 *  Close the display to deallocate any connections set up by X Windows.  Next
 *  exit from the client.
 */

    XCloseDisplay ( XtDisplay ( top ) );
    exit ( 0 );
}
```

```
/*****************************************************************************
* MODULE NAME AND FUNCTION ( tmr_recv )                                      *
*                                                                            *
* This timer function is executed at a defined interval, at which time is determines if *
* any data has been sent from the connected workstation.  If data has been received, *
* this function will update the receive window (t_recv widget) with the new text. *
*                                                                            *
*                                                                            *
* SPECIFICATION DOCUMENTS:                                                   *
*                                                                            *
*      /hisde/req/requirements                                               *
*      /hisde/design/design                                                  *
*                                                                            *
*                                                                            *
* EXTERNAL DATA USED: ('I' - Input  'O' - Output  'I/O' - Input/Output)      *
*                                                                            *
*      pipe_num (int)     (I) - Set to the current connection number needed to send and *
*                               receive data from the workstation.           *
*                                                                            *
*      t_recv   (Widget)  (I) - Pointer to the text widget which will be updated in new *
*                               data is received.                            *
*                                                                            *
*                                                                            *
* ORIGINAL AUTHOR AND IDENTIFICATION:                                        *
*                                                                            *
*      Mark D. Collier - Software Engineering Section                        *
*                        Data System Science and Technology Department       *
*                        Automation and Data Systems Division                *
*                        Southwest Research Institute                        *
*****************************************************************************/


#include <X11/Intrinsic.h>
#include <pm_constants.h>
#include <sys/ipc.h>
#include <errno.h>
#include <hisde.h>
#include <h_user_inter.h>
#include <h_talk.h>


extern  Widget  t_recv;

extern  int     pipe_num;


XtTimerCallbackProc tmr_recv ( client_data, id )

    caddr_t         client_data;    /* Character data passed to this callback function.
                                     * It is currently unused by this function.
                                     */

    XtIntervalId    id;             /* Identifies the timer which caused this function to
                                     * be activated.
                                     */
{
    register int    len;            /* Set to the number of bytes returned from the
                                     * pipe manager.
                                     */

    char            new_recv[ SIZE_PIPE_BUF + 1 ],
                                    /* Buffer into which the received data is saved.
                                     */
                    *temp;          /* Used to get the current string (length).
```

```
                                        */

    extern int      errno;              /* UNIX error message. Needed to determine pipe
                                         * manager errors.
                                         */

/*
 *  Determine if a connection has already been opened.
 */

    if ( pipe_num != -1 ) {

/*
 *      Attempt to read from the connected workstation.  Note that this call will return
 *      if no data is pending (it will not block).  If the call completes successfully,
 *      check if the string is terminated by a newline.  If not, add a newline to the
 *      end of the string.  Note that in either case, the string will be NULL terminated,
 *      as (h_read) will not perform this automatically.  Note also that the newline is
 *      added to improve readability of messages (which may not include newlines).
 */

        if ( ( len = h_read ( pipe_num, new_recv, SIZE_PIPE_BUF, IPC_NOWAIT ) ) != -1 ) {
            if ( new_recv[ len - 1 ] != NEWLINE )
                new_recv[ len++ ] = NEWLINE;
            new_recv[ len ] = NULL;

/*
 *          Get the current receive buffer length and use to set cursor to last position.
 *          Add the new text.
 */

            temp = get_text_widget ( t_recv );
            XmTextSetInsertionPosition ( t_recv, strlen ( temp ) );
            XtFree ( temp );

            insert_text_widget ( t_recv, new_recv );

/*
 *      Otherwise, determine if the remote pipe manager has opened or closed the con-
 *      nection on the remote end.  In either case, output a warning to the system
 *      message window.
 */

        } else if ( errno == OPEN_PIPE_REQ )
            display_message ( MSG_WARNING, "Remote pipe manager has opened connection" );
        else if ( errno == CLOSE_PIPE_REQ )
            display_message ( MSG_WARNING, "Remote pipe manager has closed connection" );

/*
 *      Otherwise, if an error occurred (and is not simply a 'no message available'
 *      error), output a warning to the system message window.
 */

        else if ( errno != ENOMSG )
            display_message ( MSG_ERROR, "Remote pipe manager encountered an error" );

    }

/*
 *  Reinitialize the timer to cause this function to be executed at the next interval.
 *  It is necessary to perform this each time as the interval is deinitialized after
 *  it completes (indicated by execution of this function).
 */
```

```
        XtAddTimeOut ( TIMER_VALUE, tmr_recv, NULL );
    }
```

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

CONTINUATION OF RESEARCH IN SOFTWARE FOR SPACE OPERATIONS SUPPORT

DATA TABLE DISPLAY WIDGET

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

SwRI has developed a prototype widget which provides a mechanism for fast, efficient display of multiple textual values. The purpose of this prototype is to allow an application programmer to textually display a buffer of dynamic data values where each individual value will be updated as necessary. Any display of data will reflect the entire buffer of data at a given time. This prototype supports several different layout styles for both values and labels and provides minimum and maximum limit checks for each data value.

The means for implementing this prototype mechanism is the creation of an X Windows widget. This widget has been created following the standard practices used in the creation of all toolkit widgets under X Windows.

2.0 RESEARCH GOALS

The goal of this research effort was to provide a mechanism by which a user could display large amounts of dynamic textual information in an easy and efficient manner using X Windows. This goal was attained by:

- Creating a widget which is compatible with the standard format for X Windows widgets.
- Providing the user with widget resources which will allow the information to be displayed in a variety of layout styles.
- Using the prototype as a basis for the text display in the converted X Windows/Motif-based Display Manager.

3.0 RESEARCH DETAILS

The prototype widget was developed in X Windows using only Xlib (low level library) and Xt (standard toolkit) functions. There are not any references to functions and definitions from other toolkits.

The data table display widget, or Dbdata widget, will provide resources for the application programmer to perform the following functions:

- Allow the application to update the actual data values.
- Specify minimum and maximum limits for each data value.
- Specify whether the buffer of data should be displayed in row-major or column-major order.
- Specify the table labels and their orientation: single label, row/column labels, corresponding labels, or no labels.
- Specify the foreground and background colors for labels, data values within their specified range, data values below the minimum limit, and data values above the maximum limit.
- Allow the application to specify a callback function to be called when any data value exceeds its specified limit.
- Specify the font to be used in the display of values and labels.
- Specify the time interval to be used when checking for changed values.

This widget implementation also provides for initialization, resizing, exposing, destroying, and setting resource values for a particular instance of the widget.

The main benefits of this widget are the speed at which updates appear on the display and the reduction in data space used by a single widget as compared to multiple instances of a widget for each value and label.

3.1 Implementation Overview

The Dbdata widget is a subclass of the Core Widget Class and is instantiated by a call to the standard X Windows create function, XtCreateWidget. The Class name for the Dbdata widget is XeDbdata and the class pointer is XeDbDataWidgetClass.

The Dbdata widget includes methods to optimally handle initialization, a resized widget, exposed and unexposed portions of a widget, a destroyed widget, and reset resource values. These methods were developed using the standard widget creation routines indicated in the class record for the widget.

In order to create a widget which interacts correctly with the X Intrinsics there needs to be three files created:

- Dbdata.h - The public header file which defines all Resource names special to this widget, as well as any structures which must be defined for the widget to be used by the application instantiating the widget.
- DbdataP.h - The private header file which defines the structures necessary to create an instance of the Rtdata widget. The contents of this header file are for use by the widget source code.
- Dbdata.c - The source file containing the initialization of all resources, and the code to initialize, expose, resize, destroy, update, manage, and reset resources for a particular instance of a widget.

This widget is designed to receive the address of a buffer which contains pointers to the data values, limits, colors, and display attributes. The widget will display the data in the layout specified by the resources using the specified labels, limits, and colors. Once the initial buffer is displayed, a TimerCallback will be used to check the values of the current buffer with the displayed buffer. The Timer will be set to the interval indicated by the user. If any data values have changed, those individual data strings will be modified on the display.

3.1.1 Functional Description of a Data Table Widget

The RtData widget is a special-purpose manager capable of displaying a buffer of data values using text images and the Xlib XDrawImageString function. It will support several different layout styles and provide minimum and maximum limit checks for data values along with associated actions for deviations.

The layout is controlled by the various layout resources set by the application. It can be configured to lay out its children in either rows or columns. In addition, the application can specify whether the children should be packed tightly together (not into organized rows and columns), or whether each image should be placed in an identically-sized box and placed symmetrically, or whether specific layout by x and y positions should be done. Labels will be allowed in four orientations: one label for the entire buffer, row and column labels, one-to-one corresponding labels, or no labels.

3.1.2 Name

XeDbData - the DbData widget class.

3.1.3 Include Files

#include <Xe/DbData.h>

3.1.4 Classes

The DbData widget class inherits behavior and resources from the Core widget class.

The class pointer is XeDbDataWidgetClass.

The class name is XeDbData.

3.1.5 Data Table Widget Functions

The following functions are available to create a DbData widget:

- Widget XtCreateWidget (name, xeDbDataWidgetClass, parent, arglist, argcount)
- Widget XeCreateDbData (parent, name, arglist, argcount)

Both functions create an instance of a DbData widget and return the associated widget ID. XtCreateWidget() is the standard X Toolkit create function. XeCreateDbData() is the DbData-specific create function.

3.1.6 New Resources

In order to provide the programmer with a somewhat simple interface to the widget, the widget will be designed with resources similar to the Motif RowColumn widget, as well as some resources which are specific to the data table display widget.

Those widget resources which will be accessible by the user to specify data are:

- Font - Indicates the type of font to be used for all text displayed in one widget instance. The default is FIXED.
- LabForeground - Pixel value indicating the foreground for a label. The default is WHITE.
- LabBackground - Pixel value indicating the background for a label. The default is BLUE.
- DefForeground - Pixel value indicating the default foreground display color. The default is BLACK.
- DefBackground - Pixel value indicating the default background display color. The default is WHITE.
- Interval - Integer value indicating the number of milliseconds to wait when checking data values. The default is 1000 (1 second).
- MinCallback - Callback for minimum limit exception. This function is called with a linked list of indices for the data values which have exceeded their minimum limits.
- MaxCallback - Callback for maximum limit exception. This function is called with a linked list of indices for the data values which have exceeded their maximum limits.
- Orientation - Will determine whether the data values are laid out in row major order, column major order, or user-specified. The available values are XeNO_ORIENTA-

TION for user-specified, XeVERTICAL for column major and XeHORIZONTAL for row major. The default is XeNO_ORIENTATION.

- Packing - Will specify how to pack the data values. The data values may be packed tightly along the major dimension, resized to the size of the largest value and aligned in columns, or not packed at all, but positioned according to the user-specified (x,y) positions. The available values are XePACK_TIGHT, XeNO_PACK_COLUMN, and XeNO_PACKING. The default is XeNO_PACK_COLUMN.
- NumColumns - If packing is set to PACK_COLUMN, this resource determines the maximum number of rows (for horizontal orientation) or columns (for vertical orientation. Default is the maximum number which will fit packed tightly.
- LabelOrientation - Will specify how to label the data values. The data values may be labeled by one label, row and column labels, individual labels, or no labels. The available values are XeTABLE_LABEL, XeCORR_LABEL, XeROW_COLUMN, and XeNO_LABEL_ORIENT. The default is XeNO_LABEL_ORIENT.
- NumValues - Will indicate the number of data values to be displayed by the widget. The maximum number allowed is 750. Mandatory.
- Values - Will point to a buffer of pointers to the data values, associated limits, and X and Y coordinates. Mandatory. The structure for this resource is defined in the DbElement structure in the Dbdata.h file.
- Labels - Buffer of string values to be used as labels for the associated data values and their X and Y coordinates.

Currently there are a number of screen types which may be assigned for the display of data. The available types are floating point, signed integer, scientific notation, hexadecimal, octal, binary, character, multitext, unsigned integer, and a number of different time formats. These formats are defined in the Dbdata.h file with the constant values for each display type. This attribute is set in the Type element of the DbElement structure.

4.0 RESEARCH CONCLUSIONS

Conclusions drawn from this research effort include finding that the creation of a widget to display and manage large amounts of data can be done in a useful and efficient manner using the standard procedures in X Windows. During the early development of the widget a program was successfully written to display a window containing four of the data table display widgets. Each widget contained from 50 to 200 data values and were displayed with different data and label orientations. This test program successfully updated all of the data values within a widget once per second.

During the integration of this widget into the X Windows/Motif-based Display Manager, it was found that the Dbdata widget did not replace widgets for each individual label and data value. With the current design of Display Manager the widget was only used to display data values and the labels were displayed by using the X Library and Toolkit functions. In this scenario, all of the data values were displayed under the control of one data table display widget. It would be more beneficial to the performance of Display Manager to use the widget to display both the labels and the values, but time did not allow this type of a redesign effort. By using the widget and its associated methods, X events are automatically handled and the overhead associated with managing the

resizing, redrawing, and destroying of display areas in the program would be eliminated and carried out in the widget methods.

5.0 ATTACHMENTS

The following pages contain the actual code for the data table display widget. The code and related files which are present include:

- Dbdata.h - The public header file which defines all Resource names special to this widget, as well as any structures which must be defined for the widget to be used by the application instantiating the widget.
- DbdataP.h - The private header file which defines the structures necessary to create an instance of the Rtdata widget. The contents of this header file are for use by the widget source code.
- Dbdata.c - The source file containing the initialization of all resources, and the code to initialize, expose, resize, destroy, update, manage, and reset resources for a particular instance of a widget.

ATTACHMENT 1 - Public Header File

```
/*****************************************************************************
 * MODULE NAME: Dbdata.h
 *
 *  Public header file for dbdata Widget Class
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Nancy L. Martin - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#ifndef DBDATA_H
#define DBDATA_H

extern  WidgetClass XedbdataWidgetClass;

typedef struct _XeDbdataClassRec *XeDbdataWidgetClass;

typedef struct _XeDbdataRec *XeDbdataWidget;
/*
 *  Define the resource strings for the Dbdata widget.
 */
#define XtNminimum          "minimum"
#define XtNmaximum          "maximum"
#define XtNlabForeground    "labForeground"
#define XtNlabBackground    "labBackground"
#define XtNdefForeground    "defForeground"
#define XtNdefBackground    "defBackground"
#define XtNinterval         "interval"
#define XtNnumColumns       "numColumns"
#define XtNnumValues        "numValues"
#define XtNvalues           "values"
#define XtNlabels           "labels"
#define XtNmaxCallback      "maxCallback"
#define XtNminCallback      "minCallback"
#define XtNorientation      "orientation"
#define XtNpacking          "packing"
#define XtNlabelOrientation "labelOrientation"

#define XtCNumColumns       "NumColumns"
#define XtCNumValues        "NumValues"
#define XtCValues           "Values"
#define XtCLabels           "Labels"
#define XtCOrientation      "Orientation"
#define XtCPacking          "Packing"
#define XtCLabelOrientation "LabelOrientation"

#define XtROrientation      "Orientation"
#define XtRPacking          "Packing"
#define XtRLabelOrientation "LabelOrientation"
/*
 *  Define the values which may be used for the various
 *  orientation settings.
 */
#define XeNO_ORIENTATION    0
#define XeVERTICAL          1
#define XeHORIZONTAL        2
#define XeNO_PACKING        0
#define XeNO_PACK_COLUMN    1
#define XePACK_TIGHT        2
```

```
#define XeNO_LABEL_ORIENT    0
#define XeTABLE_LABEL        1
#define XeROW_COLUMN         2
#define XeCORR_LABEL         3

#define XeFLOAT              1
#define XeSIGNED             2
#define XeSCIENTIFIC         3
#define XeHEXADECIMAL        4
#define XeOCTAL              5
#define XeBINARY             6
#define XeCHARACTER          8
#define XeMULTITEXT          9
#define XeTIME10             10
#define XeTIME11             11
#define XeTIME12             12
#define XeTIME13             13
#define XeTIME15             15
#define XeTIME16             16
#define XeTIME17             17
#define XeTIME18             18
#define XeTIME19             19
#define XeTIME20             20
#define XeUNSIGNED           21

#define XeYES                    1
#define XeNO                     0

#define XeMISSING_DATA       0x80000000  /* Missing data mask for status bit 0   */
#define XeDEAD_DATA          0x40000000  /* Dead data mask for status bit 1      */
#define XeOFF_SCALE_HIGH     0x20000000  /* Off scale high mask for status bit 2 */
#define XeOFF_SCALE_LOW      0x10000000  /* Off scale low mask for status bit 3  */
#define XeSTATIC_DATA        0x08000000  /* Static data mask for status bit 4    */
#define XeLIMIT_HIGH         0x04000000  /* Limit high mask for status bit 5     */
#define XeLIMIT_LOW          0x02000000  /* Limit low mask for status bit 6      */
#define XeHOMOG_DATA         0x01000000  /* Homog. data mask for status bit 7    */
#define XeUNKNN_DATA         0x01000000  /* ?????? data mask for status bit 8    */
#define XeCRITICAL_HIGH      0x00400000  /* Critical high mask for status bit 9  */
#define XeCRITICAL_LOW       0x00200000  /* Critical low mask for status bit 10  */
#define XeUNAVAIL_DATA       0x00000040  /* Unavail. data mask for status bit 25 */
/*
 *  Define the structures to be used as the buffers containing the data
 *  values, labels, and limits.
 */
typedef struct  _element_Struct {
    char        *Value;
    int         Type;
    int         Year;
    int         YearCat;
    char        Attrib;
    int         Length;
    int         Width;
    int         Precision;
    int         JustFlag;
    int         DispStat;
    int         StatFlag;
    Position    X;
    Position    Y;
    double      MinLimit;
    double      MaxLimit;
    long        LowColor;
    long        HiColor;
    double      CrMinLimit;
    double      CrMaxLimit;
```

```
    long        CrLColor;
    long        CrHColor;
    long        NomColor;
    long        StaColor;
    long        OvrColor;
    long        DeadColor;
    XFontStruct *DefFont;
}   DbElement, *DbElementPtr;

typedef struct  _label_struct {
    char        *Label;
    Position    LabelX;
    Position    LabelY;
}   DbLabel, *DbLabelPtr;

/*
 *  Define the structure to be used as the callback structure.
 */
struct list {
    int             index;
    struct  list    *next;
};

typedef struct {
    struct  list    *indices;
}  xedbdataCallbackStruct;

#endif DBDATA_H
```

ATTACHMENT 2- Private Header File

```
/*****************************************************************************
 * MODULE NAME: Dbdata.h
 *
 *  Private header file for dbdata Widget Class
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *  Nancy L. Martin - Software Engineering Section
 *                    Data Systems Department
 *                    Automation and Data Systems Division
 *                    Southwest Research Institute
 *****************************************************************************/


#ifndef DBDATAP_H
#define DBDATAP_H

#include "Dbdata.h"
/*
 *  Define the maximum number of data values which will be allowed
 *  within one instance of an Dbdata widget.
 */
#define MAXVAL              750
#define FLEN                11
/*
 *  Define a structure containing the Dbdata widget's contribution to the
 *  class record.  In this case the Dbdata widget does not have anything
 *  to add to the class record.
 */
typedef struct _XeDbdataClassPart {
    int     ignore;
}   XeDbdataClassPart;
/*
 *  Define the Dbdata widget's full class record with the core class part
 *  and the dummy Dbdata part.'
 */
typedef struct _XeDbdataClassRec {
    CoreClassPart   core_class;
    XeDbdataClassPart   dbdata_class;
}   XeDbdataClassRec;

extern XeDbdataClassRec XedbdataClassRec;
/*
 *  Define the instance record of the Dbdata widget class.  This structure
 *  defines the resources added by the Dbdata widget.
 */
typedef struct _XeDbdataPart {
    short           Orientation;        /*  Data orientation            */
    short           Packing;            /*  Data packing                */
    int             NumColumns;         /*  Number of rows or columns,  */
                                        /*    if orientation set        */
    short           LabelOrientation;   /*  Label orientation           */
    Pixel           LabForeground;      /*  Label foreground color      */
    Pixel           LabBackground;      /*  Label background color      */
    GC              LabGC;              /*  GC created for labels       */
    Pixel           DefForeground;      /*  Label foreground color      */
    Pixel           DefBackground;      /*  Label background color      */
    GC              NomGC;              /*  GC created for nominal      */
                                        /*    values                    */
    GC              StaGC;              /*  GC created for static       */
                                        /*    values                    */
    GC              OvrGC;              /*  GC created for override     */
                                        /*    values                    */
```

```
    GC              DeadGC;          /*  GC created for dead          */
                                     /*    values                     */
    GC              LowGC;           /*  GC created for low           */
                                     /*    values                     */
    GC              HiGC;            /*  GC created for high          */
                                     /*    values                     */
    GC              CrLGC;           /*  GC created for critical      */
                                     /*    low values                 */
    GC              CrHGC;           /*  GC created for critical      */
                                     /*    high values                */
    XFontStruct     *DefFont;        /*  Font used for display        */
    DbElementPtr    Elements;        /*  Pointer to data, limits,     */
                                     /*    and positions              */
    DbLabelPtr      Labels;          /*  Pointer to labels and        */
                                     /*    positions                  */
    int             NumValues;       /*  Number of data values        */
    Position        MidX;            /*  Coordinates for midpoint     */
    Position        MidY;            /*    of window                  */
    int             NumLabels;       /*  Number of labels based       */
                                     /*    on label orientation       */
    int             FontHeight;      /*  Maximum height of selected   */
                                     /*    font                       */
    int             MaxWidth;        /*  Maximum width of all values  */
    int             LabelWidth;      /*  Maximum width of all values  */
    char            *Label[MAXVAL];  /*  Displayed label              */
    char*           OldValue[MAXVAL]; /*  Currently displayed values  */
    char            Format[MAXVAL][FLEN]; /*  Format for each value    */
    long            Interval;        /*  Specified timer interval     */
    XtIntervalId    Id;              /*  Id for timer                 */
    XtCallbackList  MinCB;           /*  Callback list for minimum    */
                                     /*    values                     */
    XtCallbackList  MaxCB;           /*  Callback list for maximum    */
                                     /*    values                     */
    Boolean         Redisplay;       /*  Indicates whether the        */
                                     /*    is being redisplayed       */
} XeDbdataPart;
/*
 *  Define the Dbdata widget's instance record by combining the CorePart as
 *  defined by the Core widget class and the XeDbdataPart.
 */
typedef struct _XeDbdataRec {
    CorePart        core;
    XeDbdataPart    dbdata;
} XeDbdataRec;

union   data_types  {
        double  ddata;
        float   sfdata[2];
        long    sldata[2];
        short   ssdata[4];
        unsigned long   ldata[2];
        unsigned long   uldata[2];
        unsigned short  usdata[4];
};
#endif DBDATAP_H
```

ATTACHMENT 3 - Source Code

```
/*****************************************************************************
 * FILE NAME:  Dbdata.c  The Dbdata Widget Methods
 *
 *  This is the source code which contains the declaration and static
 *  initialization of the Dbdata widget's class record, and also contains
 *  the widget's methods.
 *
 * ORIGINAL AUTHOR:
 *
 *      Nancy L. Martin - Software Engineering Section
 *                        Data Systems Department
 *                        Automation and Data Systems Division
 *                        Southwest Research Institute
 *
 *****************************************************************************/


#include <stdio.h>
#include <math.h>
#include <X11/IntrinsicP.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/CoreP.h>
#include <DbdataP.h>
#include <Dbdata.h>

#define MAXLENGTH       256
#define MAX(a, b)       ((a>b)?a:b)

static void Initialize();
static void Redisplay();
static void Resize();
static void Destroy();
static Boolean SetValues();

static unsigned char resource_packing       = XeNO_PACK_COLUMN;
static unsigned char resource_orient        = XeNO_ORIENTATION;
static unsigned char resource_label_orient  = XeNO_LABEL_ORIENT;
static unsigned int resource_values         = 0;
static unsigned int resource_labels         = 0;
static unsigned short resource_columns      = 1;
/*
 *  Initialize the resource list used by the resource manager for
 *  the Dbdata widget.  The resources are automatically stored in
 *  the appropriate fields of the instance record.
 */
static  XtResource  resources[]  = {
    {   XtNfont,
        XtCFont,
        XtRFontStruct,
        sizeof (XFontStruct*),
        XtOffset(XeDbdataWidget, dbdata.DefFont),
        XtRString,
        "Fixed"
    },
    {   XtNlabForeground,
        XtCForeground,
        XtRPixel,
        sizeof(Pixel),
        XtOffset(XeDbdataWidget, dbdata.LabForeground),
        XtRString,
        "white"
    },
    {   XtNlabBackground,
```

```
        XtCBackground,
        XtRPixel,
        sizeof(Pixel),
        XtOffset(XeDbdataWidget, dbdata.LabBackground),
        XtRString,
        "lightblue"
    },
    {   XtNdefForeground,
        XtCForeground,
        XtRPixel,
        sizeof(Pixel),
        XtOffset(XeDbdataWidget, dbdata.DefForeground),
        XtRString,
        "black"
    },
    {   XtNdefBackground,
        XtCBackground,
        XtRPixel,
        sizeof(Pixel),
        XtOffset(XeDbdataWidget, dbdata.DefBackground),
        XtRString,
        "white"
    },
    {   XtNinterval,
        XtCInterval,
        XtRInt,
        sizeof(long int),
        XtOffset(XeDbdataWidget, dbdata.Interval),
        XtRString,
        "1000",
    },
    {   XtNminCallback,
        XtCCallback,
        XtRCallback,
        sizeof(XtCallbackList),
        XtOffset(XeDbdataWidget, dbdata.MinCB),
        XtRCallback,
        (caddr_t)NULL
    },
    {   XtNmaxCallback,
        XtCCallback,
        XtRCallback,
        sizeof(XtCallbackList),
        XtOffset(XeDbdataWidget, dbdata.MaxCB),
        XtRCallback,
        (caddr_t)NULL
    },
    {   XtNorientation,
        XtCOrientation,
        XtROrientation,
        sizeof(short),
        XtOffset(XeDbdataWidget, dbdata.Orientation),
        XtROrientation,
        (caddr_t) &resource_orient
    },
    {   XtNpacking,
        XtCPacking,
        XtRPacking,
        sizeof(short),
        XtOffset(XeDbdataWidget, dbdata.Packing),
        XtRPacking,
        (caddr_t) &resource_packing
    },
    {   XtNnumColumns,
```

```
            XtCNumColumns,
            XtRShort,
            sizeof(int),
            XtOffset(XeDbdataWidget, dbdata.NumColumns),
            XtRShort,
            (caddr_t) &resource_columns
        },
        {   XtNlabelOrientation,
            XtCLabelOrientation,
            XtRLabelOrientation,
            sizeof(short),
            XtOffset(XeDbdataWidget, dbdata.LabelOrientation),
            XtRLabelOrientation,
            (caddr_t) &resource_label_orient
        },
        {   XtNnumValues,
            XtCNumValues,
            XtRInt,
            sizeof(int),
            XtOffset(XeDbdataWidget, dbdata.NumValues),
            XtRInt,
            (caddr_t) &resource_values
        },
        {   XtNvalues,
            XtCValues,
            XtRPointer,
            sizeof(int*),
            XtOffset(XeDbdataWidget, dbdata.Elements),
            XtRPointer,
            NULL
        },
        {   XtNlabels,
            XtCLabels,
            XtRPointer,
            sizeof(int*),
            XtOffset(XeDbdataWidget, dbdata.Labels),
            XtRPointer,
            NULL
        }
};
/*
 *  Define the contents of the Dbdata widget's class record, which is
 *  initialized at compile time by declaring the contents of the
 *  structure statically in the source code.
 */
XeDbdataClassRec    XedbdataClassRec = {
/*  Core class fields.                                                  */
    {
        (WidgetClass)&widgetClassRec,   /*  Superclass              */
        "Dbdata",                       /*  Class Name              */
        sizeof(XeDbdataRec),            /*  Widget Size             */
        NULL,                           /*  Class Initialize        */
        NULL,                           /*  Class Part Initialize   */
        FALSE,                          /*  Class Initialized       */
        Initialize,                     /*  Initialize Process      */
        NULL,                           /*  Initialize Hook         */
        XtInheritRealize,               /*  Realize Process         */
        NULL,                           /*  Actions List            */
        0,                              /*  Number of Actions       */
        resources,                      /*  Resources               */
        XtNumber(resources),            /*  Number of Resources     */
        NULL,                           /*  Resource Manager Class  */
        TRUE,                           /*  Compress Motion         */
        TRUE,                           /*  Compress Exposure       */
```

```
        TRUE,                                   /*  Compress Enter/Leave    */
        TRUE,                                   /*  Visible Interest        */
        Destroy,                                /*  Destroy Process         */
        Resize,                                 /*  Resize Process          */
        Redisplay,                              /*  Expose Process          */
        SetValues,                              /*  Set Values              */
        NULL,                                   /*  Set Values Hook         */
        XtInheritSetValuesAlmost,               /*  Set Values Almost       */
        NULL,                                   /*  Get Values Hook         */
        NULL,                                   /*  Accept Focus            */
        XtVersion,                              /*  Version                 */
        NULL,                                   /*  Callback Private        */
        NULL,                                   /*  Translation Table       */
        NULL,                                   /*  Query Geometry          */
        NULL,                                   /*  Display Accelerator     */
        NULL,                                   /*  Extension               */
    },
/*  Dbdata class fields.                                                    */
    {   0,                                      /*  Ignore                  */
    }
};

WidgetClass XedbdataWidgetClass = (WidgetClass) &XedbdataClassRec;
/*
 *  Define the procedure to be called at a specified interval to
 *  check for changed data values.
 */
XtTimerCallbackProc check_data();
/*
 *  Set up the structure to be used for extracting data.
 */
union   data_types  Data;
/*****************************************************************************
 *  METHOD NAME:  Initialize
 *
 *   This method is used to initialize the instance record at run
 *      run time.
 *
 *****************************************************************************/
static void Initialize (request, new)
    XeDbdataWidget  request,
                    new;
{
    int     ascent,
            descent,
            dir,
            n,
            width;
    XCharStruct char_info;
    XGCValues   crl_gcv,
                crh_gcv,
                dead_gcv,
                hi_gcv,
                lab_gcv,
                low_gcv,
                nom_gcv,
                ovr_gcv,
                sta_gcv;
    int         mask = GCForeground;
    int         labmask = GCForeground | GCBackground;
    caddr_t     client_data;
    Display     *display = XtDisplay(new);
    Window      root = RootWindowOfScreen(XtScreen(new));
```

```
    if ( request->core.width == 0 )
        new->core.width = 100;
    if ( request->core.height == 0 )
        new->core.height = 100;

    for ( n = 0; n < request->dbdata.NumValues; n++ ) {
        if ( request->dbdata.Elements[n].MinLimit >
             request->dbdata.Elements[n].MaxLimit) {
            request->dbdata.Elements[n].MinLimit = 0;
            request->dbdata.Elements[n].MaxLimit = 100;
            request->dbdata.OldValue[n] = request->dbdata.Elements[n].Value - 1;
        }
        if ( request->dbdata.Elements[n].CrMinLimit >
             request->dbdata.Elements[n].CrMaxLimit) {
            request->dbdata.Elements[n].CrMinLimit = 0;
            request->dbdata.Elements[n].CrMaxLimit = 100;
        }
    }
/*
 *  Create the GCs for possibly different value status.
 */
    crl_gcv.foreground = request->dbdata.DefForeground;
    new->dbdata.CrLGC = XCreateGC (display, root, mask, &crl_gcv);

    crh_gcv.foreground = request->dbdata.DefForeground;
    new->dbdata.CrHGC = XCreateGC (display, root, mask, &crh_gcv);

    dead_gcv.foreground = request->dbdata.DefForeground;
    new->dbdata.DeadGC = XCreateGC (display, root, mask, &dead_gcv);

    hi_gcv.foreground = request->dbdata.DefForeground;
    new->dbdata.HiGC = XCreateGC (display, root, mask, &hi_gcv);

    lab_gcv.foreground = new->dbdata.LabForeground;
    lab_gcv.background = new->dbdata.LabBackground;
    new->dbdata.LabGC = XCreateGC (display, root, labmask, &lab_gcv);

    low_gcv.foreground = request->dbdata.DefForeground;
    new->dbdata.LowGC = XCreateGC (display, root, mask, &low_gcv);

    nom_gcv.foreground = request->dbdata.DefForeground;
    new->dbdata.NomGC = XCreateGC (display, root, mask, &nom_gcv);

    ovr_gcv.foreground = request->dbdata.DefForeground;
    new->dbdata.OvrGC = XCreateGC (display, root, mask, &ovr_gcv);

    sta_gcv.foreground = request->dbdata.DefForeground;
    new->dbdata.StaGC = XCreateGC (display, root, mask, &sta_gcv);
/*
 *  If there are labels, calculate the maximum label width.  This loop
 *  will not be executed, if there are not any labels.
 */
    width = 0;
    for ( n = 0; n < new->dbdata.NumLabels; n++ ) {
        if ( new->dbdata.Labels[n].Label )
            width = MAX(width, strlen(new->dbdata.Labels[n].Label));
    }
    new->dbdata.LabelWidth = 0;
    if ( new->dbdata.LabelOrientation != XeNO_LABEL_ORIENT ) {
        for ( n = 0; n < new->dbdata.NumLabels; n++ ) {
            if ( new->dbdata.Labels[n].Label) {
                sprintf (new->dbdata.Label[n], "%*s", width,
                         new->dbdata.Labels[n].Label);
                XTextExtents (new->dbdata.DefFont, new->dbdata.Label[n],
```

```
                                width, &dir,
                                &ascent, &descent, &char_info);
                        new->dbdata.LabelWidth =
                                MAX(new->dbdata.LabelWidth, char_info.width);
                }
            }
        }
/*
 *  Now call the Resize method to set up the display.
 */
    Resize (new);
/*
 *  Start the timer for checking the data values.
 */
    client_data = (caddr_t)new;
    new->dbdata.Id = XtAddTimeOut (new->dbdata.Interval, check_data, client_data);
}
/*****************************************************************************
 *  METHOD NAME:  Destroy
 *
 *   This method is used to clean up any resources the Dbdata widget
 *      has created.
 *
 *****************************************************************************/
static void Destroy (w)
    XeDbdataWidget  w;
{
/*
 *  Free all of the GCs allocated for this widget.
 */
    XFreeGC (w, w->dbdata.LabGC);
    XFreeGC (w, w->dbdata.NomGC);
    XFreeGC (w, w->dbdata.StaGC);
    XFreeGC (w, w->dbdata.OvrGC);
    XFreeGC (w, w->dbdata.DeadGC);
    XFreeGC (w, w->dbdata.LowGC);
    XFreeGC (w, w->dbdata.HiGC);
    XFreeGC (w, w->dbdata.CrLGC);
    XFreeGC (w, w->dbdata.CrHGC);
/*
 *  Remove the callbacks setup for this widget.
 */
    XtRemoveAllCallbacks (w, XtNminCallback, w->dbdata.MinCB);
    XtRemoveAllCallbacks (w, XtNmaxCallback, w->dbdata.MaxCB);
}
/*****************************************************************************
 *  METHOD NAME:  Destroy
 *
 *   This method is used to examine the members of the widget structure
 *      and recalculate any derived data that is dependent on the
 *      configuration of the widget's window.
 *
 *****************************************************************************/
static void Resize (w)
    XeDbdataWidget  w;
{
    int     ascent,
            delta_x,
            delta_y,
            descent,
            digit,
            dir,
            row,
            columns,
```

```
            label_rows,
            label_columns,
            rows,
            label_width,
            n,
            width;
    unsigned int    idata;
    int     ival;
    long    uval;
    unsigned long   days,
                    hours,
                    milliseconds,
                    minutes,
                    seconds;
    char    length[3],
            lead[5],
            precision[4],
            convert[2];
    double  num_sqrt,
            real_hours,
            real_min,
            real_sec,
            val_sqrt;
    Position        startx,
                    starty;
    XCharStruct char_info;
    char    hold[MAXLENGTH],
            bitstr[MAXLENGTH];
    void    get_bin();

/*
 *  Calculate the maximum width of the data values.
 */
    w->dbdata.MaxWidth = 0;
    for ( n = 0; n < w->dbdata.NumValues; n++ ) {
        if ( w->dbdata.Elements[n].Width > 0 )
            sprintf ( length, "%d", w->dbdata.Elements[n].Width );
        else
            sprintf (length, "");
        if ( w->dbdata.Elements[n].Precision >= 0 )
            sprintf ( precision, ".%d", w->dbdata.Elements[n].Precision );
        else
            sprintf (precision, "");
/*
 *  Set up the format for the selected screen type.
 */
        if ( w->dbdata.Elements[n].Type == XeFLOAT) {
            sprintf ( convert, "f" );
            sprintf ( lead, "%%" );
        } else if ( w->dbdata.Elements[n].Type == XeSIGNED) {
            sprintf ( convert, "d" );
            sprintf ( lead, "%%" );
        } else if ( w->dbdata.Elements[n].Type == XeSCIENTIFIC) {
            sprintf ( precision, ".%d", w->dbdata.Elements[n].Precision-4 );
            sprintf ( convert, "e" );
            sprintf ( lead, "%%" );
        } else if ( w->dbdata.Elements[n].Type == XeHEXADECIMAL) {
            sprintf ( convert, "x" );
            sprintf ( lead, "0x%%0" );
            sprintf ( precision, ".%d", w->dbdata.Elements[n].Width );
        } else if ( w->dbdata.Elements[n].Type == XeOCTAL) {
            sprintf ( lead, "0%%0" );
            sprintf ( precision, ".%d", w->dbdata.Elements[n].Width );
            sprintf ( convert, "o" );
```

```
} else if (( w->dbdata.Elements[n].Type == XeBINARY ) ||
        ( w->dbdata.Elements[n].Type == XeCHARACTER ) ||
        ( w->dbdata.Elements[n].Type == XeMULTITEXT ) ||
        ( w->dbdata.Elements[n].Type == XeTIME10 ) ||
        ( w->dbdata.Elements[n].Type == XeTIME11 ) ||
        ( w->dbdata.Elements[n].Type == XeTIME12 ) ||
        ( w->dbdata.Elements[n].Type == XeTIME13 ) ||
        ( w->dbdata.Elements[n].Type == XeTIME15 ) ||
        ( w->dbdata.Elements[n].Type == XeTIME16 ) ||
        ( w->dbdata.Elements[n].Type == XeTIME17 ) ||
        ( w->dbdata.Elements[n].Type == XeTIME18 ) ||
        ( w->dbdata.Elements[n].Type == XeTIME19 ) ||
        ( w->dbdata.Elements[n].Type == XeTIME20 )) {
    sprintf ( convert, "s" );
    sprintf ( lead, "%%" );
} else if ( w->dbdata.Elements[n].Type == XeUNSIGNED) {
    sprintf ( convert, "u" );
    sprintf ( lead, "%%" );
}
sprintf ( w->dbdata.Format[n], "%s%s%s%s", lead, length, precision, convert );

switch ( w->dbdata.Elements[n].Attrib ) {
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 ( w->dbdata.Elements[n].Type ) {

    case XeFLOAT:
        if ( w->dbdata.Elements[n].Length <= 32 ) {
            Data.sfdata[0] = *(float*)w->dbdata.Elements[n].Value;
            sprintf (hold, w->dbdata.Format[n], Data.sfdata[0]);
        } else {
            Data.ddata = *(double*)w->dbdata.Elements[n].Value;
            sprintf (hold, w->dbdata.Format[n], Data.ddata);
        }
        break;
    case XeSIGNED:
        if (( w->dbdata.Elements[n].Attrib == 'D' ) ||
            ( w->dbdata.Elements[n].Attrib == 19 )) {
            Data.ddata = *(double*)w->dbdata.Elements[n].Value;
            if (( Data.ddata < 2147483647.0 ) && (Data.ddata > -2147483648.0 ))
                digit = Data.ddata;
            else
                digit = 2147483647;
            sprintf (hold, w->dbdata.Format[n], digit);
        } else {
            if ( w->dbdata.Elements[n].Length <= 32 ) {
                Data.sldata[0] = *(long*)w->dbdata.Elements[n].Value;
                sprintf (hold, w->dbdata.Format[n], Data.sldata[0]);
            } else {
                Data.ddata = *(double*)w->dbdata.Elements[n].Value;
                sprintf (hold, w->dbdata.Format[n], Data.ddata);
            }
        }
```

```
        break;
    case XeUNSIGNED:
        if (( w->dbdata.Elements[n].Attrib == 'D' ) ||
            ( w->dbdata.Elements[n].Attrib == 19 )) {
            Data.ddata = *(double*)w->dbdata.Elements[n].Value;
            if (( Data.ddata < 2147483647.0 ) && (Data.ddata > -2147483648.0 ))
                idata = Data.ddata;
            else
                idata = 2147483647;
            sprintf (hold, w->dbdata.Format[n], idata);
        } else {
            if ( w->dbdata.Elements[n].Length <= 32 ) {
                Data.uldata[0] = *(unsigned long*)w->dbdata.Elements[n].Value;
                sprintf (hold, w->dbdata.Format[n], Data.uldata[0]);
            } else {
                Data.ddata = *(double*)w->dbdata.Elements[n].Value;
                sprintf (hold, w->dbdata.Format[n], Data.ddata);
            }
        }
        break;
    case XeSCIENTIFIC:
        if ( w->dbdata.Elements[n].Length <= 32 ) {
            if ( w->dbdata.Elements[n].StatFlag != 0 ) {
                Data.sldata[0] = *(long*)w->dbdata.Elements[n].Value;
                sprintf (hold, "%*.*E", w->dbdata.Elements[n].Width,
                            w->dbdata.Elements[n].Precision-5, Data.sldata[0]);
            } else {
                Data.sldata[0] = *(long*)w->dbdata.Elements[n].Value;
                sprintf (hold, w->dbdata.Format[n], Data.sldata[0]);
            }
        } else {
            if ( w->dbdata.Elements[n].StatFlag != 0 ) {
                Data.ddata = *(double*)w->dbdata.Elements[n].Value;
                sprintf (hold, "%*.*E", w->dbdata.Elements[n].Width,
                            w->dbdata.Elements[n].Precision-5, Data.ddata);
            } else {
                Data.ddata = *(double*)w->dbdata.Elements[n].Value;
                sprintf (hold, w->dbdata.Format[n], Data.ddata);
            }
        }
        break;
    case XeHEXADECIMAL:
        Data.ddata = *(double*)w->dbdata.Elements[n].Value;
        sprintf (hold, w->dbdata.Format[n], Data.ddata);
        break;
    case XeOCTAL:
        Data.ddata = *(double*)w->dbdata.Elements[n].Value;
        sprintf (hold, w->dbdata.Format[n], Data.ddata);
        break;
    case XeBINARY:
        Data.ldata[0] = *(unsigned long*)w->dbdata.Elements[n].Value;
        get_bin ( Data.ldata, w->dbdata.Elements[n].Width, bitstr );
        sprintf (hold, w->dbdata.Format[n], bitstr);
        break;
    case XeMULTITEXT:
        strcpy (hold, w->dbdata.Elements[n].Value);
        break;
    case XeTIME10:              /* Tabular time (ddd:hh:mm:ss.sss)       */
    case XeTIME11:              /* Tabular time (yyyy:ddd:hh:mm:ss.sss)  */
    case XeTIME12:              /* Tabular time (yy:ddd:hh:mm:ss.sss)    */
    case XeTIME18:              /* Tabular time (ddd:hh:mm:ss.sss)       */
    case XeTIME19:              /* Tabular time (yyyy:ddd:hh:mm:ss.sss)  */
    case XeTIME20:              /*Tabular time (yyyy:ddd:hh:mm:ss.sss)   */
        if ( w->dbdata.Elements[n].Attrib == 'D' ) {
```

```
            Data.ddata = *(double*)w->dbdata.Elements[n].Value;
            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 ( w->dbdata.Elements[n].Type = XeTIME10 )
                sprintf (hold, "%03d:%02d:%02d:%02d.%03d",
                    days, hours, minutes, seconds, milliseconds );
            else if ( w->dbdata.Elements[n].Type = XeTIME11 )
                sprintf (hold, "%d:%03d:%02d:%02d:%02d.%03d",
                    w->dbdata.Elements[n].YearCat,
                    days, hours, minutes, seconds, milliseconds);
            else if ( w->dbdata.Elements[n].Type = XeTIME12 )
                sprintf (hold, "%d:%03d:%02d:%02d:%02d.%03d",
                    w->dbdata.Elements[n].Year,
                    days, hours, minutes, seconds, milliseconds);
            else if ( w->dbdata.Elements[n].Type = XeTIME18 )
                sprintf (hold, "%03d:%02d:%02d:%02d.%03d",
                    days, hours, minutes, seconds, milliseconds);
            else if ( w->dbdata.Elements[n].Type = XeTIME19 )
                sprintf (hold, "%d:%03d:%02d:%02d:%02d.%03d",
                    w->dbdata.Elements[n].YearCat,
                    days, hours, minutes, seconds, milliseconds);
            else if ( w->dbdata.Elements[n].Type = XeTIME20 )
                sprintf (hold, "%d:%03d:%02d:%02d:%02d.%03d",
                    w->dbdata.Elements[n].Year,
                    days, hours, minutes, seconds, milliseconds);
        } else {
            Data.usdata[0] = *(long*)w->dbdata.Elements[n].Value;
            Data.uldata[0] = *(unsigned long*)w->dbdata.Elements[n].Value;
            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 ( w->dbdata.Elements[n].Type = XeTIME10 )
                sprintf (hold, "%03d:%02d:%02d:%02d.%03d",
                    days, hours, minutes, seconds, milliseconds );
            else if ( w->dbdata.Elements[n].Type = XeTIME11 )
                sprintf (hold, "%d:%03d:%02d:%02d:%02d.%03d",
                    w->dbdata.Elements[n].YearCat,
                    days, hours, minutes, seconds, milliseconds);
            else if ( w->dbdata.Elements[n].Type = XeTIME12 )
                sprintf (hold, "%d:%03d:%02d:%02d:%02d.%03d",
                    w->dbdata.Elements[n].Year,
                    days, hours, minutes, seconds, milliseconds);
            else if ( w->dbdata.Elements[n].Type = XeTIME18 )
                sprintf (hold, "%03d:%02d:%02d:%02d.%03d",
                    days, hours, minutes, seconds, milliseconds);
            else if ( w->dbdata.Elements[n].Type = XeTIME19 )
                sprintf (hold, "%d:%03d:%02d:%02d:%02d.%03d",
                    w->dbdata.Elements[n].YearCat,
                    days, hours, minutes, seconds, milliseconds);
            else if ( w->dbdata.Elements[n].Type = XeTIME20 )
                sprintf (hold, "%03d:%02d:%02d:%02d.%03d",
                    days, hours, minutes, seconds, milliseconds);
        }
        break;
    case XeTIME13:                          /* Tabular time (hhh:mm:ss.sss) */
        Data.usdata[0] = *(long*)w->dbdata.Elements[n].Value;
```

```
            hours = Data.usdata[0] & 0x003F;
            sprintf (hold, "%03x", hours );
            break;
        case XeTIME16:                          /* Tabular time (hhh:mm:ss.sss) */
            Data.uldata[0] = *(unsigned long*)w->dbdata.Elements[n].Value;
            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 (hold, "%02x:%02x:%02x.%03d",
                hours, minutes, seconds, milliseconds);
            break;
        case XeTIME15:                          /* Tabular time (mm:ss.sss)     */
            Data.uldata[0] = *(unsigned long*)w->dbdata.Elements[n].Value;
            minutes = (Data.uldata[0] & 0x0000FE00) >> 9;
            seconds = (Data.uldata[0] & 0x000001FF) >> 2;
            milliseconds = (Data.uldata[1] & 0x1FFF) >> 3;
            sprintf (hold, "%02x:%02x.%03d",
                minutes, seconds, milliseconds);
            break;
        case XeTIME17:                          /* Tabular time (sssss.sss)     */
            Data.usdata[0] = *(long*)w->dbdata.Elements[n].Value;
            Data.uldata[0] = *(unsigned long*)w->dbdata.Elements[n].Value;
            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) & 0x0000007) * 10;
            seconds += (days * 86400) + (hours * 3600) + (minutes * 60);

            milliseconds = (Data.uldata[1] & 0x1FFF) >> 3;
            sprintf (hold, "%*d.%03d", w->dbdata.Elements[n].Width-4,
                        seconds, milliseconds);
            break;
        default:
            break;
        }
        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 Analog Variable              */
    case 12:                            /* BC Hex Analog Variable           */
    case 14:                            /* Bit Weighted Analog Variable     */

        switch ( w->dbdata.Elements[n].Type ) {
        case XeFLOAT:
            if ( w->dbdata.Elements[n].Length <= 32 ) {
                Data.sfdata[0] = *(float*)w->dbdata.Elements[n].Value;
```

```
            sprintf (hold, w->dbdata.Format[n], Data.sfdata[0]);
        } else {
            Data.ddata = *(double*)w->dbdata.Elements[n].Value;
            sprintf (hold, w->dbdata.Format[n], Data.ddata);
        }
        break;
    case XeSIGNED:
        if ( w->dbdata.Elements[n].Attrib == 'E' ) {
            Data.sfdata[0] = *(float*)w->dbdata.Elements[n].Value;
            digit = Data.sfdata[0];
        } else {
            if ( w->dbdata.Elements[n].Length <= 32 ) {
                Data.sldata[0] = *(long*)w->dbdata.Elements[n].Value;
                sprintf (hold, w->dbdata.Format[n], Data.sldata[0]);
            } else {
                Data.ddata = *(double*)w->dbdata.Elements[n].Value;
                sprintf (hold, w->dbdata.Format[n], Data.ddata);
            }
        }
        break;
    case XeUNSIGNED:
        if ( w->dbdata.Elements[n].Attrib == 'E' ) {
            Data.ddata = *(double*)w->dbdata.Elements[n].Value;
            if (( Data.ddata < 2147483647.0 ) && (Data.ddata > -2147483648.0 ))
                idata = Data.ddata;
            else
                idata = 2147483647;
            sprintf (hold, w->dbdata.Format[n], idata);
        } else {
            if ( w->dbdata.Elements[n].Length <= 32 ) {
                Data.uldata[0] = *(unsigned long*)w->dbdata.Elements[n].Value;
                sprintf (hold, w->dbdata.Format[n], Data.uldata[0]);
            } else {
                Data.ddata = *(double*)w->dbdata.Elements[n].Value;
                sprintf (hold, w->dbdata.Format[n], Data.ddata);
            }
        }
        break;
    case XeSCIENTIFIC:
        if ( w->dbdata.Elements[n].Length <= 32 ) {
            if ( w->dbdata.Elements[n].StatFlag != 0 ) {
                Data.sldata[0] = *(long*)w->dbdata.Elements[n].Value;
                sprintf (hold, "%*.*E", w->dbdata.Elements[n].Width,
                            w->dbdata.Elements[n].Precision-5, Data.sldata[0]);
            } else {
                Data.sldata[0] = *(long*)w->dbdata.Elements[n].Value;
                sprintf (hold, w->dbdata.Format[n], Data.sldata[0]);
            }
        } else {
            if ( w->dbdata.Elements[n].StatFlag != 0 ) {
                Data.ddata = *(double*)w->dbdata.Elements[n].Value;
                sprintf (hold, "%*.*E", w->dbdata.Elements[n].Width,
                            w->dbdata.Elements[n].Precision-5, Data.ddata);
            } else {
                Data.ddata = *(double*)w->dbdata.Elements[n].Value;
                sprintf (hold, w->dbdata.Format[n], Data.ddata);
            }
        }
        break;
    case XeHEXADECIMAL:
        Data.ddata = *(double*)w->dbdata.Elements[n].Value;
        sprintf (hold, w->dbdata.Format[n], Data.ddata);
        break;
    case XeOCTAL:
```

```
                Data.ddata = *(double*)w->dbdata.Elements[n].Value;
                sprintf (hold, w->dbdata.Format[n], Data.ddata);
                break;
            case XeBINARY:
                Data.ldata[0] = *(unsigned long*)w->dbdata.Elements[n].Value;
                get_bin ( Data.ldata[0], w->dbdata.Elements[n].Width, bitstr );
                sprintf (hold, w->dbdata.Format[n], bitstr);
                break;
            case XeMULTITEXT:
                strcpy (hold, w->dbdata.Elements[n].Value);
                break;
            default:
                break;
            }
            break;

        case 'B':                                                   /* Discrete     */
        case 24:                                                    /* Discrete     */
            if ( w->dbdata.Elements[n].Type == XeMULTITEXT)
                strcpy (hold, w->dbdata.Elements[n].Value);
            break;

        case 'A':                                                /* ASCII Character String   */
        case 22:                                                 /* EBCDIC Character String  */
        case 23:                                                 /* ASCII Character String   */
            if ( w->dbdata.Elements[n].Type == XeCHARACTER )
                sprintf (hold, w->dbdata.Format[n], w->dbdata.Elements[n].Value );
            break;
        default:
            Data.ddata = *(double*)w->dbdata.Elements[n].Value;
            sprintf (hold, w->dbdata.Format[n], Data.ddata);
            break;
        }
        XTextExtents (w->dbdata.DefFont, hold, MAXLENGTH, &dir,
            &ascent, &descent, &char_info);
        w->dbdata.MaxWidth = MAX(w->dbdata.MaxWidth, char_info.width);
    }
    w->dbdata.FontHeight = ascent + descent;
/*
 *  If the user has specified PACKING as XeNO_PACKING, use the user positions.
 *  Otherwise, set up the column and row orientation based on the resources
 *  selected by the user.
 */
    if ( w->dbdata.Packing != XeNO_PACKING ) {

        if ( w->dbdata.LabelOrientation == XeCORR_LABEL ){
            width = w->dbdata.MaxWidth + label_width;
        } else
            width = w->dbdata.MaxWidth;
        if ( w->dbdata.Orientation == XeVERTICAL ) {
            rows = w->dbdata.NumValues / w->dbdata.NumColumns;
            columns = w->dbdata.NumColumns;
            if ( rows*columns < w->dbdata.NumValues ) rows++;
        } else if ( w->dbdata.Orientation == XeHORIZONTAL ) {
            rows = w->dbdata.NumColumns;
            columns = w->dbdata.NumValues / w->dbdata.NumColumns;
            if ( rows*columns < w->dbdata.NumValues ) columns++;
        } else {
            num_sqrt = w->dbdata.NumValues*(w->dbdata.FontHeight+1)*w->core.width/
                        (width*w->core.height);
            val_sqrt = sqrt(num_sqrt);
            columns = val_sqrt;
            rows = w->dbdata.NumValues / columns;
            if ( rows*columns < w->dbdata.NumValues ) rows++;
```

```
        }
    /*
     *  Set the number of rows and columns of labels based on the type of labels
     *  to be displayed.
     */
        if ( w->dbdata.LabelOrientation == XeTABLE_LABEL ) {
            label_rows = 1;
            label_columns = 0;
        } else if ( w->dbdata.LabelOrientation == XeROW_COLUMN ) {
            label_rows = 1;
            label_columns = 1;
        } else if ( w->dbdata.LabelOrientation == XeCORR_LABEL ) {
            label_rows = 0;
            label_columns = columns;
        } else if ( w->dbdata.LabelOrientation == XeNO_LABEL_ORIENT ) {
            label_rows = 0;
            label_columns = 0;
        }
    /*
     *  Calculate the midpoint of the window.
     */
        w->dbdata.MidY = w->core.height/2;
        w->dbdata.MidX = w->core.width/2;
    /*
     *  Calculate the delta values to go from one value and label to the next.
     */
        if ( w->dbdata.Packing == XePACK_TIGHT ) {
            delta_y = w->dbdata.FontHeight + 1;
            delta_x = width + 5;
        } else {
            delta_y = w->core.height / (rows + label_rows);
            if ( delta_y < w->dbdata.FontHeight )
                    delta_y = w->dbdata.FontHeight;
            delta_x = w->core.width / columns;
            if ( delta_x < width )
                    delta_x = width + 5;
        }
    /*
     *  Calculate the starting position of each value.
     */
        starty = w->dbdata.MidY - ((((label_rows + rows) * delta_y) - delta_y/2) / 2);
        if ( w->dbdata.LabelOrientation == XeTABLE_LABEL ) {
            startx = w->dbdata.MidX - ((delta_x * columns) - delta_x/2) / 2;
        } else if ( w->dbdata.LabelOrientation == XeROW_COLUMN ) {
            startx = w->dbdata.MidX - (((delta_x * columns) - delta_x/2 +
                                        (label_columns * w->dbdata.LabelWidth )) / 2);
        } else if ( w->dbdata.LabelOrientation == XeCORR_LABEL ) {
            startx = w->dbdata.MidX - ((delta_x * columns) - delta_x/2) / 2;
            startx -= w->dbdata.LabelWidth;
        } else if ( w->dbdata.LabelOrientation == XeNO_LABEL_ORIENT ) {
            startx = w->dbdata.MidX - (((delta_x * columns) - delta_x/2) / 2);
        }
        if ( startx < 0 ) startx = 0;
        if ( starty < w->dbdata.FontHeight ) starty = w->dbdata.FontHeight;
    /*
     *  Set up the label orientations based on the resources selected by the
     *  user.
     */
        for ( n = 0; n < w->dbdata.NumValues; n++ ) {
            w->dbdata.Labels[n].LabelX = -100;
            w->dbdata.Labels[n].LabelY = -100;
        }
        if ( w->dbdata.LabelOrientation == XeTABLE_LABEL ) {
            w->dbdata.Labels[0].LabelX = startx;
```

```
            w->dbdata.Labels[0].LabelY = starty;
            starty += delta_y;
        } else if ( w->dbdata.LabelOrientation == XeROW_COLUMN ) {
            if ( w->dbdata.Orientation == XeVERTICAL ) {
                w->dbdata.Labels[0].LabelX = startx + w->dbdata.LabelWidth;
                w->dbdata.Labels[0].LabelY = starty;
                for ( n = 1; n < columns; n++ ) {
                    w->dbdata.Labels[n].LabelX = w->dbdata.Labels[n-1].LabelX + delta_x;
                    w->dbdata.Labels[n].LabelY = w->dbdata.Labels[n-1].LabelY;
                }
                w->dbdata.Labels[n].LabelX = startx;
                w->dbdata.Labels[n].LabelY = starty + delta_y;
                for ( n = columns+1; n < w->dbdata.NumLabels; n++ ) {
                    w->dbdata.Labels[n].LabelX = w->dbdata.Labels[n-1].LabelX;
                    w->dbdata.Labels[n].LabelY = w->dbdata.Labels[n-1].LabelY + delta_y;
                }
            } else {
                w->dbdata.Labels[0].LabelX = startx;
                w->dbdata.Labels[0].LabelY = starty + delta_y;
                for ( n = 1; n < rows; n++ ) {
                    w->dbdata.Labels[n].LabelX = w->dbdata.Labels[n-1].LabelX;
                    w->dbdata.Labels[n].LabelY = w->dbdata.Labels[n-1].LabelY + delta_y;
                }
                w->dbdata.Labels[n].LabelX = startx + w->dbdata.LabelWidth;
                w->dbdata.Labels[n].LabelY = starty;
                for ( n = rows+1; n < w->dbdata.NumLabels; n++ ) {
                    w->dbdata.Labels[n].LabelX = w->dbdata.Labels[n-1].LabelX + delta_x;
                    w->dbdata.Labels[n].LabelY = w->dbdata.Labels[n-1].LabelY;
                }
            }
            startx += w->dbdata.LabelWidth;
            starty += delta_y;
        } else if ( w->dbdata.LabelOrientation == XeCORR_LABEL ) {
            w->dbdata.Labels[0].LabelX = startx;
            w->dbdata.Labels[0].LabelY = starty;
            row = 0;
            for ( n = 0; n < w->dbdata.NumLabels; n++ ) {
                row += 1;
                if ( w->dbdata.Orientation == XeHORIZONTAL ) {
                    if ( n != 0 ) {
                        w->dbdata.Labels[n].LabelY = w->dbdata.Labels[n-1].LabelY + delta_
y;

                        w->dbdata.Labels[n].LabelX = w->dbdata.Labels[n-1].LabelX;
                    }
                    if ( row > rows ) {
                        row = 1;
                        w->dbdata.Labels[n].LabelY = starty;
                        w->dbdata.Labels[n].LabelX += delta_x;
                    }
                } else {
                    if ( n != 0 ) {
                        w->dbdata.Labels[n].LabelY = w->dbdata.Labels[n-1].LabelY;
                        w->dbdata.Labels[n].LabelX =
                                    w->dbdata.Labels[n-1].LabelX + delta_x;
                    }
                    if ( row > columns ) {
                        row = 1;
                        w->dbdata.Labels[n].LabelY += delta_y;
                        w->dbdata.Labels[n].LabelX = startx;
                    }
                }
            }
            startx += w->dbdata.LabelWidth;
        }
```

```
    /*
     *  Calculate the positions of each value.
     */
            w->dbdata.Elements[0].Y = starty;
            w->dbdata.Elements[0].X = startx;
            row = 0;
            for ( n = 0; n < w->dbdata.NumValues; n++ ) {
                row += 1;
                if ( w->dbdata.Orientation == XeHORIZONTAL ) {
                    if ( n != 0 ) {
                        w->dbdata.Elements[n].Y = w->dbdata.Elements[n-1].Y + delta_y;
                        w->dbdata.Elements[n].X = w->dbdata.Elements[n-1].X;
                    }
                    if ( row > rows ) {
                        row = 1;
                        w->dbdata.Elements[n].Y = starty;
                        w->dbdata.Elements[n].X += delta_x;
                    }
                } else {
                    if ( n != 0 ) {
                        w->dbdata.Elements[n].Y = w->dbdata.Elements[n-1].Y;
                        w->dbdata.Elements[n].X = w->dbdata.Elements[n-1].X + delta_x;
                    }
                    if ( row > columns ) {
                        row = 1;
                        w->dbdata.Elements[n].Y += delta_y;
                        w->dbdata.Elements[n].X = startx;
                    }
                }
            }
        }
    }
    /*****************************************************************************
     *  METHOD NAME:  Redisplay
     *
     *   This method is responsible for redrawing any information in the
     *      widget's window when an Expose event occurs.
     *
     *****************************************************************************/
    static void Redisplay ( w, event, region )
        XeDbdataWidget  w;
        XEvent          *event;
        Region          region;
    {
        Display         *display = XtDisplay(w);
        if ( w->core.visible ) {
            XSetRegion ( display, w->dbdata.LabGC, region );
            XSetRegion ( display, w->dbdata.NomGC, region );
            XSetRegion ( display, w->dbdata.StaGC, region );
            XSetRegion ( display, w->dbdata.OvrGC, region );
            XSetRegion ( display, w->dbdata.DeadGC, region );
            XSetRegion ( display, w->dbdata.LowGC, region );
            XSetRegion ( display, w->dbdata.HiGC, region );
            XSetRegion ( display, w->dbdata.CrLGC, region );
            XSetRegion ( display, w->dbdata.CrHGC, region );
            w->dbdata.Redisplay = TRUE;
            update_gc ( w );
            w->dbdata.Redisplay = FALSE;
        }
    }
    /*****************************************************************************
     *  MODULE NAME:  updata_gc
     *
     *   This function is an internal function used to calculate the
```

```
 *      display positions of all values and labels based on the
 *      orientation settings.
 *
 ******************************************************************************/
static update_gc ( w )
    XeDbdataWidget  w;
{
    int             digit,
                    i,
                    ival,
                    n;
    unsigned int    idata;
    short           first_status = XeNO,    /* Msid status                  */
                    truncate_flag = XeNO;   /* Set to yes when truncated    */
    long            color,
                    uval;
    unsigned long   days,
                    hours,
                    milliseconds,
                    minutes,
                    seconds;
    double          real_hours,
                    real_min,
                    real_sec;
    char            hold[MAXLENGTH],
                    bitstr[MAXLENGTH];
    Display         *display = XtDisplay (w);
    Window          root = XtWindow(w);
    GC              gc;
    Boolean         Min_Hit = FALSE,
                    Max_Hit = FALSE;
    char            *malloc();
    char            stat_char[3];
    void            get_bin();
    struct  list    *top_min,
                    *cur_min,
                    *prev_min,
                    *top_max,
                    *cur_max,
                    *prev_max;
    xedbdataCallbackStruct  cb;

    for ( n = 0; n < w->dbdata.NumLabels; n++ ) {
        if ( w->dbdata.Labels[n].LabelX != -100 ) {
            XDrawImageString (display, root, w->dbdata.LabGC,
                        w->dbdata.Labels[n].LabelX, w->dbdata.Labels[n].LabelY,
                        w->dbdata.Label[n],
                        strlen(w->dbdata.Label[n]));
        }
    }
    for ( n = 0; n < w->dbdata.NumValues; n++ ) {
        if (( w->dbdata.OldValue[n] != w->dbdata.Elements[n].Value ) ||
            ( w->dbdata.Redisplay)) {
            w->dbdata.OldValue[n] = w->dbdata.Elements[n].Value;
            switch ( w->dbdata.Elements[n].Attrib ) {
            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 ( w->dbdata.Elements[n].Type ) {
                case XeFLOAT:
                    if ( w->dbdata.Elements[n].Length <= 32 ) {
                        Data.sfdata[0] = *(long*)w->dbdata.Elements[n].Value;
                        sprintf (hold, w->dbdata.Format[n], Data.sfdata[0]);
                    } else {
                        Data.ddata = *(double*)w->dbdata.Elements[n].Value;
                        sprintf (hold, w->dbdata.Format[n], Data.ddata);
                    }
                    break;
                case XeSIGNED:
                    if (( w->dbdata.Elements[n].Attrib == 'D' ) ||
                        ( w->dbdata.Elements[n].Attrib == 19 )) {
                        Data.ddata = *(double*)w->dbdata.Elements[n].Value;
                        if (( Data.ddata < 2147483647.0 ) && (Data.ddata > -2147483648.0 )
)
                            digit = Data.ddata;
                        else
                            digit = 2147483647;
                        sprintf (hold, w->dbdata.Format[n], digit);
                    } else {
                        if ( w->dbdata.Elements[n].Length <= 32 ) {
                            Data.ldata[0] = *(unsigned long*)w->dbdata.Elements[n].Value;
                            sprintf (hold, w->dbdata.Format[n], Data.ldata[0]);
                        } else {
                            Data.ddata = *(double*)w->dbdata.Elements[n].Value;
                            sprintf (hold, w->dbdata.Format[n], Data.ddata);
                        }
                    }
                    break;
                case XeUNSIGNED:
                    if (( w->dbdata.Elements[n].Attrib == 'D' ) ||
                        ( w->dbdata.Elements[n].Attrib == 19 )) {
                        Data.ddata = *(double*)w->dbdata.Elements[n].Value;
                        if (( Data.ddata < 2147483647.0 ) && (Data.ddata > -2147483648.0 )
)
                            Data.usdata[0] = Data.ddata;
                        else
                            Data.usdata[0] = 2147483647;
                        sprintf (hold, w->dbdata.Format[n], Data.usdata[0]);
                    } else {
                        if ( w->dbdata.Elements[n].Length <= 32 ) {
                            Data.ldata[0] = *(unsigned long*)w->dbdata.Elements[n].Value;
                            sprintf (hold, w->dbdata.Format[n], Data.ldata[0]);
                        } else {
                            Data.ddata = *(double*)w->dbdata.Elements[n].Value;
                            sprintf (hold, w->dbdata.Format[n], Data.ddata);
                        }
                    }
                    break;
                case XeSCIENTIFIC:
                    if ( w->dbdata.Elements[n].Length <= 32 ) {
                        if ( w->dbdata.Elements[n].StatFlag != 0 ) {
                            Data.ldata[0] = *(unsigned long*)w->dbdata.Elements[n].Value;
                            sprintf (hold, "%*.*E", w->dbdata.Elements[n].Width,
                                    w->dbdata.Elements[n].Precision-5, Data.ldata[0]);
                        } else {
                            Data.ldata[0] = *(unsigned long*)w->dbdata.Elements[n].Value;
                            sprintf (hold, w->dbdata.Format[n], Data.ldata[0]);
                        }
```

```
        } else {
            if ( w->dbdata.Elements[n].StatFlag != 0 ) {
                Data.ddata = *(double*)w->dbdata.Elements[n].Value;
                sprintf (hold, "%*.*E", w->dbdata.Elements[n].Width,
                            w->dbdata.Elements[n].Precision-5, Data.ddata);
            } else {
                Data.ddata = *(double*)w->dbdata.Elements[n].Value;
                sprintf (hold, w->dbdata.Format[n], Data.ddata);
            }
        }
        break;
    case XeHEXADECIMAL:
        Data.ddata = *(double*)w->dbdata.Elements[n].Value;
        sprintf (hold, w->dbdata.Format[n], Data.ddata);
        break;
    case XeOCTAL:
        Data.ddata = *(double*)w->dbdata.Elements[n].Value;
        sprintf (hold, w->dbdata.Format[n], Data.ddata);
        break;
    case XeBINARY:
        Data.ldata[0] = *(unsigned long*)w->dbdata.Elements[n].Value;
        get_bin ( Data.ldata[0], w->dbdata.Elements[n].Width, bitstr );
        sprintf (hold, w->dbdata.Format[n], bitstr);
        break;
    case XeMULTITEXT:
        strcpy (hold, w->dbdata.Elements[n].Value);
        break;
    case XeTIME10:          /* Tabular time (ddd:hh:mm:ss.sss)       */
    case XeTIME11:          /* Tabular time (yyyy:ddd:hh:mm:ss.sss)  */
    case XeTIME12:          /* Tabular time (yy:ddd:hh:mm:ss.sss)    */
    case XeTIME18:          /* Tabular time (ddd:hh:mm:ss.sss)       */
    case XeTIME19:          /* Tabular time (yyyy:ddd:hh:mm:ss.sss)  */
    case XeTIME20:          /* Tabular time (yyyy:ddd:hh:mm:ss.sss)  */
        if ( w->dbdata.Elements[n].Attrib == 'D' ) {
            Data.ddata = *(double*)w->dbdata.Elements[n].Value;
            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 ( w->dbdata.Elements[n].Type = XeTIME10 )
                sprintf (hold, "%03d:%02d:%02d:%02d.%03d",
                    days, hours, minutes, seconds, milliseconds );
            else if ( w->dbdata.Elements[n].Type = XeTIME11 )
                sprintf (hold, "%d:%03d:%02d:%02d:%02d.%03d",
                    w->dbdata.Elements[n].YearCat,
                    days, hours, minutes, seconds, milliseconds);
            else if ( w->dbdata.Elements[n].Type = XeTIME12 )
                sprintf (hold, "%d:%03d:%02d:%02d:%02d.%03d",
                    w->dbdata.Elements[n].Year,
                    days, hours, minutes, seconds, milliseconds);
            else if ( w->dbdata.Elements[n].Type = XeTIME18 )
                sprintf (hold, "%03d:%02d:%02d:%02d.%03d",
                    days, hours, minutes, seconds, milliseconds);
            else if ( w->dbdata.Elements[n].Type = XeTIME19 )
                sprintf (hold, "%d:%03d:%02d:%02d:%02d.%03d",
                    w->dbdata.Elements[n].YearCat,
                    days, hours, minutes, seconds, milliseconds);
            else if ( w->dbdata.Elements[n].Type = XeTIME20 )
                sprintf (hold, "%d:%03d:%02d:%02d:%02d.%03d",
                    w->dbdata.Elements[n].Year,
```

```
                    days, hours, minutes, seconds, milliseconds);
            } else {
                Data.usdata[0] = *(long*)w->dbdata.Elements[n].Value;
                Data.uldata[0] = *(unsigned long*)w->dbdata.Elements[n].Value;
                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[0] & 0x1FFF ) >> 3;
                if ( w->dbdata.Elements[n].Type = XeTIME10 )
                    sprintf (hold, "%03d:%02d:%02d:%02d.%03d",
                        days, hours, minutes, seconds, milliseconds );
                else if ( w->dbdata.Elements[n].Type = XeTIME11 )
                    sprintf (hold, "%d:%03d:%02d:%02d:%02d.%03d",
                        w->dbdata.Elements[n].YearCat,
                        days, hours, minutes, seconds, milliseconds);
                else if ( w->dbdata.Elements[n].Type = XeTIME12 )
                    sprintf (hold, "%d:%03d:%02d:%02d:%02d.%03d",
                        w->dbdata.Elements[n].Year,
                        days, hours, minutes, seconds, milliseconds);
                else if ( w->dbdata.Elements[n].Type = XeTIME18 )
                    sprintf (hold, "%03d:%02d:%02d:%02d.%03d",
                        days, hours, minutes, seconds, milliseconds);
                else if ( w->dbdata.Elements[n].Type = XeTIME19 )
                    sprintf (hold, "%d:%03d:%02d:%02d:%02d.%03d",
                        w->dbdata.Elements[n].YearCat,
                        days, hours, minutes, seconds, milliseconds);
                else if ( w->dbdata.Elements[n].Type = XeTIME20 )
                    sprintf (hold, "%03d:%02d:%02d:%02d.%03d",
                        days, hours, minutes, seconds, milliseconds);
            }
            break;
        case XeTIME13:                  /* Tabular time (hhh:mm:ss.sss)     */
            Data.usdata[0] = *(long*)w->dbdata.Elements[n].Value;
            hours = Data.usdata[0] & 0x003F;
            sprintf (hold, "%03x", hours );
            break;
        case XeTIME16:                  /* Tabular time (hhh:mm:ss.sss)     */
            Data.uldata[0] = *(unsigned long*)w->dbdata.Elements[n].Value;
            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 (hold, "%02x:%02x:%02x.%03d",
                hours, minutes, seconds, milliseconds);
            break;
        case XeTIME15:                  /* Tabular time (mm:ss.sss)     */
            Data.uldata[0] = *(unsigned long*)w->dbdata.Elements[n].Value;
            minutes = (Data.uldata[0] & 0x0000FE00) >> 9;
            seconds = (Data.uldata[0] & 0x000001FF) >> 2;
            milliseconds = (Data.uldata[1] & 0x1FFF) >> 3;
            sprintf (hold, "%02x:%02x.%03d",
                minutes, seconds, milliseconds);
            break;
        case XeTIME17:                  /* Tabular time (sssss.sss)     */
            Data.usdata[0] = *(long*)w->dbdata.Elements[n].Value;
            Data.uldata[0] = *(unsigned long*)w->dbdata.Elements[n].Value;
            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) & 0x0000007) * 10;
            seconds += (days * 86400) + (hours * 3600) + (minutes * 60);

            milliseconds = (Data.uldata[1] & 0x1FFF) >> 3;
            sprintf (hold, "%*d.%03d", w->dbdata.Elements[n].Width-4,
                        seconds, milliseconds);
            break;
        default:
            break;
        }
        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 Analog Variable              */
    case 12:                        /* BCD Hex Analog Variable          */
    case 14:                        /* Bit Weighted Analog Varialble    */
        switch ( w->dbdata.Elements[n].Type ) {
        case XeFLOAT:
            if ( w->dbdata.Elements[n].Length <= 32 ) {
                Data.sfdata[0] = *(float*)w->dbdata.Elements[n].Value;
                sprintf (hold, w->dbdata.Format[n], Data.sfdata[0]);
            } else {
                Data.ddata = *(double*)w->dbdata.Elements[n].Value;
                sprintf (hold, w->dbdata.Format[n], Data.ddata);
            }
            break;
        case XeSIGNED:
            if ( w->dbdata.Elements[n].Attrib == 'E' ) {
                Data.sfdata[0] = *(long*)w->dbdata.Elements[n].Value;
                digit = Data.sfdata[0];
            } else {
                if ( w->dbdata.Elements[n].Length <= 32 ) {
                    Data.ldata[0] = *(unsigned long*)w->dbdata.Elements[n].Value;
                    sprintf (hold, w->dbdata.Format[n], Data.ldata[0]);
                } else {
                    Data.ddata = *(double*)w->dbdata.Elements[n].Value;
                    sprintf (hold, w->dbdata.Format[n], Data.ddata);
                }
            }
            break;
        case XeUNSIGNED:
            if ( w->dbdata.Elements[n].Attrib == 'E' ) {
                Data.ddata = *(double*)w->dbdata.Elements[n].Value;
                if (( Data.ddata < 2147483647.0 ) && (Data.ddata > -2147483648.0 )
)
                    idata = Data.ddata;
                else
                    idata = 2147483647;
                sprintf (hold, w->dbdata.Format[n], idata);
            } else {
                if ( w->dbdata.Elements[n].Length <= 32 ) {
```

```
                    Data.ldata[0] = *(unsigned long*)w->dbdata.Elements[n].Value;
                    sprintf (hold, w->dbdata.Format[n], Data.ldata[0]);
                } else {
                    Data.ddata = *(double*)w->dbdata.Elements[n].Value;
                    sprintf (hold, w->dbdata.Format[n], Data.ddata);
                }
            }
            break;
        case XeSCIENTIFIC:
            if ( w->dbdata.Elements[n].Length <= 32 ) {
                if ( w->dbdata.Elements[n].StatFlag != 0 ) {
                    Data.ldata[0] = *(unsigned long*)w->dbdata.Elements[n].Value;
                    sprintf (hold, "%*.*E", w->dbdata.Elements[n].Width,
                                w->dbdata.Elements[n].Precision-5, Data.ldata[0]);
                } else {
                    Data.ldata[0] = *(unsigned long*)w->dbdata.Elements[n].Value;
                    sprintf (hold, w->dbdata.Format[n], Data.ldata[0]);
                }
            } else {
                if ( w->dbdata.Elements[n].StatFlag != 0 ) {
                    Data.ddata = *(double*)w->dbdata.Elements[n].Value;
                    sprintf (hold, "%*.*E", w->dbdata.Elements[n].Width,
                                w->dbdata.Elements[n].Precision-5, Data.ddata);
                } else {
                    Data.ddata = *(double*)w->dbdata.Elements[n].Value;
                    sprintf (hold, w->dbdata.Format[n], Data.ddata);
                }
            }
            break;
        case XeHEXADECIMAL:
            Data.ddata = *(double*)w->dbdata.Elements[n].Value;
            sprintf (hold, w->dbdata.Format[n], Data.ddata);
            break;
        case XeOCTAL:
            Data.ddata = *(double*)w->dbdata.Elements[n].Value;
            sprintf (hold, w->dbdata.Format[n], Data.ddata);
            break;
        case XeBINARY:
            Data.ldata[0] = *(unsigned long*)w->dbdata.Elements[n].Value;
            get_bin ( Data.ldata[0], w->dbdata.Elements[n].Width, bitstr );
            sprintf (hold, w->dbdata.Format[n], bitstr);
            break;
        case XeMULTITEXT:
            strcpy (hold, w->dbdata.Elements[n].Value);
            break;
        default:
            break;
        }
        break;
    case 'B':                                       /* Discrete      */
    case 24:                                        /* Discrete      */
        if ( w->dbdata.Elements[n].Type == XeMULTITEXT)
            strcpy (hold, w->dbdata.Elements[n].Value);
        break;
    case 'A':                               /* ASCII Character String   */
    case 22:                                /* EBCDIC Character String  */
    case 23:                                /* ASCII Character String   */
        if ( w->dbdata.Elements[n].Type == XeCHARACTER ) {
            sprintf (hold, w->dbdata.Format[n], w->dbdata.Elements[n].Value );
        }
        break;
    default:
        Data.ddata = *(double*)w->dbdata.Elements[n].Value;
        sprintf (hold, w->dbdata.Format[n], Data.ddata);
```

```
                break;
            }
/*
 *  Check for set status bits and attach a status character to the display string.
 */
            first_status = XeNO;
/*
 *  Dead Data
 */
            if ( w->dbdata.Elements[n].DispStat & XeDEAD_DATA ) {
                color = w->dbdata.Elements[n].DeadColor;
                gc = w->dbdata.DeadGC;
                stat_char[0] = 'D';
                for ( i = 0; i < MAXLENGTH; i++ )
                    hold[i]  = ' ';
                first_status = XeYES;
/*
 *  Missing
 */
            } else if ( w->dbdata.Elements[n].DispStat & XeMISSING_DATA ) {
                color = w->dbdata.Elements[n].StaColor;
                gc = w->dbdata.StaGC;
                stat_char[0] = 'M';
                for ( i = 0; i < MAXLENGTH; i++ )
                    hold[i] = ' ';
                first_status = XeYES;
/*
 *  Static
 */
            } else if ( w->dbdata.Elements[n].DispStat & XeSTATIC_DATA ) {
                color = w->dbdata.Elements[n].StaColor;
                gc = w->dbdata.StaGC;
                stat_char[0] = 'S';
                first_status = XeYES;
/*
 *  Out of crit. high
 */
            } else if ( w->dbdata.Elements[n].DispStat & XeOFF_SCALE_HIGH ) {
                color = w->dbdata.Elements[n].CrHColor;
                gc = w->dbdata.CrHGC;
                stat_char[0] = 'H';
                first_status = XeYES;
/*
 *  Out of crit. low
 */
            } else if ( w->dbdata.Elements[n].DispStat & XeOFF_SCALE_LOW ) {
                color = w->dbdata.Elements[n].CrLColor;
                gc = w->dbdata.CrLGC;
                stat_char[0] = 'L';
                first_status = XeYES;
/*
 *  Out of crit. high
 */
            } else if ( w->dbdata.Elements[n].DispStat & XeCRITICAL_HIGH ) {
                color = w->dbdata.Elements[n].CrHColor;
                gc = w->dbdata.CrHGC;
                stat_char[0] = 'H';
                first_status = XeYES;
/*
 *  Out of crit. low
 */
            } else if ( w->dbdata.Elements[n].DispStat & XeCRITICAL_LOW ) {
                color = w->dbdata.Elements[n].CrLColor;
                gc = w->dbdata.CrLGC;
```

```
                stat_char[0] = 'L';
                first_status = XeYES;
/*
 *  Out of limits high
 */
            } else if ( w->dbdata.Elements[n].DispStat & XeLIMIT_HIGH ) {
                color = w->dbdata.Elements[n].HiColor;
                gc = w->dbdata.HiGC;
                stat_char[0] = 'H';
                first_status = XeYES;
                if ( !Max_Hit ) {
                    top_max = (struct list*) malloc(sizeof(struct list));
                    cur_max = top_max;
                    prev_max = top_max;
                } else {
                    prev_max->next = (struct list*) malloc(sizeof(struct list));
                    cur_max = prev_max->next;
                }
                cur_max->index = n;
                cur_max->next = NULL;
                prev_max = cur_max;
                Max_Hit = TRUE;
/*
 *  Out of limits low
 */
            } else if ( w->dbdata.Elements[n].DispStat & XeLIMIT_LOW ) {
                color = w->dbdata.Elements[n].LowColor;
                gc = w->dbdata.LowGC;
                stat_char[0] = 'L';
                first_status = XeYES;
                if ( !Min_Hit ) {
                    top_min = (struct list*) malloc(sizeof(struct list));
                    cur_min = top_min;
                    prev_min = top_min;
                } else {
                    prev_min->next = (struct list*) malloc(sizeof(struct list));
                    cur_min = prev_min->next;
                }
                cur_min->index = n;
                cur_min->next = NULL;
                prev_min = cur_min;
                Min_Hit = TRUE;
            }
/*
 *  Truncation
 */
            if ( truncate_flag == XeYES ) {
                stat_char[0] = 'T';
                truncate_flag = XeNO;
                first_status = XeYES;
            }
            if ( first_status == XeNO ) {
                color = w->dbdata.Elements[n].NomColor;
                gc = w->dbdata.NomGC;
                stat_char[0] = ' ';
            }
            hold[w->dbdata.Elements[n].Width] = NULL;
            stat_char[1] = NULL;

            if ( w->dbdata.Elements[n].StatFlag == 0 )
                stat_char[0] = ' ';
            strncat ( hold, stat_char, 2 );

            XDrawImageString (display, root, gc,
```

```
                w->dbdata.Elements[n].X, w->dbdata.Elements[n].Y, hold,
                        strlen(hold));
            }       /*  End of if Change or Redisplay   */
        }       /*  End of Number of Values Loop    */
    /*
     *  Check for limit exceptions and execute the appropriate callback routines.
     */
        if ( Max_Hit && !w->dbdata.Redisplay ) {
            cb.indices = top_max;
            XtCallCallbacks (w, XtNmaxCallback, &cb);
            cur_max = top_max;
            while ( cur_max != NULL ) {
                prev_max = cur_max;
                cur_max = prev_max->next;
                free ( prev_max );
            }
        }
        if ( Min_Hit && !w->dbdata.Redisplay ) {
            cb.indices = top_min;
            XtCallCallbacks (w, XtNminCallback, &cb);
            cur_min = top_min;
            while ( cur_min != NULL ) {
                prev_min = cur_min;
                cur_min = prev_min->next;
                free ( prev_min );
            }
        }
    }
    /*****************************************************************************
     *  MODULE NAME:  get_bin
     *
     *   This function converts an unsigned integer value into a string
     *      of length 'precision'.
     *
     *****************************************************************************/
    void get_bin ( uval, width, bitstr )
        long        uval;
        int         width;
        char        bitstr[];
    {
        int         n;
        long        bit;

        for ( n = 0; n < width; n++ ) {
            bit = (( uval >> ( width - n - 1 )) & ~(~0<<1));
            if ( bit & 01 )
                bitstr[n] = '1';
            else
                bitstr[n] = '0';
        }
        bitstr[n] = '\0';
    }
    /*****************************************************************************
     *  METHOD NAME:  SetValues
     *
     *   This method allows a widget to be notified when one of its
     *      resources is set or changed.  This can occur when the
     *      resource manager initializes the widget's resources,
     *      or when an application calls XtSetValues().
     *
     *****************************************************************************/
    static  Boolean SetValues   (current, request, new)
        XeDbdataWidget  current,
                        request,
```

```
                    new;
{
    caddr_t         client_data;
    int             n;
    XGCValues       values;
    XtGCMask        valueMask = GCForeground | GCBackground;
    Boolean         redraw = FALSE;
    Boolean         warning = FALSE;
    Display         *display = XtDisplay (current);
    Window          root = RootWindowOfScreen(XtScreen(current));
/*
 *  If the timer interval has changed, remove the current callback and
 *  add another one with the new interval.
 */
    if ( new->dbdata.Interval != current->dbdata.Interval )
        XtRemoveTimeOut ( current->dbdata.Id );
        client_data = (caddr_t)new;
        new->dbdata.Id = XtAddTimeOut (new->dbdata.Interval, check_data,
                                       client_data);
/*
 *  Check minimum and maximum.
 */
    for ( n = 0; n < new->dbdata.NumValues; n++ ) {
        if ( new->dbdata.Elements[n].MinLimit >
             new->dbdata.Elements[n].MaxLimit ) {
            new->dbdata.Elements[n].MinLimit = 0;
            new->dbdata.Elements[n].MaxLimit = 100;
            warning = TRUE;
        }
    }
    if ( warning )
        XtWarning ("Minimum must be less than maximum");
/*
 *  If any colors have changed, generate new GCs and set the
 *  redraw flag.
 */
    if ( new->dbdata.DefBackground != current->dbdata.DefBackground ||
         new->dbdata.DefForeground != current->dbdata.DefForeground ) {
        values.background = new->dbdata.DefBackground;
        values.foreground = new->dbdata.DefForeground;
        XFreeGC (new, new->dbdata.NomGC);
        new->dbdata.NomGC = XCreateGC(display, root, valueMask, &values);
        redraw = TRUE;
    }
    if ( new->dbdata.LabBackground != current->dbdata.LabBackground ||
         new->dbdata.LabForeground != current->dbdata.LabForeground ) {
        values.background = new->dbdata.LabBackground;
        values.foreground = new->dbdata.LabForeground;
        XFreeGC (new, new->dbdata.LabGC);
        new->dbdata.LabGC = XCreateGC(display, root, valueMask, &values);
        redraw = TRUE;
    }
    return (redraw);
}
/*****************************************************************************
 * MODULE NAME:  check_data
 *
 *  This function is an internal timer function used to check the
 *      data values for changes.  This timer interval can be
 *      specified as a resource.
 *
 *****************************************************************************/
XtTimerCallbackProc check_data ( client_data, id )
    caddr_t         client_data;
```

```
    XtIntervalId    *id;
{
    XtIntervalId    data_id;
    XeDbdataWidget  w;

    w = (XeDbdataWidget) client_data;
    update_gc (w);
    w->dbdata.Id = XtAddTimeOut (w->dbdata.Interval, check_data, client_data);
}
```

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

CONTINUATION OF RESEARCH IN SOFTWARE FOR SPACE OPERATIONS SUPPORT

X WINDOWS PERFORMANCE TESTS

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

This report summarizes the results of tests which were conducted to gather basic statistics on displaying text using low-level Xlib functions of the X Windows system. In this report is a table containing the results of 18 separate tests. The results of these tests are to be compared with the results of tests being conducted by Lorie Howard of NASA/JSC. Ms. Howard is conducting similar tests using the GKS system instead of X Windows.

The function of each test was to display 100 cycles of 500 values to an X window using the Xlib function XDrawImageString (XDrawString for tests 17 & 18). Performance data was collected using the PRM (Process Resource Monitor) application. The focus of the test was to determine what effect font size, text formatting, and alternating graphics contexts had on CPU utilization in the display of text.

Following the test results is a key which expands each test name into a more descriptive list of attributes describing the characteristics of the test.

2.0 TEST RESULTS

					80% CPU Usage		50% CPU Usage	
#	Test Name	User Time	System Time	Total Time	Milli Sec	% of Total	Milli Sec	% of Total
1	stnofmt9x15	2.916	1.616	4.532	34	1.70	33	1.65
2	stnfgc9x15	2.800	1.983	4.783	34	1.70	33	1.65
3	stnfmgc9x15	3.033	1.533	4.566	34	1.70	33	1.65
4	stint9x15	4.916	1.216	6.132	100	5.00	67	3.35
5	stingc9x15	5.450	1.350	6.800	116	5.80	84	4.20
6	stinmgc9x15	5.700	1.483	7.183	116	5.80	100	5.00
7	stfloat9x15	19.166	2.266	21.432	184	9.20	167	8.35
8	stflgc9x15	19.116	2.083	21.199	184	9.20	167	8.35
9	stflmgc9x15	19.866	1.716	21.582	184	9.20	167	8.35
10	stnofmt6x10	2.366	1.900	4.266	49	2.45	33	1.65
11	stfloat6x10	20.100	1.933	22.033	233	11.65	200	10.00
12	stnofmttms16	2.700	2.516	5.216	50	2.50	17	0.85
13	stflmgctms16	20.150	1.766	21.916	200	10.00	183	9.15
14	stnofmttms22	2.983	2.016	4.999	50	2.50	33	1.65
15	stflmgctms22	20.950	1.550	22.500	200	10.00	167	8.35
16	stflmgccs38	19.600	2.133	21.733	183	9.15	149	7.45
17	drnofmt9x15	2.733	1.733	4.466	50	2.50	33	1.65
18	drflmgc9x15	23.150	2.216	25.366	234	11.70	217	10.85

Test Name Key:

Test Name	X Text Command	Format	Font	# graphics context switches/cycle
stnofmt9x15	XDrawImageString	none	9x15	none
stnfgc9x15	XDrawImageString	none	9x15	1
stnfmgc9x15	XDrawImageString	none	9x15	100
stint9x15	XDrawImageString	integer	9x15	none
stingc9x15	XDrawImageString	integer	9x15	1
stinmgc9x15	XDrawImageString	integer	9x15	100
stfloat9x15	XDrawImageString	float	9x15	none
stflgc9x15	XDrawImageString	float	9x15	1
stflmgc9x15	XDrawImageString	float	9x15	100
stnofmt6x10	XDrawImageString	none	6x10	none
stfloat6x10	XDrawImageString	float	6x10	none
stnofmttms16	XDrawImageString	none	times16	none
stflmgctms16	XDrawImageString	float	times16	100
stnofmttms22	XDrawImageString	none	times22	none
stflmgctms22	XDrawImageString	float	times22	100
stflmgccs38	XDrawImageString	float	cyr-s38	100
drnofmt9x15	XDrawString	none	9x15	none
drflmgc9x15	XDrawString	float	9x15	100

3.0 CONCLUSIONS

The data collected in these tests indicate that for display of individual text strings using the Xlib functions XDrawImageString:

- Alternating graphics contexts has no detectable impact on performance.
- Text fonts of varying sizes may be displayed without performance degradation.
- Formatting of text for display using the C function sprintf has a significant negative impact on performance, particularly when formatting floating-point values.

A few similar tests were conducted using X Pixmaps and XCopyArea. However, these tests quickly demonstrated a much lower performance, and the test results were not included.

In addition, it was thought that it might be more efficient to clear the entire window using the Xlib function XClearArea, and then use XDrawString to display the text items. However, XDrawImageString could not be shown to be less efficient than XDrawString (compare tests 17 & 18 with tests 1 & 9).

4.0 ATTACHMENTS

The following pages contain the detailed information collected for this performance evaluation. This information includes:

- Makefile - used to build each of the performance measurement programs.
- Source code - the source code for the performance measurement programs.
- Analysis results - the output from the Process Resource Monitor (PRM) application, which was used as the statistical basis for the performance evaluation.

ATTACHMENT 1 - Makefile

```
PROGS = drflmgc drnofmt pxfloat pxnofmt stflmgc stflgc stfloat \
    stingc stinmgc stint stnfgc stnfmgc stnofmt

all:    ${PROGS}


drflmgc:
    cc -o drflmgc drflmgc.c -lXm -lXt -lX11

drnofmt:
    cc -o drnofmt drnofmt.c -lXm -lXt -lX11

pxfloat:
    cc -o pxfloat pxfloat.c -lXm -lXt -lX11

pxnofmt:
    cc -o pxnofmt pxnofmt.c -lXm -lXt -lX11

stflgc:
    cc -o stflgc stflgc.c -lXm -lXt -lX11

stflmgc:
    cc -o stflmgc stflmgc.c -lXm -lXt -lX11

stfloat:
    cc -o stfloat stfloat.c -lXm -lXt -lX11

stingc:
    cc -o stingc stingc.c -lXm -lXt -lX11

stinmgc:
    cc -o stinmgc stinmgc.c -lXm -lXt -lX11

stint:
    cc -o stint stint.c -lXm -lXt -lX11

stnfgc:
    cc -o stnfgc stnfgc.c -lXm -lXt -lX11

stnfmgc:
    cc -o stnfmgc stnfmgc.c -lXm -lXt -lX11

stnofmt:
    cc -o stnofmt stnofmt.c -lXm -lXt -lX11
```

ATTACHMENT 2 - Source Code

```
/*
    Draw straight to screen using XDrawString
    Multiple graphics context switches/cycle
    %10.5f text formatting
 */

#include <stdio.h>
#include <string.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Cardinals.h>
#include <X11/Shell.h>
#include <X11/MwmUtil.h>
#include <Xm/Xm.h>
#include <Xm/DialogS.h>
#include <Xm/DrawingA.h>
#include <Xm/MainW.h>
#include <Xm/RowColumn.h>


#define CYCLES      100
#define TIMER_VALUE 1000
#define ROWS        50
#define COLS        10

Display             *display;

Visual              *visual;

Pixmap              pixmap;

GC                  gc[5];

Font                font;

Window              window;

XFontStruct         *font_info;

int                 screen;
                    ascent,
                    width,
                    height;

char                string[] = "1000.00001",
                    format[10];


int main ( argc, argv )

    int     argc;

    char    *argv[];
{
    register int            i;

    Widget                  top, m_main, mb_main, mp_file, scroll, draw;

    Arg                     args[10];

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

```
    XtCallbackProc          cb_expose();

    XColor                  color;

    Colormap                xcmap;

    XVisualInfo             v,
                            *visual_list;

    XSetWindowAttributes    attributes;

    XCharStruct             overall;

    int                     visuals_matched;

    unsigned long           mask;


    int                     z;


/*
 *  Initialize the Toolkit.
 */

    top = XtInitialize ( argv[0], "Fast", NULL, 0, &argc, argv );

/*
 *  Save a pointer to the X Windows display structure. Save the current screen.
 */

    display = XtDisplay ( top );
    screen  = DefaultScreen ( display );

/*
 *  Initialize font information.
 */

    if ( ( font_info = XLoadQueryFont ( display, argv[1] ) ) == 0 )
        exit ( 1 );

    font = font_info->fid;

/*
 *  Save the format to use for printing out of data.
 */

    strcpy ( format, argv[2] );

/*
 *  Initialize size of string to draw.
 */

    XTextExtents ( font_info, string, strlen ( string ), &ascent,
                   &ascent, &ascent, &overall );
    width  = overall.width;
    height = overall.ascent + overall.descent;
    ascent = overall.ascent;

/*
 *  Query the X server to find out if there is the right type of visual.
 */
```

```
    v.screen = screen;
    v.depth  = 8;
    v.class  = PseudoColor;

    visual_list = XGetVisualInfo ( display,
                        VisualScreenMask | VisualDepthMask | VisualClassMask,
                        &v, &visuals_matched );

/*
 *  Copy the required visual and free up memory allocated for by the visual
 *  query function.
 */

    memcpy ( &visual, &visual_list[0].visual, sizeof ( visual ) );
    XFree ( visual_list );

/*
 *  Create the main window and a menu bar.
 */

    XtManageChild ( m_main = XmCreateMainWindow ( top, "", NULL, 0 ) );
    XtManageChild ( mb_main = XmCreateMenuBar ( m_main, "", NULL, 0 ) );

/*
 *  Create menu.
 */

    mp_file = XmCreatePulldownMenu ( mb_main, "", NULL, 0 );

    i = 0;
    XtSetArg ( args[i], XmNsubMenuId, mp_file ); i++;
    XtManageChild ( XmCreateCascadeButton ( mb_main, "File", args, i ) );
    XtManageChild ( XmCreatePushButton    ( mp_file, "Exit", NULL, 0 ) );

/*
 *  Create a scrolled window widget.
 */

    i = 0;
    XtSetArg ( args[i], XmNwidth,  COLS*(width +2)); i++;
    XtSetArg ( args[i], XmNheight, ROWS*(height+2)); i++;
    XtManageChild ( scroll =
            XmCreateScrolledWindow ( m_main, "scroll", args, i ) );

/*
 *  Create the drawing area widget.
 */

    cb[0].callback = (XtCallbackProc)cb_expose;
    cb[0].closure  = (caddr_t)0;

    i = 0;
    XtSetArg ( args[i], XmNexposeCallback, cb             ); i++;
    XtSetArg ( args[i], XmNwidth,          COLS*(width +2)); i++;
    XtSetArg ( args[i], XmNheight,         ROWS*(height+2)); i++;
    XtManageChild ( draw =
            XmCreateDrawingArea ( scroll, "draw", args, i ) );

/*
 *  Realize the widgets.
 */

    XtRealizeWidget ( top );
```

```
/*
 *  Set the attributes necessary to create the actual window.
 */

    attributes.save_under        = 0;
    attributes.backing_store     = NotUseful;
    attributes.border_pixel      = BlackPixel ( display, screen );
    attributes.background_pixel  = WhitePixel ( display, screen );
    attributes.bit_gravity       = NorthWestGravity;

    mask = CWBackingStore | CWSaveUnder | CWBackPixel |
           CWBorderPixel  | CWBitGravity;

/*
 *  Create the window for the drawing area widget.
 */

    XtCreateWindow ( draw, CopyFromParent, visual, mask, &attributes );
    window = XtWindow ( draw );

    xcmap = DefaultColormap(display, screen);

/*
 *  Create the graphics context.
 */

    gc[1] = XCreateGC ( display, window, NULL, NULL );

    gc[2] = XCreateGC ( display, window, NULL, NULL );
    color.red = 65000;
    color.blue = 0;
    color.green = 0;
    if (XAllocColor(display, xcmap, &color))
        XSetBackground(display, gc[2], color.pixel);
    else
        fprintf(stderr, "couldn't allocate color");

    gc[3] = XCreateGC ( display, window, NULL, NULL );
    color.red = 0;
    color.blue = 65000;
    color.green = 0;
    if (XAllocColor(display, xcmap, &color))
        XSetBackground(display, gc[3], color.pixel);
    else
        fprintf(stderr, "couldn't allocate color");

    gc[4] = XCreateGC ( display, window, NULL, NULL );
    color.red = 0;
    color.blue = 0;
    color.green = 65000;
    if (XAllocColor(display, xcmap, &color))
        XSetBackground(display, gc[4], color.pixel);
    else
        fprintf(stderr, "couldn't allocate color");

    for (z=1; z<5; z++)
        XSetFont ( display, gc[z], font );

/*
 *  Create the pixmap.
 */

    pixmap = XCreatePixmap ( display, window, COLS*(width +2), ROWS*(height+2),
                             8 );
```

```
/*
 *  Add a time out.
 */

    XtAddTimeOut ( TIMER_VALUE, cb_expose, NULL );

/*
 *  Loop forever.
 */

    XtMainLoop ( );
}


XtCallbackProc cb_expose ( widget, closure, calldata )

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

    caddr_t closure;            /* Callback specific data.  This parameter
                                 * indicates the selected function.
                                 */

    XmDrawingAreaCallbackStruct *calldata;
                                /* Specifies any callback-specific data the
                                 * widget needs to pass to the client.
                                 */
{
    register int        x, y;

    static double       value = 1000.00001;

    static int          count=1;

    int                 z;

/*
 *  Update the string.
 */

    value += (double)0.00001;

/*
 *  Write out the strings.
 */

    for ( y = 0; y < ROWS; y++ )
        for ( x = 0; x < COLS; x++ ) {
            z = ((x + y) % 4) + 1;
            sprintf ( string, format, value );
            XDrawString ( display, window, gc[z], x*(width+2),
                    ascent + (y*(height+2)), string, strlen ( string ) );
        }

/*
 *  Flush the buffer immediately.
 */

    XFlush ( display );

/*
 *  Test and increment counter
 */
```

```
    if (count > CYCLES)
        exit(1);
    count++;


/*
 *  Reset the timer.
 */

    XtAddTimeOut ( TIMER_VALUE, cb_expose, NULL );
}
```

```
/*
    Draw straight to screen using XDrawString
    no gc switching
    No text formatting
 */

#include <stdio.h>
#include <string.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Cardinals.h>
#include <X11/Shell.h>
#include <X11/MwmUtil.h>
#include <Xm/Xm.h>
#include <Xm/DialogS.h>
#include <Xm/DrawingA.h>
#include <Xm/MainW.h>
#include <Xm/RowColumn.h>


#define CYCLES      100
#define TIMER_VALUE 1000
#define ROWS        50
#define COLS        10

Display             *display;

Visual              *visual;

Pixmap              pixmap;

GC                  gc;

Font                font;

Window              window;

XFontStruct         *font_info;

int                 screen;
                    ascent,
                    width,
                    height;

char                string[] = "1000.00001",
                    format[10];


int main ( argc, argv )

    int     argc;

    char    *argv[];
{
    register int            i;

    Widget                  top, m_main, mb_main, mp_file, scroll, draw;

    Arg                     args[10];

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

```
    XtCallbackProc          cb_expose();

    XColor                  color;

    XVisualInfo             v,
                            *visual_list;

    XSetWindowAttributes    attributes;

    XCharStruct             overall;

    int                     visuals_matched;

    unsigned long           mask;

/*
 *  Initialize the Toolkit.
 */

    top = XtInitialize ( argv[0], "Fast", NULL, 0, &argc, argv );

/*
 *  Save a pointer to the X Windows display structure. Save the current screen.
 */

    display = XtDisplay ( top );
    screen  = DefaultScreen ( display );

/*
 *  Initialize font information.
 */

    if ( ( font_info = XLoadQueryFont ( display, argv[1] ) ) == 0 )
        exit ( 1 );

    font = font_info->fid;

/*
 *  Save the format to use for printing out of data.
 */

    strcpy ( format, argv[2] );

/*
 *  Initialize size of string to draw.
 */

    XTextExtents ( font_info, string, strlen ( string ), &ascent,
                   &ascent, &ascent, &overall );
    width  = overall.width;
    height = overall.ascent + overall.descent;
    ascent = overall.ascent;

/*
 *  Query the X server to find out if there is the right type of visual.
 */

    v.screen = screen;
    v.depth  = 8;
    v.class  = PseudoColor;

    visual_list = XGetVisualInfo ( display,
                      VisualScreenMask | VisualDepthMask | VisualClassMask,
```

```
                        &v, &visuals_matched );

    /*
     *  Copy the required visual and free up memory allocated for by the visual
     *  query function.
     */

        memcpy ( &visual, &visual_list[0].visual, sizeof ( visual ) );
        XFree ( visual_list );

    /*
     *  Create the main window and a menu bar.
     */

        XtManageChild ( m_main = XmCreateMainWindow ( top, "", NULL, 0 ) );
        XtManageChild ( mb_main = XmCreateMenuBar ( m_main, "", NULL, 0 ) );

    /*
     *  Create menu.
     */

        mp_file = XmCreatePulldownMenu ( mb_main, "", NULL, 0 );

        i = 0;
        XtSetArg ( args[i], XmNsubMenuId, mp_file ); i++;
        XtManageChild ( XmCreateCascadeButton ( mb_main, "File", args, i ) );
        XtManageChild ( XmCreatePushButton    ( mp_file, "Exit", NULL, 0 ) );

    /*
     *  Create a scrolled window widget.
     */

        i = 0;
        XtSetArg ( args[i], XmNwidth,  COLS*(width +2)); i++;
        XtSetArg ( args[i], XmNheight, ROWS*(height+2)); i++;
        XtManageChild ( scroll =
                XmCreateScrolledWindow ( m_main, "scroll", args, i ) );

    /*
     *  Create the drawing area widget.
     */

        cb[0].callback = (XtCallbackProc)cb_expose;
        cb[0].closure  = (caddr_t)0;

        i = 0;
        XtSetArg ( args[i], XmNexposeCallback, cb             ); i++;
        XtSetArg ( args[i], XmNwidth,          COLS*(width +2)); i++;
        XtSetArg ( args[i], XmNheight,         ROWS*(height+2)); i++;
        XtManageChild ( draw =
                XmCreateDrawingArea ( scroll, "draw", args, i ) );

    /*
     *  Realize the widgets.
     */

        XtRealizeWidget ( top );

    /*
     *  Set the attributes necessary to create the actual window.
     */

        attributes.save_under        = 0;
        attributes.backing_store     = NotUseful;
```

```
    attributes.border_pixel      = BlackPixel ( display, screen );
    attributes.background_pixel  = WhitePixel ( display, screen );
    attributes.bit_gravity       = NorthWestGravity;

    mask = CWBackingStore | CWSaveUnder | CWBackPixel |
           CWBorderPixel  | CWBitGravity;

/*
 *  Create the window for the drawing area widget.
 */

    XtCreateWindow ( draw, CopyFromParent, visual, mask, &attributes );
    window = XtWindow ( draw );

/*
 *  Create the graphics context.
 */

    gc = XCreateGC ( display, window, NULL, NULL );

    XSetFont ( display, gc, font );

/*
 *  Create the pixmap used to retain a copy of the image for refreshing
 *  the window.
 */

    pixmap = XCreatePixmap ( display, window, COLS*(width +2), ROWS*(height+2),
                             8 );

/*
 *  Add a time out.
 */

    XtAddTimeOut ( TIMER_VALUE, cb_expose, NULL );

/*
 *  Loop forever.
 */

    XtMainLoop ( );
}


XtCallbackProc cb_expose ( widget, closure, calldata )

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

    caddr_t closure;            /* Callback specific data.  This parameter
                                 * indicates the selected function.
                                 */
    XmDrawingAreaCallbackStruct *calldata;
                                /* Specifies any callback-specific data the
                                 * widget needs to pass to the client.
                                 */
{
    register int        x, y;

    static double       value = 1000.00001;

    static int          count=1;
```

```
/*
 *  Update the string.
 */

    value += (double)0.00001;

/*
 *  Write out the strings.
 */

    for ( y = 0; y < ROWS; y++ )
        for ( x = 0; x < COLS; x++ ) {
            XDrawString ( display, window, gc, x*(width+2),
                          ascent + (y*(height+2)), string, strlen ( string ) );
        }

/*
 *  Flush the buffer immediately.
 */

    XFlush ( display );

/*
 *  Test and increment counter
 */

    if (count > CYCLES)
        exit(1);
    count++;


/*
 *  Reset the timer.
 */

    XtAddTimeOut ( TIMER_VALUE, cb_expose, NULL );
}
```

```
/*
    Pixmap/XCopyArea
    No graphics context switching
    %10.5f text formatting
 */

#include <stdio.h>
#include <string.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Cardinals.h>
#include <X11/Shell.h>
#include <X11/MwmUtil.h>
#include <Xm/Xm.h>
#include <Xm/DialogS.h>
#include <Xm/DrawingA.h>
#include <Xm/MainW.h>
#include <Xm/RowColumn.h>


#define CYCLES      100
#define TIMER_VALUE 1000
#define ROWS        50
#define COLS        10

Display             *display;

Visual              *visual;

Pixmap              pixmap;

GC                  gc;

Font                font;

Window              window;

XFontStruct         *font_info;

int                 screen;
                    ascent,
                    width,
                    height;

char                string[] = "1000.00001",
                    format[10];


int main ( argc, argv )

    int     argc;

    char    *argv[];
{
    register int            i;

    Widget                  top, m_main, mb_main, mp_file, scroll, draw;

    Arg                     args[10];

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

```
    XtCallbackProc          cb_expose();

    XColor                  color;

    XVisualInfo             v,
                            *visual_list;

    XSetWindowAttributes    attributes;

    XCharStruct             overall;

    int                     visuals_matched;

    unsigned long           mask;

/*
 *  Initialize the Toolkit.
 */

    top = XtInitialize ( argv[0], "Fast", NULL, 0, &argc, argv );

/*
 *  Save a pointer to the X Windows display structure. Save the current screen.
 */

    display = XtDisplay ( top );
    screen  = DefaultScreen ( display );

/*
 *  Initialize font information.
 */

    if ( ( font_info = XLoadQueryFont ( display, argv[1] ) ) == 0 )
        exit ( 1 );

    font = font_info->fid;

/*
 *  Save the format to use for printing out of data.
 */

    strcpy ( format, argv[2] );

/*
 *  Initialize size of string to draw.
 */

    XTextExtents ( font_info, string, strlen ( string ), &ascent,
                   &ascent, &ascent, &overall );
    width  = overall.width;
    height = overall.ascent + overall.descent;
    ascent = overall.ascent;

/*
 *  Query the X server to find out if there is the right type of visual.
 */

    v.screen = screen;
    v.depth  = 8;
    v.class  = PseudoColor;

    visual_list = XGetVisualInfo ( display,
                      VisualScreenMask | VisualDepthMask | VisualClassMask,
```

```
                    &v, &visuals_matched );

    /*
     *  Copy the required visual and free up memory allocated for by the visual
     *  query function.
     */

        memcpy ( &visual, &visual_list[0].visual, sizeof ( visual ) );
        XFree ( visual_list );

    /*
     *  Create the main window and a menu bar.
     */

        XtManageChild ( m_main = XmCreateMainWindow ( top, "", NULL, 0 ) );
        XtManageChild ( mb_main = XmCreateMenuBar ( m_main, "", NULL, 0 ) );

    /*
     *  Create menu.
     */

        mp_file = XmCreatePulldownMenu ( mb_main, "", NULL, 0 );

        i = 0;
        XtSetArg ( args[i], XmNsubMenuId, mp_file ); i++;
        XtManageChild ( XmCreateCascadeButton ( mb_main, "File", args, i ) );
        XtManageChild ( XmCreatePushButton    ( mp_file, "Exit", NULL, 0 ) );

    /*
     *  Create a scrolled window widget.
     */

        i = 0;
        XtSetArg ( args[i], XmNwidth,  COLS*(width +2)); i++;
        XtSetArg ( args[i], XmNheight, ROWS*(height+2)); i++;
        XtManageChild ( scroll =
                XmCreateScrolledWindow ( m_main, "scroll", args, i ) );

    /*
     *  Create the drawing area widget.
     */

        cb[0].callback = (XtCallbackProc)cb_expose;
        cb[0].closure  = (caddr_t)0;

        i = 0;
        XtSetArg ( args[i], XmNexposeCallback, cb             ); i++;
        XtSetArg ( args[i], XmNwidth,          COLS*(width +2)); i++;
        XtSetArg ( args[i], XmNheight,         ROWS*(height+2)); i++;
        XtManageChild ( draw =
                XmCreateDrawingArea ( scroll, "draw", args, i ) );

    /*
     *  Realize the widgets.
     */

        XtRealizeWidget ( top );

    /*
     *  Set the attributes necessary to create the actual window.
     */

        attributes.save_under        = 0;
        attributes.backing_store     = NotUseful;
```

```
    attributes.border_pixel      = BlackPixel ( display, screen );
    attributes.background_pixel  = WhitePixel ( display, screen );
    attributes.bit_gravity       = NorthWestGravity;

    mask = CWBackingStore | CWSaveUnder | CWBackPixel |
           CWBorderPixel  | CWBitGravity;

/*
 *  Create the window for the drawing area widget.
 */

    XtCreateWindow ( draw, CopyFromParent, visual, mask, &attributes );
    window = XtWindow ( draw );

/*
 *  Create the graphics context.
 */

    gc = XCreateGC ( display, window, NULL, NULL );

    XSetFont ( display, gc, font );

/*
 *  Create the pixmap.
 */

    pixmap = XCreatePixmap ( display, window, COLS*(width +2), ROWS*(height+2),
                             8 );

/*
 *  Add a time out.
 */

    XtAddTimeOut ( TIMER_VALUE, cb_expose, NULL );

/*
 *  Loop forever.
 */

    XtMainLoop ( );
}


XtCallbackProc cb_expose ( widget, closure, calldata )

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

    caddr_t closure;             /* Callback specific data.  This parameter
                                  * indicates the selected function.
                                  */

    XmDrawingAreaCallbackStruct *calldata;
                                 /* Specifies any callback-specific data the
                                  * widget needs to pass to the client.
                                  */
{
    register int        x, y;

    static double       value = 1000.00001;

    static int          count=1;

/*
```

```
     *  Update the string.
     */

        value += (double)0.00001;

    /*
     *  Write out the strings.
     */

        for ( y = 0; y < ROWS; y++ )
            for ( x = 0; x < COLS; x++ ) {
                sprintf ( string, format, value );
                XDrawImageString ( display, pixmap, gc, x*(width+2),
                                   ascent + (y*(height+2)),
                                   string, strlen ( string ) );
            }

        XCopyArea ( display, pixmap, window, gc, 0, 0,
                    COLS*(width +2), ROWS*(height+2), 0, 0 );

    /*
     *  Flush the buffer immediately.
     */

        XFlush ( display );

    /*
     *  Test and increment counter
     */

        if (count > CYCLES)
            exit(1);
        count++;


    /*
     *  Reset the timer.
     */

        XtAddTimeOut ( TIMER_VALUE, cb_expose, NULL );
    }
```

```
/*
    Pixmap/XCopyArea
    No graphics context switching
    No text formatting
 */

#include <stdio.h>
#include <string.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Cardinals.h>
#include <X11/Shell.h>
#include <X11/MwmUtil.h>
#include <Xm/Xm.h>
#include <Xm/DialogS.h>
#include <Xm/DrawingA.h>
#include <Xm/MainW.h>
#include <Xm/RowColumn.h>


#define CYCLES      100
#define TIMER_VALUE 1000
#define ROWS        50
#define COLS        10

Display             *display;

Visual              *visual;

Pixmap              pixmap;

GC                  gc;

Font                font;

Window              window;

XFontStruct         *font_info;

int                 screen;
                    ascent,
                    width,
                    height;

char                string[] = "1000.00001",
                    format[10];


int main ( argc, argv )

    int     argc;

    char    *argv[];
{
    register int            i;

    Widget                  top, m_main, mb_main, mp_file, scroll, draw;

    Arg                     args[10];

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

```
    XtCallbackProc          cb_expose();

    XColor                  color;

    XVisualInfo             v,
                            *visual_list;

    XSetWindowAttributes    attributes;

    XCharStruct             overall;

    int                     visuals_matched;

    unsigned long           mask;

/*
 *  Initialize the Toolkit.
 */

    top = XtInitialize ( argv[0], "Fast", NULL, 0, &argc, argv );

/*
 *  Save a pointer to the X Windows display structure. Save the current screen.
 */

    display = XtDisplay ( top );
    screen  = DefaultScreen ( display );

/*
 *  Initialize font information.
 */

    if ( ( font_info = XLoadQueryFont ( display, argv[1] ) ) == 0 )
        exit ( 1 );

    font = font_info->fid;

/*
 *  Save the format to use for printing out of data.
 */

    strcpy ( format, argv[2] );

/*
 *  Initialize size of string to draw.
 */

    XTextExtents ( font_info, string, strlen ( string ), &ascent,
                   &ascent, &ascent, &overall );
    width  = overall.width;
    height = overall.ascent + overall.descent;
    ascent = overall.ascent;

/*
 *  Query the X server to find out if there is the right type of visual.
 */

    v.screen = screen;
    v.depth  = 8;
    v.class  = PseudoColor;

    visual_list = XGetVisualInfo ( display,
                      VisualScreenMask | VisualDepthMask | VisualClassMask,
```

```
                    &v, &visuals_matched );

/*
 *  Copy the required visual and free up memory allocated for by the visual
 *  query function.
 */

    memcpy ( &visual, &visual_list[0].visual, sizeof ( visual ) );
    XFree ( visual_list );

/*
 *  Create the main window and a menu bar.
 */

    XtManageChild ( m_main = XmCreateMainWindow ( top, "", NULL, 0 ) );
    XtManageChild ( mb_main = XmCreateMenuBar ( m_main, "", NULL, 0 ) );

/*
 *  Create menu.
 */

    mp_file = XmCreatePulldownMenu ( mb_main, "", NULL, 0 );

    i = 0;
    XtSetArg ( args[i], XmNsubMenuId, mp_file ); i++;
    XtManageChild ( XmCreateCascadeButton ( mb_main, "File", args, i ) );
    XtManageChild ( XmCreatePushButton    ( mp_file, "Exit", NULL, 0 ) );

/*
 *  Create a scrolled window widget.
 */

    i = 0;
    XtSetArg ( args[i], XmNwidth,  COLS*(width +2)); i++;
    XtSetArg ( args[i], XmNheight, ROWS*(height+2)); i++;
    XtManageChild ( scroll =
            XmCreateScrolledWindow ( m_main, "scroll", args, i ) );

/*
 *  Create the drawing area widget.
 */

    cb[0].callback = (XtCallbackProc)cb_expose;
    cb[0].closure  = (caddr_t)0;

    i = 0;
    XtSetArg ( args[i], XmNexposeCallback, cb             ); i++;
    XtSetArg ( args[i], XmNwidth,          COLS*(width +2)); i++;
    XtSetArg ( args[i], XmNheight,         ROWS*(height+2)); i++;
    XtManageChild ( draw =
            XmCreateDrawingArea ( scroll, "draw", args, i ) );

/*
 *  Realize the widgets.
 */

    XtRealizeWidget ( top );

/*
 *  Set the attributes necessary to create the actual window.
 */

    attributes.save_under       = 0;
    attributes.backing_store    = NotUseful;
```

```
        attributes.border_pixel      = BlackPixel ( display, screen );
        attributes.background_pixel  = WhitePixel ( display, screen );
        attributes.bit_gravity       = NorthWestGravity;

        mask = CWBackingStore | CWSaveUnder | CWBackPixel |
               CWBorderPixel  | CWBitGravity;

    /*
     *  Create the window for the drawing area widget.
     */

        XtCreateWindow ( draw, CopyFromParent, visual, mask, &attributes );
        window = XtWindow ( draw );

    /*
     *  Create the graphics context.
     */

        gc = XCreateGC ( display, window, NULL, NULL );

        XSetFont ( display, gc, font );

    /*
     *  Create the pixmap used to retain a copy of the image for refreshing
     *  the window.
     */

        pixmap = XCreatePixmap ( display, window, COLS*(width +2), ROWS*(height+2),
                                 8 );

    /*
     *  Add a time out.
     */

        XtAddTimeOut ( TIMER_VALUE, cb_expose, NULL );

    /*
     *  Loop forever.
     */

        XtMainLoop ( );
}


XtCallbackProc cb_expose ( widget, closure, calldata )

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

    caddr_t closure;            /* Callback specific data.  This parameter
                                 * indicates the selected function.
                                 */
    XmDrawingAreaCallbackStruct *calldata;
                                /* Specifies any callback-specific data the
                                 * widget needs to pass to the client.
                                 */
{
    register int        x, y;

    static double       value = 1000.00001;

    static int          count=1;
```

```
/*
 *  Update the string.
 */

    value += (double)0.00001;

/*
 *  Write out the strings.
 */

    for ( y = 0; y < ROWS; y++ )
        for ( x = 0; x < COLS; x++ ) {
            XDrawImageString ( display, pixmap, gc, x*(width+2),
                               ascent + (y*(height+2)),
                               string, strlen ( string ) );

        }

    XCopyArea ( display, pixmap, window, gc, 0, 0,
                COLS*(width +2), ROWS*(height+2), 0, 0 );


/*
 *  Flush the buffer immediately.
 */

    XFlush ( display );

/*
 *  Test and increment counter
 */

    if (count > CYCLES)
        exit(1);
    count++;


/*
 *  Reset the timer.
 */

    XtAddTimeOut ( TIMER_VALUE, cb_expose, NULL );
}
```

```
/*
    Draw straight to screen using XDrawImageString
    1 graphics context switch/cycle
    %10.5f text formatting
 */

#include <stdio.h>
#include <string.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Cardinals.h>
#include <X11/Shell.h>
#include <X11/MwmUtil.h>
#include <Xm/Xm.h>
#include <Xm/DialogS.h>
#include <Xm/DrawingA.h>
#include <Xm/MainW.h>
#include <Xm/RowColumn.h>


#define CYCLES       100
#define TIMER_VALUE 1000
#define ROWS         50
#define COLS         10

Display              *display;

Visual               *visual;

Pixmap               pixmap;

GC                   gc[5];

Font                 font;

Window               window;

XFontStruct          *font_info;

int                  screen;
                     ascent,
                     width,
                     height;

char                 string[] = "1000.00001",
                     format[10];


int main ( argc, argv )

    int      argc;

    char     *argv[];
{
    register int             i;

    Widget                   top, m_main, mb_main, mp_file, scroll, draw;

    Arg                      args[10];

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

```
    XtCallbackProc          cb_expose();

    XColor                  color;

    Colormap                xcmap;

    XVisualInfo             v,
                            *visual_list;

    XSetWindowAttributes    attributes;

    XCharStruct             overall;

    int                     visuals_matched;

    unsigned long           mask;


    int                     z;


/*
 *  Initialize the Toolkit.
 */

    top = XtInitialize ( argv[0], "Fast", NULL, 0, &argc, argv );

/*
 *  Save a pointer to the X Windows display structure. Save the current screen.
 */

    display = XtDisplay ( top );
    screen  = DefaultScreen ( display );

/*
 *  Initialize font information.
 */

    if ( ( font_info = XLoadQueryFont ( display, argv[1] ) ) == 0 )
        exit ( 1 );

    font = font_info->fid;

/*
 *  Save the format to use for printing out of data.
 */

    strcpy ( format, argv[2] );

/*
 *  Initialize size of string to draw.
 */

    XTextExtents ( font_info, string, strlen ( string ), &ascent,
                   &ascent, &ascent, &overall );
    width  = overall.width;
    height = overall.ascent + overall.descent;
    ascent = overall.ascent;

/*
 *  Query the X server to find out if there is the right type of visual.
 */
```

```
    v.screen = screen;
    v.depth  = 8;
    v.class  = PseudoColor;

    visual_list = XGetVisualInfo ( display,
                      VisualScreenMask | VisualDepthMask | VisualClassMask,
                      &v, &visuals_matched );

/*
 *  Copy the required visual and free up memory allocated for by the visual
 *  query function.
 */

    memcpy ( &visual, &visual_list[0].visual, sizeof ( visual ) );
    XFree ( visual_list );

/*
 *  Create the main window and a menu bar.
 */

    XtManageChild ( m_main = XmCreateMainWindow ( top, "", NULL, 0 ) );
    XtManageChild ( mb_main = XmCreateMenuBar ( m_main, "", NULL, 0 ) );

/*
 *  Create menu.
 */

    mp_file = XmCreatePulldownMenu ( mb_main, "", NULL, 0 );

    i = 0;
    XtSetArg ( args[i], XmNsubMenuId, mp_file ); i++;
    XtManageChild ( XmCreateCascadeButton ( mb_main, "File", args, i ) );
    XtManageChild ( XmCreatePushButton    ( mp_file, "Exit", NULL, 0 ) );

/*
 *  Create a scrolled window widget.
 */

    i = 0;
    XtSetArg ( args[i], XmNwidth,  COLS*(width +2)); i++;
    XtSetArg ( args[i], XmNheight, ROWS*(height+2)); i++;
    XtManageChild ( scroll =
            XmCreateScrolledWindow ( m_main, "scroll", args, i ) );

/*
 *  Create the drawing area widget.
 */

    cb[0].callback = (XtCallbackProc)cb_expose;
    cb[0].closure  = (caddr_t)0;

    i = 0;
    XtSetArg ( args[i], XmNexposeCallback, cb             ); i++;
    XtSetArg ( args[i], XmNwidth,          COLS*(width +2)); i++;
    XtSetArg ( args[i], XmNheight,         ROWS*(height+2)); i++;
    XtManageChild ( draw =
            XmCreateDrawingArea ( scroll, "draw", args, i ) );

/*
 *  Realize the widgets.
 */

    XtRealizeWidget ( top );
```

```
/*
 *  Set the attributes necessary to create the actual window.
 */

    attributes.save_under         = 0;
    attributes.backing_store      = NotUseful;
    attributes.border_pixel       = BlackPixel ( display, screen );
    attributes.background_pixel   = WhitePixel ( display, screen );
    attributes.bit_gravity        = NorthWestGravity;

    mask = CWBackingStore | CWSaveUnder | CWBackPixel |
           CWBorderPixel  | CWBitGravity;

/*
 *  Create the window for the drawing area widget.
 */

    XtCreateWindow ( draw, CopyFromParent, visual, mask, &attributes );
    window = XtWindow ( draw );

    xcmap = DefaultColormap(display, screen);

/*
 *  Create the graphics context.
 */

    gc[1] = XCreateGC ( display, window, NULL, NULL );

    gc[2] = XCreateGC ( display, window, NULL, NULL );
    color.red = 65000;
    color.blue = 0;
    color.green = 0;
    if (XAllocColor(display, xcmap, &color))
        XSetBackground(display, gc[2], color.pixel);
    else
        fprintf(stderr, "couldn't allocate color");

    gc[3] = XCreateGC ( display, window, NULL, NULL );
    color.red = 0;
    color.blue = 65000;
    color.green = 0;
    if (XAllocColor(display, xcmap, &color))
        XSetBackground(display, gc[3], color.pixel);
    else
        fprintf(stderr, "couldn't allocate color");

    gc[4] = XCreateGC ( display, window, NULL, NULL );
    color.red = 0;
    color.blue = 0;
    color.green = 65000;
    if (XAllocColor(display, xcmap, &color))
        XSetBackground(display, gc[4], color.pixel);
    else
        fprintf(stderr, "couldn't allocate color");

    for (z=1; z<5; z++)
        XSetFont ( display, gc[z], font );

/*
 *  Create the pixmap used to retain a copy of the image for refreshing
 *  the window.
 */

    pixmap = XCreatePixmap ( display, window, COLS*(width +2), ROWS*(height+2),
```

```
                                8 );

    /*
     *  Add a time out.
     */

        XtAddTimeOut ( TIMER_VALUE, cb_expose, NULL );

    /*
     *  Loop forever.
     */

        XtMainLoop ( );
    }


    XtCallbackProc cb_expose ( widget, closure, calldata )

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

        caddr_t closure;            /* Callback specific data.  This parameter
                                     * indicates the selected function.
                                     */
        XmDrawingAreaCallbackStruct *calldata;
                                    /* Specifies any callback-specific data the
                                     * widget needs to pass to the client.
                                     */
    {
        register int        x, y;

        static double       value = 1000.00001;

        static int          count=1;

        int                 z;

    /*
     *  Update the string.
     */

        value += (double)0.00001;

    /*
     *  Determine which gc to use
     */

        z = (count % 4) + 1;

    /*
     *  Write out the strings.
     */

        for ( y = 0; y < ROWS; y++ )
            for ( x = 0; x < COLS; x++ ) {
                sprintf ( string, format, value );
                XDrawImageString ( display, window, gc[z], x*(width+2), ascent + (y*(height+2)
    ),
                                   string, strlen ( string ) );
            }

    /*
     *  Flush the buffer immediately.
```

```
 */

    XFlush ( display );

/*
 *  Test and increment counter
 */

    if (count > CYCLES)
        exit(1);
    count++;


/*
 *  Reset the timer.
 */

    XtAddTimeOut ( TIMER_VALUE, cb_expose, NULL );
}
```

```
/*
    Draw straight to screen using XDrawImageString
    Multiple graphics context switches/cycle
    %10.5f text formatting
 */

#include <stdio.h>
#include <string.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Cardinals.h>
#include <X11/Shell.h>
#include <X11/MwmUtil.h>
#include <Xm/Xm.h>
#include <Xm/DialogS.h>
#include <Xm/DrawingA.h>
#include <Xm/MainW.h>
#include <Xm/RowColumn.h>


#define CYCLES      100
#define TIMER_VALUE 1000
#define ROWS        50
#define COLS        10

Display             *display;

Visual              *visual;

Pixmap              pixmap;

GC                  gc[5];

Font                font;

Window              window;

XFontStruct         *font_info;

int                 screen;
                    ascent,
                    width,
                    height;

char                string[] = "1000.00001",
                    format[10];


int main ( argc, argv )

    int     argc;

    char    *argv[];
{
    register int            i;

    Widget                  top, m_main, mb_main, mp_file, scroll, draw;

    Arg                     args[10];

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

```
    XtCallbackProc          cb_expose();
    XColor                  color;
    Colormap                xcmap;
    XVisualInfo             v,
                            *visual_list;
    XSetWindowAttributes    attributes;
    XCharStruct             overall;
    int                     visuals_matched;
    unsigned long           mask;
    int                     z;

/*
 *  Initialize the Toolkit.
 */
    top = XtInitialize ( argv[0], "Fast", NULL, 0, &argc, argv );

/*
 *  Save a pointer to the X Windows display structure. Save the current screen.
 */
    display = XtDisplay ( top );
    screen  = DefaultScreen ( display );

/*
 *  Initialize font information.
 */
    if ( ( font_info = XLoadQueryFont ( display, argv[1] ) ) == 0 )
        exit ( 1 );

    font = font_info->fid;

/*
 *  Save the format to use for printing out of data.
 */
    strcpy ( format, argv[2] );

/*
 *  Initialize size of string to draw.
 */
    XTextExtents ( font_info, string, strlen ( string ), &ascent,
                   &ascent, &ascent, &overall );
    width  = overall.width;
    height = overall.ascent + overall.descent;
    ascent = overall.ascent;

/*
 *  Query the X server to find out if there is the right type of visual.
 */
```

```
        v.screen = screen;
        v.depth  = 8;
        v.class  = PseudoColor;

        visual_list = XGetVisualInfo ( display,
                          VisualScreenMask | VisualDepthMask | VisualClassMask,
                          &v, &visuals_matched );

    /*
     *  Copy the required visual and free up memory allocated for by the visual
     *  query function.
     */

        memcpy ( &visual, &visual_list[0].visual, sizeof ( visual ) );
        XFree ( visual_list );

    /*
     *  Create the main window and a menu bar.
     */

        XtManageChild ( m_main = XmCreateMainWindow ( top, "", NULL, 0 ) );
        XtManageChild ( mb_main = XmCreateMenuBar ( m_main, "", NULL, 0 ) );

    /*
     *  Create menu.
     */

        mp_file = XmCreatePulldownMenu ( mb_main, "", NULL, 0 );

        i = 0;
        XtSetArg ( args[i], XmNsubMenuId, mp_file ); i++;
        XtManageChild ( XmCreateCascadeButton ( mb_main, "File", args, i ) );
        XtManageChild ( XmCreatePushButton    ( mp_file, "Exit", NULL, 0 ) );

    /*
     *  Create a scrolled window widget.
     */

        i = 0;
        XtSetArg ( args[i], XmNwidth,  COLS*(width +2)); i++;
        XtSetArg ( args[i], XmNheight, ROWS*(height+2)); i++;
        XtManageChild ( scroll =
                XmCreateScrolledWindow ( m_main, "scroll", args, i ) );

    /*
     *  Create the drawing area widget.
     */

        cb[0].callback = (XtCallbackProc)cb_expose;
        cb[0].closure  = (caddr_t)0;

        i = 0;
        XtSetArg ( args[i], XmNexposeCallback, cb             ); i++;
        XtSetArg ( args[i], XmNwidth,          COLS*(width +2)); i++;
        XtSetArg ( args[i], XmNheight,         ROWS*(height+2)); i++;
        XtManageChild ( draw =
                XmCreateDrawingArea ( scroll, "draw", args, i ) );

    /*
     *  Realize the widgets.
     */

        XtRealizeWidget ( top );
```

```
/*
 *  Set the attributes necessary to create the actual window.
 */

    attributes.save_under         = 0;
    attributes.backing_store      = NotUseful;
    attributes.border_pixel       = BlackPixel ( display, screen );
    attributes.background_pixel   = WhitePixel ( display, screen );
    attributes.bit_gravity        = NorthWestGravity;

    mask = CWBackingStore | CWSaveUnder | CWBackPixel |
           CWBorderPixel  | CWBitGravity;

/*
 *  Create the window for the drawing area widget.
 */

    XtCreateWindow ( draw, CopyFromParent, visual, mask, &attributes );
    window = XtWindow ( draw );

    xcmap = DefaultColormap(display, screen);

/*
 *  Create the graphics context.
 */

    gc[1] = XCreateGC ( display, window, NULL, NULL );

    gc[2] = XCreateGC ( display, window, NULL, NULL );
    color.red = 65000;
    color.blue = 0;
    color.green = 0;
    if (XAllocColor(display, xcmap, &color))
        XSetBackground(display, gc[2], color.pixel);
    else
        fprintf(stderr, "couldn't allocate color");

    gc[3] = XCreateGC ( display, window, NULL, NULL );
    color.red = 0;
    color.blue = 65000;
    color.green = 0;
    if (XAllocColor(display, xcmap, &color))
        XSetBackground(display, gc[3], color.pixel);
    else
        fprintf(stderr, "couldn't allocate color");

    gc[4] = XCreateGC ( display, window, NULL, NULL );
    color.red = 0;
    color.blue = 0;
    color.green = 65000;
    if (XAllocColor(display, xcmap, &color))
        XSetBackground(display, gc[4], color.pixel);
    else
        fprintf(stderr, "couldn't allocate color");

    for (z=1; z<5; z++)
        XSetFont ( display, gc[z], font );

/*
 *  Create the pixmap used to retain a copy of the image for refreshing
 *  the window.
 */

    pixmap = XCreatePixmap ( display, window, COLS*(width +2), ROWS*(height+2),
```

```
                                8 );

    /*
     *  Add a time out.
     */

        XtAddTimeOut ( TIMER_VALUE, cb_expose, NULL );

    /*
     *  Loop forever.
     */

        XtMainLoop ( );
    }


    XtCallbackProc cb_expose ( widget, closure, calldata )

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

        caddr_t closure;            /* Callback specific data.  This parameter
                                     * indicates the selected function.
                                     */
        XmDrawingAreaCallbackStruct *calldata;
                                    /* Specifies any callback-specific data the
                                     * widget needs to pass to the client.
                                     */
    {
        register int        x, y;

        static double       value = 1000.00001;

        static int          count=1;

        int                 z;

    /*
     *  Update the string.
     */

        value += (double)0.00001;

    /*
     *  Write out the strings.
     */

        for ( y = 0; y < ROWS; y++ )
            for ( x = 0; x < COLS; x++ ) {
                z = ((x + y) % 4) + 1;
                sprintf ( string, format, value );
                XDrawImageString ( display, window, gc[z], x*(width+2),
                        ascent + (y*(height+2)), string, strlen ( string ) );
            }

    /*
     *  Flush the buffer immediately.
     */

        XFlush ( display );

    /*
     *  Test and increment counter
```

```
    */

        if (count > CYCLES)
            exit(1);
        count++;


    /*
     *  Reset the timer.
     */

        XtAddTimeOut ( TIMER_VALUE, cb_expose, NULL );
    }
```

```
/*
    Draw straight to screen using XDrawImageString
    No graphics context switching
    %10.5f text formatting
 */

#include <stdio.h>
#include <string.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Cardinals.h>
#include <X11/Shell.h>
#include <X11/MwmUtil.h>
#include <Xm/Xm.h>
#include <Xm/DialogS.h>
#include <Xm/DrawingA.h>
#include <Xm/MainW.h>
#include <Xm/RowColumn.h>


#define CYCLES      100
#define TIMER_VALUE 1000
#define ROWS        50
#define COLS        10

Display             *display;

Visual              *visual;

Pixmap              pixmap;

GC                  gc;

Font                font;

Window              window;

XFontStruct         *font_info;

int                 screen;
                    ascent,
                    width,
                    height;

char                string[] = "1000.00001",
                    format[10];


int main ( argc, argv )

    int     argc;

    char    *argv[];
{
    register int            i;

    Widget                  top, m_main, mb_main, mp_file, scroll, draw;

    Arg                     args[10];

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

```
    XtCallbackProc          cb_expose();

    XColor                  color;

    XVisualInfo             v,
                            *visual_list;

    XSetWindowAttributes    attributes;

    XCharStruct             overall;

    int                     visuals_matched;

    unsigned long           mask;

/*
 *  Initialize the Toolkit.
 */

    top = XtInitialize ( argv[0], "Fast", NULL, 0, &argc, argv );

/*
 *  Save a pointer to the X Windows display structure. Save the current screen.
 */

    display = XtDisplay ( top );
    screen  = DefaultScreen ( display );

/*
 *  Initialize font information.
 */

    if ( ( font_info = XLoadQueryFont ( display, argv[1] ) ) == 0 )
        exit ( 1 );

    font = font_info->fid;

/*
 *  Save the format to use for printing out of data.
 */

    strcpy ( format, argv[2] );

/*
 *  Initialize size of string to draw.
 */

    XTextExtents ( font_info, string, strlen ( string ), &ascent,
                   &ascent, &ascent, &overall );
    width  = overall.width;
    height = overall.ascent + overall.descent;
    ascent = overall.ascent;

/*
 *  Query the X server to find out if there is the right type of visual.
 */

    v.screen = screen;
    v.depth  = 8;
    v.class  = PseudoColor;

    visual_list = XGetVisualInfo ( display,
                      VisualScreenMask | VisualDepthMask | VisualClassMask,
```

```
                    &v, &visuals_matched );

/*
 *  Copy the required visual and free up memory allocated for by the visual
 *  query function.
 */

    memcpy ( &visual, &visual_list[0].visual, sizeof ( visual ) );
    XFree ( visual_list );

/*
 *  Create the main window and a menu bar.
 */

    XtManageChild ( m_main = XmCreateMainWindow ( top, "", NULL, 0 ) );
    XtManageChild ( mb_main = XmCreateMenuBar ( m_main, "", NULL, 0 ) );

/*
 *  Create menu.
 */

    mp_file = XmCreatePulldownMenu ( mb_main, "", NULL, 0 );

    i = 0;
    XtSetArg ( args[i], XmNsubMenuId, mp_file ); i++;
    XtManageChild ( XmCreateCascadeButton ( mb_main, "File", args, i ) );
    XtManageChild ( XmCreatePushButton    ( mp_file, "Exit", NULL, 0 ) );

/*
 *  Create a scrolled window widget.
 */

    i = 0;
    XtSetArg ( args[i], XmNwidth,  COLS*(width +2)); i++;
    XtSetArg ( args[i], XmNheight, ROWS*(height+2)); i++;
    XtManageChild ( scroll =
            XmCreateScrolledWindow ( m_main, "scroll", args, i ) );

/*
 *  Create the drawing area widget.
 */

    cb[0].callback = (XtCallbackProc)cb_expose;
    cb[0].closure  = (caddr_t)0;

    i = 0;
    XtSetArg ( args[i], XmNexposeCallback, cb             ); i++;
    XtSetArg ( args[i], XmNwidth,          COLS*(width +2)); i++;
    XtSetArg ( args[i], XmNheight,         ROWS*(height+2)); i++;
    XtManageChild ( draw =
            XmCreateDrawingArea ( scroll, "draw", args, i ) );

/*
 *  Realize the widgets.
 */

    XtRealizeWidget ( top );

/*
 *  Set the attributes necessary to create the actual window.
 */

    attributes.save_under        = 0;
    attributes.backing_store     = NotUseful;
```

```
    attributes.border_pixel      = BlackPixel ( display, screen );
    attributes.background_pixel  = WhitePixel ( display, screen );
    attributes.bit_gravity       = NorthWestGravity;

    mask = CWBackingStore | CWSaveUnder | CWBackPixel |
           CWBorderPixel  | CWBitGravity;

/*
 *  Create the window for the drawing area widget.
 */

    XtCreateWindow ( draw, CopyFromParent, visual, mask, &attributes );
    window = XtWindow ( draw );

/*
 *  Create the graphics context.
 */

    gc = XCreateGC ( display, window, NULL, NULL );

    XSetFont ( display, gc, font );

/*
 *  Create the pixmap used to retain a copy of the image for refreshing
 *  the window.
 */

    pixmap = XCreatePixmap ( display, window, COLS*(width +2), ROWS*(height+2),
                             8 );

/*
 *  Add a time out.
 */

    XtAddTimeOut ( TIMER_VALUE, cb_expose, NULL );

/*
 *  Loop forever.
 */

    XtMainLoop ( );
}


XtCallbackProc cb_expose ( widget, closure, calldata )

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

    caddr_t closure;             /* Callback specific data.  This parameter
                                  * indicates the selected function.
                                  */
    XmDrawingAreaCallbackStruct *calldata;
                                 /* Specifies any callback-specific data the
                                  * widget needs to pass to the client.
                                  */
{
    register int        x, y;

    static double       value = 1000.00001;

    static int          count=1;
```

```
    /*
     *  Update the string.
     */

        value += (double)0.00001;

    /*
     *  Write out the strings.
     */

        for ( y = 0; y < ROWS; y++ )
            for ( x = 0; x < COLS; x++ ) {
                sprintf ( string, format, value );
                XDrawImageString ( display, window, gc, x*(width+2), ascent + (y*(height+2)),
                                   string, strlen ( string ) );
            }

    /*
     *  Flush the buffer immediately.
     */

        XFlush ( display );

    /*
     *  Test and increment counter
     */

        if (count > CYCLES)
            exit(1);
        count++;


    /*
     *  Reset the timer.
     */

        XtAddTimeOut ( TIMER_VALUE, cb_expose, NULL );
    }
```

```
/*
    Draw straight to screen using XDrawImageString
    1 gc switch/cycle
    integer formatting
 */

#include <stdio.h>
#include <string.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Cardinals.h>
#include <X11/Shell.h>
#include <X11/MwmUtil.h>
#include <Xm/Xm.h>
#include <Xm/DialogS.h>
#include <Xm/DrawingA.h>
#include <Xm/MainW.h>
#include <Xm/RowColumn.h>


#define CYCLES      100
#define TIMER_VALUE 1000
#define ROWS        50
#define COLS        10

Display             *display;

Visual              *visual;

Pixmap              pixmap;

GC                  gc[5];

Font                font;

Window              window;

XFontStruct         *font_info;

int                 screen;
                    ascent,
                    width,
                    height;

char                string[] = "1000.00001",
                    format[10];


int main ( argc, argv )

    int     argc;

    char    *argv[];
{
    register int            i;

    Widget                  top, m_main, mb_main, mp_file, scroll, draw;

    Arg                     args[10];

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

```
    XtCallbackProc          cb_expose();

    XColor                  color;

    Colormap                xcmap;

    XVisualInfo             v,
                            *visual_list;

    XSetWindowAttributes    attributes;

    XCharStruct             overall;

    int                     visuals_matched;

    unsigned long           mask;


    int                     z;


/*
 *  Initialize the Toolkit.
 */

    top = XtInitialize ( argv[0], "Fast", NULL, 0, &argc, argv );

/*
 *  Save a pointer to the X Windows display structure. Save the current screen.
 */

    display = XtDisplay ( top );
    screen  = DefaultScreen ( display );

/*
 *  Initialize font information.
 */

    if ( ( font_info = XLoadQueryFont ( display, argv[1] ) ) == 0 )
        exit ( 1 );

    font = font_info->fid;

/*
 *  Save the format to use for printing out of data.
 */

    strcpy ( format, argv[2] );

/*
 *  Initialize size of string to draw.
 */

    XTextExtents ( font_info, string, strlen ( string ), &ascent,
                   &ascent, &ascent, &overall );
    width  = overall.width;
    height = overall.ascent + overall.descent;
    ascent = overall.ascent;

/*
 *  Query the X server to find out if there is the right type of visual.
 */
```

```
    v.screen = screen;
    v.depth  = 8;
    v.class  = PseudoColor;

    visual_list = XGetVisualInfo ( display,
                      VisualScreenMask | VisualDepthMask | VisualClassMask,
                      &v, &visuals_matched );

/*
 *  Copy the required visual and free up memory allocated for by the visual
 *  query function.
 */

    memcpy ( &visual, &visual_list[0].visual, sizeof ( visual ) );
    XFree ( visual_list );

/*
 *  Create the main window and a menu bar.
 */

    XtManageChild ( m_main = XmCreateMainWindow ( top, "", NULL, 0 ) );
    XtManageChild ( mb_main = XmCreateMenuBar ( m_main, "", NULL, 0 ) );

/*
 *  Create menu.
 */

    mp_file = XmCreatePulldownMenu ( mb_main, "", NULL, 0 );

    i = 0;
    XtSetArg ( args[i], XmNsubMenuId, mp_file ); i++;
    XtManageChild ( XmCreateCascadeButton ( mb_main, "File", args, i ) );
    XtManageChild ( XmCreatePushButton    ( mp_file, "Exit", NULL, 0 ) );

/*
 *  Create a scrolled window widget.
 */

    i = 0;
    XtSetArg ( args[i], XmNwidth,  COLS*(width +2)); i++;
    XtSetArg ( args[i], XmNheight, ROWS*(height+2)); i++;
    XtManageChild ( scroll =
            XmCreateScrolledWindow ( m_main, "scroll", args, i ) );

/*
 *  Create the drawing area widget.
 */

    cb[0].callback = (XtCallbackProc)cb_expose;
    cb[0].closure  = (caddr_t)0;

    i = 0;
    XtSetArg ( args[i], XmNexposeCallback, cb             ); i++;
    XtSetArg ( args[i], XmNwidth,          COLS*(width +2)); i++;
    XtSetArg ( args[i], XmNheight,         ROWS*(height+2)); i++;
    XtManageChild ( draw =
            XmCreateDrawingArea ( scroll, "draw", args, i ) );

/*
 *  Realize the widgets.
 */

    XtRealizeWidget ( top );
```

```
/*
 *  Set the attributes necessary to create the actual window.
 */

    attributes.save_under        = 0;
    attributes.backing_store     = NotUseful;
    attributes.border_pixel      = BlackPixel ( display, screen );
    attributes.background_pixel  = WhitePixel ( display, screen );
    attributes.bit_gravity       = NorthWestGravity;

    mask = CWBackingStore | CWSaveUnder | CWBackPixel |
           CWBorderPixel  | CWBitGravity;

/*
 *  Create the window for the drawing area widget.
 */

    XtCreateWindow ( draw, CopyFromParent, visual, mask, &attributes );
    window = XtWindow ( draw );

    xcmap = DefaultColormap(display, screen);

/*
 *  Create the graphics context.
 */

    gc[1] = XCreateGC ( display, window, NULL, NULL );

    gc[2] = XCreateGC ( display, window, NULL, NULL );
    color.red = 65000;
    color.blue = 0;
    color.green = 0;
    if (XAllocColor(display, xcmap, &color))
        XSetBackground(display, gc[2], color.pixel);
    else
        fprintf(stderr, "couldn't allocate color");

    gc[3] = XCreateGC ( display, window, NULL, NULL );
    color.red = 0;
    color.blue = 65000;
    color.green = 0;
    if (XAllocColor(display, xcmap, &color))
        XSetBackground(display, gc[3], color.pixel);
    else
        fprintf(stderr, "couldn't allocate color");

    gc[4] = XCreateGC ( display, window, NULL, NULL );
    color.red = 0;
    color.blue = 0;
    color.green = 65000;
    if (XAllocColor(display, xcmap, &color))
        XSetBackground(display, gc[4], color.pixel);
    else
        fprintf(stderr, "couldn't allocate color");

    for (z=1; z<5; z++)
        XSetFont ( display, gc[z], font );

/*
 *  Create the pixmap used to retain a copy of the image for refreshing
 *  the window.
 */

    pixmap = XCreatePixmap ( display, window, COLS*(width +2), ROWS*(height+2),
```

```
                                8 );

    /*
     *  Add a time out.
     */

        XtAddTimeOut ( TIMER_VALUE, cb_expose, NULL );

    /*
     *  Loop forever.
     */

        XtMainLoop ( );
    }


    XtCallbackProc cb_expose ( widget, closure, calldata )

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

        caddr_t closure;            /* Callback specific data.  This parameter
                                     * indicates the selected function.
                                     */

        XmDrawingAreaCallbackStruct *calldata;
                                    /* Specifies any callback-specific data the
                                     * widget needs to pass to the client.
                                     */
    {
        register int        x, y;

        static long         value = 1000;

        static int          count=1;

        int                 z;

    /*
     *  Update the string.
     */

        value += 1;

    /*
     *  Determine which gc to use
     */

        z = (count % 4) + 1;

    /*
     *  Write out the strings.
     */

        for ( y = 0; y < ROWS; y++ )
            for ( x = 0; x < COLS; x++ ) {
                sprintf ( string, format, value );
                XDrawImageString ( display, window, gc[z], x*(width+2),
                                   ascent + (y*(height+2)),
                                   string, strlen ( string ) );
            }

    /*
     *  Flush the buffer immediately.
```

```
 */

    XFlush ( display );

/*
 *  Test and increment counter
 */

    if (count > CYCLES)
        exit(1);
    count++;


/*
 *  Reset the timer.
 */

    XtAddTimeOut ( TIMER_VALUE, cb_expose, NULL );
}
```

```
/*
    Draw straight to screen using XDrawImageString
    multiple gc switches/cycle
    integer formatting
 */

#include <stdio.h>
#include <string.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Cardinals.h>
#include <X11/Shell.h>
#include <X11/MwmUtil.h>
#include <Xm/Xm.h>
#include <Xm/DialogS.h>
#include <Xm/DrawingA.h>
#include <Xm/MainW.h>
#include <Xm/RowColumn.h>


#define CYCLES      100
#define TIMER_VALUE 1000
#define ROWS        50
#define COLS        10

Display             *display;

Visual              *visual;

Pixmap              pixmap;

GC                  gc[5];

Font                font;

Window              window;

XFontStruct         *font_info;

int                 screen;
                    ascent,
                    width,
                    height;

char                string[] = "1000.00001",
                    format[10];


int main ( argc, argv )

    int     argc;

    char    *argv[];
{
    register int            i;

    Widget                  top, m_main, mb_main, mp_file, scroll, draw;

    Arg                     args[10];

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

```
    XtCallbackProc          cb_expose();

    XColor                  color;

    Colormap                xcmap;

    XVisualInfo             v,
                            *visual_list;

    XSetWindowAttributes    attributes;

    XCharStruct             overall;

    int                     visuals_matched;

    unsigned long           mask;


    int                     z;


/*
 *  Initialize the Toolkit.
 */

    top = XtInitialize ( argv[0], "Fast", NULL, 0, &argc, argv );

/*
 *  Save a pointer to the X Windows display structure. Save the current screen.
 */

    display = XtDisplay ( top );
    screen  = DefaultScreen ( display );

/*
 *  Initialize font information.
 */

    if ( ( font_info = XLoadQueryFont ( display, argv[1] ) ) == 0 )
        exit ( 1 );

    font = font_info->fid;

/*
 *  Save the format to use for printing out of data.
 */

    strcpy ( format, argv[2] );

/*
 *  Initialize size of string to draw.
 */

    XTextExtents ( font_info, string, strlen ( string ), &ascent,
                   &ascent, &ascent, &overall );
    width  = overall.width;
    height = overall.ascent + overall.descent;
    ascent = overall.ascent;

/*
 *  Query the X server to find out if there is the right type of visual.
 */
```

```
    v.screen = screen;
    v.depth  = 8;
    v.class  = PseudoColor;

    visual_list = XGetVisualInfo ( display,
                        VisualScreenMask | VisualDepthMask | VisualClassMask,
                        &v, &visuals_matched );

/*
 *  Copy the required visual and free up memory allocated for by the visual
 *  query function.
 */

    memcpy ( &visual, &visual_list[0].visual, sizeof ( visual ) );
    XFree ( visual_list );

/*
 *  Create the main window and a menu bar.
 */

    XtManageChild ( m_main = XmCreateMainWindow ( top, "", NULL, 0 ) );
    XtManageChild ( mb_main = XmCreateMenuBar ( m_main, "", NULL, 0 ) );

/*
 *  Create menu.
 */

    mp_file = XmCreatePulldownMenu ( mb_main, "", NULL, 0 );

    i = 0;
    XtSetArg ( args[i], XmNsubMenuId, mp_file ); i++;
    XtManageChild ( XmCreateCascadeButton ( mb_main, "File", args, i ) );
    XtManageChild ( XmCreatePushButton    ( mp_file, "Exit", NULL, 0 ) );

/*
 *  Create a scrolled window widget.
 */

    i = 0;
    XtSetArg ( args[i], XmNwidth,  COLS*(width +2)); i++;
    XtSetArg ( args[i], XmNheight, ROWS*(height+2)); i++;
    XtManageChild ( scroll =
            XmCreateScrolledWindow ( m_main, "scroll", args, i ) );

/*
 *  Create the drawing area widget.
 */

    cb[0].callback = (XtCallbackProc)cb_expose;
    cb[0].closure  = (caddr_t)0;

    i = 0;
    XtSetArg ( args[i], XmNexposeCallback, cb             ); i++;
    XtSetArg ( args[i], XmNwidth,          COLS*(width +2)); i++;
    XtSetArg ( args[i], XmNheight,         ROWS*(height+2)); i++;
    XtManageChild ( draw =
            XmCreateDrawingArea ( scroll, "draw", args, i ) );

/*
 *  Realize the widgets.
 */

    XtRealizeWidget ( top );
```

```
/*
 *  Set the attributes necessary to create the actual window.
 */

    attributes.save_under         = 0;
    attributes.backing_store      = NotUseful;
    attributes.border_pixel       = BlackPixel ( display, screen );
    attributes.background_pixel   = WhitePixel ( display, screen );
    attributes.bit_gravity        = NorthWestGravity;

    mask = CWBackingStore | CWSaveUnder | CWBackPixel |
           CWBorderPixel  | CWBitGravity;

/*
 *  Create the window for the drawing area widget.
 */

    XtCreateWindow ( draw, CopyFromParent, visual, mask, &attributes );
    window = XtWindow ( draw );

    xcmap = DefaultColormap(display, screen);

/*
 *  Create the graphics context.
 */

    gc[1] = XCreateGC ( display, window, NULL, NULL );

    gc[2] = XCreateGC ( display, window, NULL, NULL );
    color.red = 65000;
    color.blue = 0;
    color.green = 0;
    if (XAllocColor(display, xcmap, &color))
        XSetBackground(display, gc[2], color.pixel);
    else
        fprintf(stderr, "couldn't allocate color");

    gc[3] = XCreateGC ( display, window, NULL, NULL );
    color.red = 0;
    color.blue = 65000;
    color.green = 0;
    if (XAllocColor(display, xcmap, &color))
        XSetBackground(display, gc[3], color.pixel);
    else
        fprintf(stderr, "couldn't allocate color");

    gc[4] = XCreateGC ( display, window, NULL, NULL );
    color.red = 0;
    color.blue = 0;
    color.green = 65000;
    if (XAllocColor(display, xcmap, &color))
        XSetBackground(display, gc[4], color.pixel);
    else
        fprintf(stderr, "couldn't allocate color");

    for (z=1; z<5; z++)
        XSetFont ( display, gc[z], font );

/*
 *  Create the pixmap used to retain a copy of the image for refreshing
 *  the window.
 */

    pixmap = XCreatePixmap ( display, window, COLS*(width +2), ROWS*(height+2),
```

```
                                8 );

    /*
     *  Add a time out.
     */

        XtAddTimeOut ( TIMER_VALUE, cb_expose, NULL );

    /*
     *  Loop forever.
     */

        XtMainLoop ( );
    }


    XtCallbackProc cb_expose ( widget, closure, calldata )

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

        caddr_t closure;            /* Callback specific data.  This parameter
                                     * indicates the selected function.
                                     */

        XmDrawingAreaCallbackStruct *calldata;
                                    /* Specifies any callback-specific data the
                                     * widget needs to pass to the client.
                                     */

    {
        register int        x, y;

        static long         value = 1000;

        static int          count=1;

        int                 z;

    /*
     *  Update the string.
     */

        value += 1;

    /*
     *  Write out the strings.
     */

        for ( y = 0; y < ROWS; y++ )
            for ( x = 0; x < COLS; x++ ) {
                z = ((x + y) % 4) + 1;
                sprintf ( string, format, value );
                XDrawImageString ( display, window, gc[z], x*(width+2),
                        ascent + (y*(height+2)), string, strlen ( string ) );
            }

    /*
     *  Flush the buffer immediately.
     */

        XFlush ( display );

    /*
     *  Test and increment counter
```

```
 */

    if (count > CYCLES)
        exit(1);
    count++;


/*
 *  Reset the timer.
 */

    XtAddTimeOut ( TIMER_VALUE, cb_expose, NULL );
}
```

```
/*
    Draw straight to screen using XDrawImageString
    no gc switches
    integer text formatting
 */

#include <stdio.h>
#include <string.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Cardinals.h>
#include <X11/Shell.h>
#include <X11/MwmUtil.h>
#include <Xm/Xm.h>
#include <Xm/DialogS.h>
#include <Xm/DrawingA.h>
#include <Xm/MainW.h>
#include <Xm/RowColumn.h>


#define CYCLES      100
#define TIMER_VALUE 1000
#define ROWS        50
#define COLS        10

Display             *display;

Visual              *visual;

Pixmap              pixmap;

GC                  gc;

Font                font;

Window              window;

XFontStruct         *font_info;

int                 screen;
                    ascent,
                    width,
                    height;

char                string[] = "1000.00001",
                    format[10];


int main ( argc, argv )

    int     argc;

    char    *argv[];
{
    register int            i;

    Widget                  top, m_main, mb_main, mp_file, scroll, draw;

    Arg                     args[10];

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

```
    XtCallbackProc          cb_expose();

    XColor                  color;

    XVisualInfo             v,
                            *visual_list;

    XSetWindowAttributes    attributes;

    XCharStruct             overall;

    int                     visuals_matched;

    unsigned long           mask;

/*
 *  Initialize the Toolkit.
 */

    top = XtInitialize ( argv[0], "Fast", NULL, 0, &argc, argv );

/*
 *  Save a pointer to the X Windows display structure. Save the current screen.
 */

    display = XtDisplay ( top );
    screen  = DefaultScreen ( display );

/*
 *  Initialize font information.
 */

    if ( ( font_info = XLoadQueryFont ( display, argv[1] ) ) == 0 )
        exit ( 1 );

    font = font_info->fid;

/*
 *  Save the format to use for printing out of data.
 */

    strcpy ( format, argv[2] );

/*
 *  Initialize size of string to draw.
 */

    XTextExtents ( font_info, string, strlen ( string ), &ascent,
                   &ascent, &ascent, &overall );
    width  = overall.width;
    height = overall.ascent + overall.descent;
    ascent = overall.ascent;

/*
 *  Query the X server to find out if there is the right type of visual.
 */

    v.screen = screen;
    v.depth  = 8;
    v.class  = PseudoColor;

    visual_list = XGetVisualInfo ( display,
                      VisualScreenMask | VisualDepthMask | VisualClassMask,
```

```
                    &v, &visuals_matched );

    /*
     *  Copy the required visual and free up memory allocated for by the visual
     *  query function.
     */

        memcpy ( &visual, &visual_list[0].visual, sizeof ( visual ) );
        XFree ( visual_list );

    /*
     *  Create the main window and a menu bar.
     */

        XtManageChild ( m_main = XmCreateMainWindow ( top, "", NULL, 0 ) );
        XtManageChild ( mb_main = XmCreateMenuBar ( m_main, "", NULL, 0 ) );

    /*
     *  Create menu.
     */

        mp_file = XmCreatePulldownMenu ( mb_main, "", NULL, 0 );

        i = 0;
        XtSetArg ( args[i], XmNsubMenuId, mp_file ); i++;
        XtManageChild ( XmCreateCascadeButton ( mb_main, "File", args, i ) );
        XtManageChild ( XmCreatePushButton    ( mp_file, "Exit", NULL, 0 ) );

    /*
     *  Create a scrolled window widget.
     */

        i = 0;
        XtSetArg ( args[i], XmNwidth,  COLS*(width +2)); i++;
        XtSetArg ( args[i], XmNheight, ROWS*(height+2)); i++;
        XtManageChild ( scroll =
                XmCreateScrolledWindow ( m_main, "scroll", args, i ) );

    /*
     *  Create the drawing area widget.
     */

        cb[0].callback = (XtCallbackProc)cb_expose;
        cb[0].closure  = (caddr_t)0;

        i = 0;
        XtSetArg ( args[i], XmNexposeCallback, cb             ); i++;
        XtSetArg ( args[i], XmNwidth,          COLS*(width +2)); i++;
        XtSetArg ( args[i], XmNheight,         ROWS*(height+2)); i++;
        XtManageChild ( draw =
                XmCreateDrawingArea ( scroll, "draw", args, i ) );

    /*
     *  Realize the widgets.
     */

        XtRealizeWidget ( top );

    /*
     *  Set the attributes necessary to create the actual window.
     */

        attributes.save_under        = 0;
        attributes.backing_store     = NotUseful;
```

```
    attributes.border_pixel      = BlackPixel ( display, screen );
    attributes.background_pixel  = WhitePixel ( display, screen );
    attributes.bit_gravity       = NorthWestGravity;

    mask = CWBackingStore | CWSaveUnder | CWBackPixel |
           CWBorderPixel  | CWBitGravity;

/*
 *  Create the window for the drawing area widget.
 */

    XtCreateWindow ( draw, CopyFromParent, visual, mask, &attributes );
    window = XtWindow ( draw );

/*
 *  Create the graphics context.
 */

    gc = XCreateGC ( display, window, NULL, NULL );

    XSetFont ( display, gc, font );

/*
 *  Create the pixmap used to retain a copy of the image for refreshing
 *  the window.
 */

    pixmap = XCreatePixmap ( display, window, COLS*(width +2), ROWS*(height+2),
                             8 );

/*
 *  Add a time out.
 */

    XtAddTimeOut ( TIMER_VALUE, cb_expose, NULL );

/*
 *  Loop forever.
 */

    XtMainLoop ( );
}


XtCallbackProc cb_expose ( widget, closure, calldata )

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

    caddr_t closure;             /* Callback specific data.  This parameter
                                  * indicates the selected function.
                                  */
    XmDrawingAreaCallbackStruct *calldata;
                                 /* Specifies any callback-specific data the
                                  * widget needs to pass to the client.
                                  */
{
    register int        x, y;

    static long         value = 1000;

    static int          count=1;
```

```
/*
 *  Update the string.
 */

    value += 1;

/*
 *  Write out the strings.
 */

    for ( y = 0; y < ROWS; y++ )
        for ( x = 0; x < COLS; x++ ) {
            sprintf ( string, format, value );
            XDrawImageString ( display, window, gc, x*(width+2),
                               ascent + (y*(height+2)),
                               string, strlen ( string ) );

        }

/*
 *  Flush the buffer immediately.
 */

    XFlush ( display );

/*
 *  Test and increment counter
 */

    if (count > CYCLES)
        exit(1);
    count++;


/*
 *  Reset the timer.
 */

    XtAddTimeOut ( TIMER_VALUE, cb_expose, NULL );
}
```

```
/*
    Draw straight to screen using XDrawImageString
    1 gc switch/cycle
    no text formatting
 */

#include <stdio.h>
#include <string.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Cardinals.h>
#include <X11/Shell.h>
#include <X11/MwmUtil.h>
#include <Xm/Xm.h>
#include <Xm/DialogS.h>
#include <Xm/DrawingA.h>
#include <Xm/MainW.h>
#include <Xm/RowColumn.h>


#define CYCLES      100
#define TIMER_VALUE 1000
#define ROWS        50
#define COLS        10

Display             *display;

Visual              *visual;

Pixmap              pixmap;

GC                  gc[5];

Font                font;

Window              window;

XFontStruct         *font_info;

int                 screen;
                    ascent,
                    width,
                    height;

char                string[] = "1000.00001",
                    format[10];


int main ( argc, argv )

    int     argc;

    char    *argv[];
{
    register int            i;

    Widget                  top, m_main, mb_main, mp_file, scroll, draw;

    Arg                     args[10];

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

```
    XtCallbackProc          cb_expose();
    XColor                  color;
    Colormap                xcmap;
    XVisualInfo             v,
                            *visual_list;
    XSetWindowAttributes    attributes;
    XCharStruct             overall;
    int                     visuals_matched;
    unsigned long           mask;
    int                     z;

/*
 *  Initialize the Toolkit.
 */
    top = XtInitialize ( argv[0], "Fast", NULL, 0, &argc, argv );

/*
 *  Save a pointer to the X Windows display structure. Save the current screen.
 */
    display = XtDisplay ( top );
    screen  = DefaultScreen ( display );

/*
 *  Initialize font information.
 */
    if ( ( font_info = XLoadQueryFont ( display, argv[1] ) ) == 0 )
        exit ( 1 );
    font = font_info->fid;

/*
 *  Save the format to use for printing out of data.
 */
    strcpy ( format, argv[2] );

/*
 *  Initialize size of string to draw.
 */
    XTextExtents ( font_info, string, strlen ( string ), &ascent,
                   &ascent, &ascent, &overall );
    width  = overall.width;
    height = overall.ascent + overall.descent;
    ascent = overall.ascent;

/*
 *  Query the X server to find out if there is the right type of visual.
 */
```

```
        v.screen = screen;
        v.depth  = 8;
        v.class  = PseudoColor;

        visual_list = XGetVisualInfo ( display,
                          VisualScreenMask | VisualDepthMask | VisualClassMask,
                          &v, &visuals_matched );

    /*
     *  Copy the required visual and free up memory allocated for by the visual
     *  query function.
     */

        memcpy ( &visual, &visual_list[0].visual, sizeof ( visual ) );
        XFree ( visual_list );

    /*
     *  Create the main window and a menu bar.
     */

        XtManageChild ( m_main = XmCreateMainWindow ( top, "", NULL, 0 ) );
        XtManageChild ( mb_main = XmCreateMenuBar ( m_main, "", NULL, 0 ) );

    /*
     *  Create menu.
     */

        mp_file = XmCreatePulldownMenu ( mb_main, "", NULL, 0 );

        i = 0;
        XtSetArg ( args[i], XmNsubMenuId, mp_file ); i++;
        XtManageChild ( XmCreateCascadeButton ( mb_main, "File", args, i ) );
        XtManageChild ( XmCreatePushButton    ( mp_file, "Exit", NULL, 0 ) );

    /*
     *  Create a scrolled window widget.
     */

        i = 0;
        XtSetArg ( args[i], XmNwidth,  COLS*(width +2)); i++;
        XtSetArg ( args[i], XmNheight, ROWS*(height+2)); i++;
        XtManageChild ( scroll =
                XmCreateScrolledWindow ( m_main, "scroll", args, i ) );

    /*
     *  Create the drawing area widget.
     */

        cb[0].callback = (XtCallbackProc)cb_expose;
        cb[0].closure  = (caddr_t)0;

        i = 0;
        XtSetArg ( args[i], XmNexposeCallback, cb             ); i++;
        XtSetArg ( args[i], XmNwidth,          COLS*(width +2)); i++;
        XtSetArg ( args[i], XmNheight,         ROWS*(height+2)); i++;
        XtManageChild ( draw =
                XmCreateDrawingArea ( scroll, "draw", args, i ) );

    /*
     *  Realize the widgets.
     */

        XtRealizeWidget ( top );
```

```
/*
 *  Set the attributes necessary to create the actual window.
 */

    attributes.save_under         = 0;
    attributes.backing_store      = NotUseful;
    attributes.border_pixel       = BlackPixel ( display, screen );
    attributes.background_pixel   = WhitePixel ( display, screen );
    attributes.bit_gravity        = NorthWestGravity;

    mask = CWBackingStore | CWSaveUnder | CWBackPixel |
           CWBorderPixel  | CWBitGravity;

/*
 *  Create the window for the drawing area widget.
 */

    XtCreateWindow ( draw, CopyFromParent, visual, mask, &attributes );
    window = XtWindow ( draw );

    xcmap = DefaultColormap(display, screen);

/*
 *  Create the graphics context.
 */

    gc[1] = XCreateGC ( display, window, NULL, NULL );

    gc[2] = XCreateGC ( display, window, NULL, NULL );
    color.red = 65000;
    color.blue = 0;
    color.green = 0;
    if (XAllocColor(display, xcmap, &color))
        XSetBackground(display, gc[2], color.pixel);
    else
        fprintf(stderr, "couldn't allocate color");

    gc[3] = XCreateGC ( display, window, NULL, NULL );
    color.red = 0;
    color.blue = 65000;
    color.green = 0;
    if (XAllocColor(display, xcmap, &color))
        XSetBackground(display, gc[3], color.pixel);
    else
        fprintf(stderr, "couldn't allocate color");

    gc[4] = XCreateGC ( display, window, NULL, NULL );
    color.red = 0;
    color.blue = 0;
    color.green = 65000;
    if (XAllocColor(display, xcmap, &color))
        XSetBackground(display, gc[4], color.pixel);
    else
        fprintf(stderr, "couldn't allocate color");

    for (z=1; z<5; z++)
        XSetFont ( display, gc[z], font );

/*
 *  Create the pixmap used to retain a copy of the image for refreshing
 *  the window.
 */

    pixmap = XCreatePixmap ( display, window, COLS*(width +2), ROWS*(height+2),
```

```
                                8 );

    /*
     *  Add a time out.
     */

        XtAddTimeOut ( TIMER_VALUE, cb_expose, NULL );

    /*
     *  Loop forever.
     */

        XtMainLoop ( );
    }


    XtCallbackProc cb_expose ( widget, closure, calldata )

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

        caddr_t closure;            /* Callback specific data.  This parameter
                                     * indicates the selected function.
                                     */
        XmDrawingAreaCallbackStruct *calldata;
                                    /* Specifies any callback-specific data the
                                     * widget needs to pass to the client.
                                     */
    {
        register int        x, y;

        static double       value = 1000.00001;

        static int          count=1;

        int                 z;

    /*
     *  Update the string.
     */

        value += (double)0.00001;

    /*
     *  Determine which gc to use
     */

        z = (count % 4) + 1;

    /*
     *  Write out the strings.
     */

        for ( y = 0; y < ROWS; y++ )
            for ( x = 0; x < COLS; x++ ) {
                XDrawImageString ( display, window, gc[z], x*(width+2),
                                   ascent + (y*(height+2)),
                                   string, strlen ( string ) );
            }

    /*
     *  Flush the buffer immediately.
     */
```

```
    XFlush ( display );

/*
 *  Test and increment counter
 */

    if (count > CYCLES)
        exit(1);
    count++;


/*
 *  Reset the timer.
 */

    XtAddTimeOut ( TIMER_VALUE, cb_expose, NULL );
}
```

```
/*
    Draw straight to screen using XDrawImageString
    multiple gc switches/cycle
    no text formatting
 */

#include <stdio.h>
#include <string.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Cardinals.h>
#include <X11/Shell.h>
#include <X11/MwmUtil.h>
#include <Xm/Xm.h>
#include <Xm/DialogS.h>
#include <Xm/DrawingA.h>
#include <Xm/MainW.h>
#include <Xm/RowColumn.h>


#define CYCLES       100
#define TIMER_VALUE 1000
#define ROWS         50
#define COLS         10

Display              *display;

Visual               *visual;

Pixmap               pixmap;

GC                   gc[5];

Font                 font;

Window               window;

XFontStruct          *font_info;

int                  screen;
                     ascent,
                     width,
                     height;

char                 string[] = "1000.00001",
                     format[10];


int main ( argc, argv )

    int      argc;

    char     *argv[];
{
    register int             i;

    Widget                   top, m_main, mb_main, mp_file, scroll, draw;

    Arg                      args[10];

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

```
    XtCallbackProc          cb_expose();

    XColor                  color;

    Colormap                xcmap;

    XVisualInfo             v,
                            *visual_list;

    XSetWindowAttributes    attributes;

    XCharStruct             overall;

    int                     visuals_matched;

    unsigned long           mask;


    int                     z;


/*
 *  Initialize the Toolkit.
 */

    top = XtInitialize ( argv[0], "Fast", NULL, 0, &argc, argv );

/*
 *  Save a pointer to the X Windows display structure. Save the current screen.
 */

    display = XtDisplay ( top );
    screen  = DefaultScreen ( display );

/*
 *  Initialize font information.
 */

    if ( ( font_info = XLoadQueryFont ( display, argv[1] ) ) == 0 )
        exit ( 1 );

    font = font_info->fid;

/*
 *  Save the format to use for printing out of data.
 */

    strcpy ( format, argv[2] );

/*
 *  Initialize size of string to draw.
 */

    XTextExtents ( font_info, string, strlen ( string ), &ascent,
                   &ascent, &ascent, &overall );
    width  = overall.width;
    height = overall.ascent + overall.descent;
    ascent = overall.ascent;

/*
 *  Query the X server to find out if there is the right type of visual.
 */
```

```
    v.screen = screen;
    v.depth  = 8;
    v.class  = PseudoColor;

    visual_list = XGetVisualInfo ( display,
                      VisualScreenMask | VisualDepthMask | VisualClassMask,
                      &v, &visuals_matched );

/*
 *  Copy the required visual and free up memory allocated for by the visual
 *  query function.
 */

    memcpy ( &visual, &visual_list[0].visual, sizeof ( visual ) );
    XFree ( visual_list );

/*
 *  Create the main window and a menu bar.
 */

    XtManageChild ( m_main = XmCreateMainWindow ( top, "", NULL, 0 ) );
    XtManageChild ( mb_main = XmCreateMenuBar ( m_main, "", NULL, 0 ) );

/*
 *  Create menu.
 */

    mp_file = XmCreatePulldownMenu ( mb_main, "", NULL, 0 );

    i = 0;
    XtSetArg ( args[i], XmNsubMenuId, mp_file ); i++;
    XtManageChild ( XmCreateCascadeButton ( mb_main, "File", args, i ) );
    XtManageChild ( XmCreatePushButton    ( mp_file, "Exit", NULL, 0 ) );

/*
 *  Create a scrolled window widget.
 */

    i = 0;
    XtSetArg ( args[i], XmNwidth,  COLS*(width +2)); i++;
    XtSetArg ( args[i], XmNheight, ROWS*(height+2)); i++;
    XtManageChild ( scroll =
            XmCreateScrolledWindow ( m_main, "scroll", args, i ) );

/*
 *  Create the drawing area widget.
 */

    cb[0].callback = (XtCallbackProc)cb_expose;
    cb[0].closure  = (caddr_t)0;

    i = 0;
    XtSetArg ( args[i], XmNexposeCallback, cb             ); i++;
    XtSetArg ( args[i], XmNwidth,          COLS*(width +2)); i++;
    XtSetArg ( args[i], XmNheight,         ROWS*(height+2)); i++;
    XtManageChild ( draw =
            XmCreateDrawingArea ( scroll, "draw", args, i ) );

/*
 *  Realize the widgets.
 */

    XtRealizeWidget ( top );
```

```
/*
 *  Set the attributes necessary to create the actual window.
 */

    attributes.save_under         = 0;
    attributes.backing_store      = NotUseful;
    attributes.border_pixel       = BlackPixel ( display, screen );
    attributes.background_pixel   = WhitePixel ( display, screen );
    attributes.bit_gravity        = NorthWestGravity;

    mask = CWBackingStore | CWSaveUnder | CWBackPixel |
           CWBorderPixel  | CWBitGravity;

/*
 *  Create the window for the drawing area widget.
 */

    XtCreateWindow ( draw, CopyFromParent, visual, mask, &attributes );
    window = XtWindow ( draw );

/*
 *  Create and install a new color map. The color map is associated with
 *  the shell widget and is "installed" by the window manager.
 */

    /*xcmap = XCreateColormap ( display, window, visual, AllocAll );
    XSetWindowColormap ( display, XtWindow ( top ), xcmap );*/

    xcmap = DefaultColormap(display, screen);

/*
 *  Create the graphics context.
 */

    gc[1] = XCreateGC ( display, window, NULL, NULL );

    gc[2] = XCreateGC ( display, window, NULL, NULL );
    color.red = 65000;
    color.blue = 0;
    color.green = 0;
    if (XAllocColor(display, xcmap, &color))
        XSetBackground(display, gc[2], color.pixel);
    else
        fprintf(stderr, "couldn't allocate color");

    gc[3] = XCreateGC ( display, window, NULL, NULL );
    color.red = 0;
    color.blue = 65000;
    color.green = 0;
    if (XAllocColor(display, xcmap, &color))
        XSetBackground(display, gc[3], color.pixel);
    else
        fprintf(stderr, "couldn't allocate color");

    gc[4] = XCreateGC ( display, window, NULL, NULL );
    color.red = 0;
    color.blue = 0;
    color.green = 65000;
    if (XAllocColor(display, xcmap, &color))
        XSetBackground(display, gc[4], color.pixel);
    else
        fprintf(stderr, "couldn't allocate color");

    for (z=1; z<5; z++)
```

```
        XSetFont ( display, gc[z], font );

    /*
     *  Create the pixmap used to retain a copy of the image for refreshing
     *  the window.
     */

        pixmap = XCreatePixmap ( display, window, COLS*(width +2), ROWS*(height+2),
                                 8 );

    /*
     *  Add a time out.
     */

        XtAddTimeOut ( TIMER_VALUE, cb_expose, NULL );

    /*
     *  Loop forever.
     */

        XtMainLoop ( );
    }


    XtCallbackProc cb_expose ( widget, closure, calldata )

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

        caddr_t closure;            /* Callback specific data.  This parameter
                                     * indicates the selected function.
                                     */
        XmDrawingAreaCallbackStruct *calldata;
                                    /* Specifies any callback-specific data the
                                     * widget needs to pass to the client.
                                     */
    {
        register int        x, y;

        static double       value = 1000.00001;

        static int          count=1;

        int                 z;

    /*
     *  Update the string.
     */

        value += (double)0.00001;

    /*
     *  Write out the strings.
     */

        for ( y = 0; y < ROWS; y++ )
            for ( x = 0; x < COLS; x++ ) {
                z = ((x + y) % 4) + 1;
                XDrawImageString ( display, window, gc[z], x*(width+2),
                        ascent + (y*(height+2)), string, strlen ( string ) );
            }

    /*
```

```
 *  Flush the buffer immediately.
 */

    XFlush ( display );

/*
 *  Test and increment counter
 */

    if (count > CYCLES)
        exit(1);
    count++;


/*
 *  Reset the timer.
 */

    XtAddTimeOut ( TIMER_VALUE, cb_expose, NULL );
}
```

```
/*
    Draw straight to screen using XDrawImageString
    no gc switching
    No text formatting
 */

#include <stdio.h>
#include <string.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Cardinals.h>
#include <X11/Shell.h>
#include <X11/MwmUtil.h>
#include <Xm/Xm.h>
#include <Xm/DialogS.h>
#include <Xm/DrawingA.h>
#include <Xm/MainW.h>
#include <Xm/RowColumn.h>


#define CYCLES       100
#define TIMER_VALUE 1000
#define ROWS         50
#define COLS         10

Display              *display;

Visual               *visual;

Pixmap               pixmap;

GC                   gc;

Font                 font;

Window               window;

XFontStruct          *font_info;

int                  screen;
                     ascent,
                     width,
                     height;

char                 string[] = "1000.00001",
                     format[10];


int main ( argc, argv )

    int      argc;

    char     *argv[];
{
    register int            i;

    Widget                  top, m_main, mb_main, mp_file, scroll, draw;

    Arg                     args[10];

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

```
    XtCallbackProc          cb_expose();

    XColor                  color;

    XVisualInfo             v,
                            *visual_list;

    XSetWindowAttributes    attributes;

    XCharStruct             overall;

    int                     visuals_matched;

    unsigned long           mask;

/*
 *  Initialize the Toolkit.
 */

    top = XtInitialize ( argv[0], "Fast", NULL, 0, &argc, argv );

/*
 *  Save a pointer to the X Windows display structure. Save the current screen.
 */

    display = XtDisplay ( top );
    screen  = DefaultScreen ( display );

/*
 *  Initialize font information.
 */

    if ( ( font_info = XLoadQueryFont ( display, argv[1] ) ) == 0 )
        exit ( 1 );

    font = font_info->fid;

/*
 *  Save the format to use for printing out of data.
 */

    strcpy ( format, argv[2] );

/*
 *  Initialize size of string to draw.
 */

    XTextExtents ( font_info, string, strlen ( string ), &ascent,
                   &ascent, &ascent, &overall );
    width  = overall.width;
    height = overall.ascent + overall.descent;
    ascent = overall.ascent;

/*
 *  Query the X server to find out if there is the right type of visual.
 */

    v.screen = screen;
    v.depth  = 8;
    v.class  = PseudoColor;

    visual_list = XGetVisualInfo ( display,
                      VisualScreenMask | VisualDepthMask | VisualClassMask,
```

```
                        &v, &visuals_matched );

    /*
     *  Copy the required visual and free up memory allocated for by the visual
     *  query function.
     */

        memcpy ( &visual, &visual_list[0].visual, sizeof ( visual ) );
        XFree ( visual_list );

    /*
     *  Create the main window and a menu bar.
     */

        XtManageChild ( m_main = XmCreateMainWindow ( top, "", NULL, 0 ) );
        XtManageChild ( mb_main = XmCreateMenuBar ( m_main, "", NULL, 0 ) );

    /*
     *  Create menu.
     */

        mp_file = XmCreatePulldownMenu ( mb_main, "", NULL, 0 );

        i = 0;
        XtSetArg ( args[i], XmNsubMenuId, mp_file ); i++;
        XtManageChild ( XmCreateCascadeButton ( mb_main, "File", args, i ) );
        XtManageChild ( XmCreatePushButton    ( mp_file, "Exit", NULL, 0 ) );

    /*
     *  Create a scrolled window widget.
     */

        i = 0;
        XtSetArg ( args[i], XmNwidth,  COLS*(width +2)); i++;
        XtSetArg ( args[i], XmNheight, ROWS*(height+2)); i++;
        XtManageChild ( scroll =
                XmCreateScrolledWindow ( m_main, "scroll", args, i ) );

    /*
     *  Create the drawing area widget.
     */

        cb[0].callback = (XtCallbackProc)cb_expose;
        cb[0].closure  = (caddr_t)0;

        i = 0;
        XtSetArg ( args[i], XmNexposeCallback, cb             ); i++;
        XtSetArg ( args[i], XmNwidth,          COLS*(width +2)); i++;
        XtSetArg ( args[i], XmNheight,         ROWS*(height+2)); i++;
        XtManageChild ( draw =
                XmCreateDrawingArea ( scroll, "draw", args, i ) );

    /*
     *  Realize the widgets.
     */

        XtRealizeWidget ( top );

    /*
     *  Set the attributes necessary to create the actual window.
     */

        attributes.save_under       = 0;
        attributes.backing_store    = NotUseful;
```

```
    attributes.border_pixel      = BlackPixel ( display, screen );
    attributes.background_pixel  = WhitePixel ( display, screen );
    attributes.bit_gravity       = NorthWestGravity;

    mask = CWBackingStore | CWSaveUnder | CWBackPixel |
           CWBorderPixel  | CWBitGravity;

/*
 *  Create the window for the drawing area widget.
 */

    XtCreateWindow ( draw, CopyFromParent, visual, mask, &attributes );
    window = XtWindow ( draw );

/*
 *  Create and install a new color map. The color map is associated with
 *  the shell widget and is "installed" by the window manager.
 */

    /*xcmap = XCreateColormap ( display, window, visual, AllocAll );
    XSetWindowColormap ( display, XtWindow ( top ), xcmap );*/

/*
 *  Create the graphics context.
 */

    gc = XCreateGC ( display, window, NULL, NULL );

    XSetFont ( display, gc, font );

/*
 *  Create the pixmap used to retain a copy of the image for refreshing
 *  the window.
 */

    pixmap = XCreatePixmap ( display, window, COLS*(width +2), ROWS*(height+2),
                             8 );

/*
 *  Add a time out.
 */

    XtAddTimeOut ( TIMER_VALUE, cb_expose, NULL );

/*
 *  Loop forever.
 */

    XtMainLoop ( );
}


XtCallbackProc cb_expose ( widget, closure, calldata )

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

    caddr_t closure;             /* Callback specific data.  This parameter
                                  * indicates the selected function.
                                  */
    XmDrawingAreaCallbackStruct *calldata;
                                 /* Specifies any callback-specific data the
                                  * widget needs to pass to the client.
```

```
                                        */
{
    register int        x, y;

    static double       value = 1000.00001;

    static int          count=1;

/*
 *  Update the string.
 */

    value += (double)0.00001;

/*
 *  Write out the strings.
 */

    for ( y = 0; y < ROWS; y++ )
        for ( x = 0; x < COLS; x++ ) {
            XDrawImageString ( display, window, gc, x*(width+2),
                               ascent + (y*(height+2)),
                               string, strlen ( string ) );
        }

/*
 *  Flush the buffer immediately.
 */

    XFlush ( display );

/*
 *  Test and increment counter
 */

    if (count > CYCLES)
        exit(1);
    count++;


/*
 *  Reset the timer.
 */

    XtAddTimeOut ( TIMER_VALUE, cb_expose, NULL );
}
```

ATTACHMENT 3 - Analysis Results

```
*********************************************
***** Process Resource Monitor Phase 2 *****
*****        Analysis Report            *****
*********************************************
```

LogTime from: Fri Aug 17 11:09:47 1990
to: Fri Aug 17 11:12:16 1990
Host Id: aries
CPU type: 68030
number: 2

COMMAND	PROCESS ID.	80% CPU UASAGE		50% CPU UASAGE		MEMORY COMSUMPTION		
		Milli Sec.	% of Total	Milli Sec.	% of Total	Phys	Virt	Shr
SWAPPER	0	0	0.00	0	0.00	0	0	0.0
init	1	0	0.00	0	0.00	15	20	0.0
PAGEDAEMON	2	0	0.00	0	0.00	8	2056	0.0
lwp1	3	0	0.00	0	0.00	8	8	0.0
lwp2	4	0	0.00	0	0.00	8	8	0.0
getty	72	0	0.00	0	0.00	12	18	0.0
update	34	0	0.00	0	0.00	3	8	0.0
csh	71	0	0.00	0	0.00	41	50	0.0
getty	73	0	0.00	0	0.00	12	18	0.0
errdemon	50	0	0.00	0	0.00	10	16	0.0
cron	54	0	0.00	0	0.00	31	36	0.0
Xgcm	69	0	0.00	0	0.00	171	198	0.0
Xgcm	70	217	10.85	184	9.20	1956	3286	0.0
portmap	74	0	0.00	0	0.00	22	29	0.0
netd	75	0	0.00	0	0.00	24	31	0.0
rwhod	76	0	0.00	0	0.00	24	29	0.0
talkd	77	0	0.00	0	0.00	25	32	0.0
rwalld	78	0	0.00	0	0.00	16	22	0.0
rusersd	79	0	0.00	0	0.00	22	29	0.0
timed	80	0	0.00	0	0.00	31	39	0.0
xdm	91	0	0.00	0	0.00	104	115	2.0
ypbind	82	0	0.00	0	0.00	18	24	0.0
biod	83	0	0.00	0	0.00	11	11	0.0
mountd	84	0	0.00	0	0.00	25	31	0.0
csh	372	0	0.00	0	0.00	43	50	0.0
xdm	86	0	0.00	0	0.00	81	92	0.0
nfsd	87	0	0.00	0	0.00	13	19	0.0
xdm	88	0	0.00	0	0.00	82	97	2.0
xdm	89	0	0.00	0	0.00	82	97	2.0
xcterm	371	0	0.00	0	0.00	193	227	2.0
csh	361	0	0.00	0	0.00	43	50	0.0
xdm	254	0	0.00	0	0.00	105	116	0.0
xcterm	355	0	0.00	0	0.00	190	224	2.0
prm	551	100	5.00	84	4.20	54	76	0.0
mwm	357	0	0.00	0	0.00	209	231	2.0
csh	405	0	0.00	0	0.00	43	50	0.0
xcterm	404	0	0.00	0	0.00	194	228	2.0
csh	484	0	0.00	0	0.00	43	50	0.0
drflmgc	552	234	11.70	217	10.85	142	188	2.0
xcterm	483	0	0.00	0	0.00	193	227	2.0

PROCESS ID.	CONTEXT SWITCHING VOL. Total	VOL. Avg.	INVOL. Total	INVOL. Avg.	PROCESS PRIORITY High	Low	Avg.	TOTAL I/O ACT.	SIGNALS RECEIVED
0	0	0.0	0	0.0	35	35	35	0	0
1	0	0.0	0	0.0	45	45	45	331	295
2	0	0.0	0	0.0	-25	-25	-25	0	0
3	73	0.5	0	0.0	35	35	35	0	0
4	0	0.0	0	0.0	44	44	44	0	0
72	0	0.0	0	0.0	45	45	45	0	1
34	15	0.1	0	0.0	46	45	45	7861	2289
71	0	0.0	0	0.0	45	45	45	19	17
73	0	0.0	0	0.0	45	45	45	1	1
50	0	0.0	0	0.0	46	46	46	4	0
54	0	0.0	0	0.0	45	45	45	79	39
69	0	0.0	0	0.0	40	40	40	51	38
70	610	4.1	16	0.1	47	41	43	104	30
74	0	0.0	0	0.0	45	45	45	6	0
75	9	0.1	0	0.0	45	45	45	16	1
76	0	0.0	0	0.0	45	45	45	2023	382
77	0	0.0	0	0.0	46	46	46	4	0
78	0	0.0	0	0.0	45	45	45	3	0
79	0	0.0	0	0.0	46	46	46	5	0
80	0	0.0	0	0.0	45	45	45	21	0
91	0	0.0	0	0.0	49	49	49	4	0
82	0	0.0	0	0.0	45	45	45	7	3
83	442	3.0	0	0.0	46	45	45	5	0
84	0	0.0	0	0.0	45	45	45	54	0
372	0	0.0	0	0.0	45	45	45	0	6
86	0	0.0	0	0.0	47	47	47	11	0
87	582	3.9	0	0.0	48	45	45	423	0
88	0	0.0	0	0.0	46	46	46	1	0
89	0	0.0	0	0.0	46	46	46	0	0
371	0	0.0	0	0.0	44	44	44	0	0
361	0	0.0	0	0.0	49	49	49	3	7
254	0	0.0	0	0.0	48	48	48	10	0
355	0	0.0	0	0.0	44	44	44	6	0
551	314	2.1	223	1.5	47	45	45	244	1
357	3	0.0	2	0.0	46	44	45	10	0
405	0	0.0	0	0.0	45	45	45	1	62
404	0	0.0	0	0.0	45	45	45	3	0
484	3	0.0	0	0.0	46	45	45	0	25
552	322	2.9	296	2.7	52	44	48	29	0
483	2	0.0	0	0.0	45	45	45	0	0

```
*********************************************
***** Process Resource Monitor Phase 2 *****
*****        Analysis Report            *****
*********************************************
```

LogTime from: Thu Aug 16 17:05:38 1990
to: Thu Aug 16 17:08:07 1990
Host Id: aries
CPU type: 68030
number: 2

COMMAND	PROCESS ID.	80% CPU UASAGE		50% CPU UASAGE		MEMORY COMSUMPTION		
		Milli Sec.	% of Total	Milli Sec.	% of Total	Phys	Virt	Shr
SWAPPER	0	0	0.00	0	0.00	0	0	0.0
init	1	0	0.00	0	0.00	15	20	0.0
PAGEDAEMON	2	0	0.00	0	0.00	8	2056	0.0
lwp1	3	0	0.00	0	0.00	8	8	0.0
lwp2	4	0	0.00	0	0.00	8	8	0.0
getty	72	0	0.00	0	0.00	12	18	0.0
update	34	0	0.00	0	0.00	3	8	0.0
getty	71	0	0.00	0	0.00	12	18	0.0
getty	73	0	0.00	0	0.00	12	18	0.0
errdemon	50	0	0.00	0	0.00	10	16	0.0
cron	54	0	0.00	0	0.00	31	36	0.0
Xgcm	69	0	0.00	0	0.00	170	196	0.0
Xgcm	70	183	9.15	166	8.30	1839	3118	0.0
portmap	74	0	0.00	0	0.00	17	23	0.0
netd	75	0	0.00	0	0.00	24	31	0.0
rwhod	76	0	0.00	0	0.00	24	29	0.0
talkd	77	0	0.00	0	0.00	25	32	0.0
rwalld	78	0	0.00	0	0.00	16	22	0.0
rusersd	79	0	0.00	0	0.00	22	29	0.0
timed	80	0	0.00	0	0.00	31	39	0.0
xdm	91	0	0.00	0	0.00	104	115	2.0
ypbind	82	0	0.00	0	0.00	18	24	0.0
biod	83	0	0.00	0	0.00	11	11	0.0
mountd	84	0	0.00	0	0.00	24	31	0.0
csh	122	0	0.00	0	0.00	43	50	0.0
xdm	86	0	0.00	0	0.00	81	92	0.0
nfsd	87	0	0.00	0	0.00	13	19	0.0
xdm	88	0	0.00	0	0.00	82	97	2.0
xdm	89	0	0.00	0	0.00	82	97	2.0
xcterm	121	0	0.00	0	0.00	215	228	2.0
xdm	94	0	0.00	0	0.00	105	116	0.0
mwm	107	0	0.00	0	0.00	215	226	2.0
csh	111	0	0.00	0	0.00	43	50	0.0
xcterm	105	0	0.00	0	0.00	212	225	2.0
go	207	0	0.00	0	0.00	34	43	0.0
drnofmt	228	50	2.50	33	1.65	143	188	2.0
prm	227	99	4.95	83	4.15	54	76	0.0
sleep	229	0	0.00	0	0.00	5	13	2.0

PROCESS ID.	CONTEXT SWITCHING VOL. Total	VOL. Avg.	INVOL. Total	INVOL. Avg.	PROCESS PRIORITY High	Low	Avg.	TOTAL I/O ACT.	SIGNALS RECEIVED
0	0	0.0	0	0.0	35	35	35	0	0
1	0	0.0	0	0.0	45	45	45	177	28
2	0	0.0	0	0.0	-25	-25	-25	0	0
3	74	0.5	0	0.0	35	35	35	0	0
4	0	0.0	0	0.0	44	44	44	0	0
72	0	0.0	0	0.0	45	45	45	0	1
34	24	0.2	2	0.0	45	45	45	967	126
71	0	0.0	0	0.0	46	46	46	3	0
73	0	0.0	0	0.0	45	45	45	1	1
50	0	0.0	0	0.0	46	46	46	4	0
54	0	0.0	0	0.0	45	45	45	16	2
69	0	0.0	0	0.0	40	40	40	51	2
70	686	4.6	209	1.4	49	41	44	48	0
74	0	0.0	0	0.0	45	45	45	6	0
75	0	0.0	0	0.0	45	45	45	15	1
76	9	0.1	0	0.0	45	45	45	183	21
77	0	0.0	0	0.0	46	46	46	4	0
78	0	0.0	0	0.0	45	45	45	3	0
79	0	0.0	0	0.0	46	46	46	5	0
80	0	0.0	0	0.0	45	45	45	6	0
91	0	0.0	0	0.0	49	49	49	4	0
82	0	0.0	0	0.0	45	45	45	7	1
83	444	3.0	0	0.0	46	45	45	5	0
84	0	0.0	0	0.0	46	46	46	6	0
122	0	0.0	0	0.0	45	45	45	0	25
86	0	0.0	0	0.0	47	47	47	11	0
87	0	0.0	0	0.0	45	45	45	17	0
88	0	0.0	0	0.0	46	46	46	0	0
89	0	0.0	0	0.0	46	46	46	0	0
121	2	0.0	4	0.0	45	44	44	2	0
94	0	0.0	0	0.0	52	52	52	3	0
107	5	0.0	102	0.7	47	45	45	17	0
111	0	0.0	0	0.0	56	56	56	4	7
105	0	0.0	0	0.0	50	50	50	17	0
207	0	0.0	0	0.0	45	45	45	1	16
228	357	3.2	106	1.0	49	45	45	30	0
227	294	2.0	321	2.2	47	45	46	226	1
229	17	0.4	4	0.1	45	45	45	0	0

```
*********************************************
***** Process Resource Monitor Phase 2 *****
*****        Analysis Report            *****
*********************************************
```

LogTime from: Fri Aug 17 08:49:39 1990
to: Fri Aug 17 08:52:08 1990
Host Id: aries
CPU type: 68030
number: 2

COMMAND	PROCESS ID.	80% CPU UASAGE		50% CPU UASAGE		MEMORY COMSUMPTION		
		Milli Sec.	% of Total	Milli Sec.	% of Total	Phys	Virt	Shr
SWAPPER	0	0	0.00	0	0.00	0	0	0.0
init	1	0	0.00	0	0.00	15	20	0.0
PAGEDAEMON	2	0	0.00	0	0.00	8	2056	0.0
lwp1	3	0	0.00	0	0.00	8	8	0.0
lwp2	4	0	0.00	0	0.00	8	8	0.0
getty	72	0	0.00	0	0.00	12	18	0.0
update	34	0	0.00	0	0.00	3	8	0.0
getty	71	0	0.00	0	0.00	12	18	0.0
getty	73	0	0.00	0	0.00	12	18	0.0
errdemon	50	0	0.00	0	0.00	10	16	0.0
cron	54	0	0.00	0	0.00	31	36	0.0
Xgcm	69	0	0.00	0	0.00	171	198	0.0
Xgcm	70	133	6.65	100	5.00	1878	3158	0.0
portmap	74	0	0.00	0	0.00	22	29	0.0
netd	75	0	0.00	0	0.00	24	31	0.0
rwhod	76	0	0.00	0	0.00	24	29	0.0
talkd	77	0	0.00	0	0.00	25	32	0.0
rwalld	78	0	0.00	0	0.00	16	22	0.0
rusersd	79	0	0.00	0	0.00	22	29	0.0
timed	80	0	0.00	0	0.00	31	39	0.0
xdm	91	0	0.00	0	0.00	104	115	2.0
ypbind	82	0	0.00	0	0.00	18	24	0.0
biod	83	0	0.00	0	0.00	11	11	0.0
mountd	84	0	0.00	0	0.00	25	31	0.0
csh	372	0	0.00	0	0.00	43	50	0.0
xdm	86	0	0.00	0	0.00	81	92	0.0
nfsd	87	0	0.00	0	0.00	13	19	0.0
xdm	88	0	0.00	0	0.00	82	97	2.0
xdm	89	0	0.00	0	0.00	82	97	2.0
xcterm	371	0	0.00	0	0.00	193	227	2.0
csh	361	0	0.00	0	0.00	43	50	0.0
xdm	254	0	0.00	0	0.00	105	116	0.0
xcterm	355	0	0.00	0	0.00	190	224	2.0
csh	439	0	0.00	0	0.00	43	50	0.0
xcterm	438	0	0.00	0	0.00	193	227	2.0
mwm	357	0	0.00	0	0.00	206	228	2.0
csh	405	0	0.00	0	0.00	43	50	0.0
xcterm	404	0	0.00	0	0.00	193	227	2.0
stflgc	449	184	9.20	167	8.35	142	188	2.0
prm	448	100	5.00	84	4.20	54	76	2.0
cron	450	0	0.00	0	0.00	31	36	0.0

PROCESS ID.	CONTEXT SWITCHING VOL. Total	VOL. Avg.	INVOL. Total	INVOL. Avg.	PROCESS PRIORITY High	Low	Avg.	TOTAL I/O ACT.	SIGNALS RECEIVED
0	0	0.0	0	0.0	35	35	35	0	0
1	5	0.0	2	0.0	45	45	45	203	257
2	0	0.0	0	0.0	-25	-25	-25	0	0
3	73	0.5	0	0.0	35	35	35	0	0
4	0	0.0	0	0.0	44	44	44	0	0
72	0	0.0	0	0.0	45	45	45	0	1
34	9	0.1	0	0.0	45	44	44	5691	2010
71	0	0.0	0	0.0	46	46	46	3	0
73	0	0.0	0	0.0	45	45	45	1	1
50	0	0.0	0	0.0	46	46	46	4	0
54	3	0.0	2	0.0	46	45	45	62	34
69	0	0.0	0	0.0	45	45	45	51	33
70	676	4.5	6	0.0	47	40	42	93	30
74	0	0.0	0	0.0	45	45	45	6	0
75	9	0.1	0	0.0	45	45	45	15	1
76	7	0.0	0	0.0	45	45	45	1715	335
77	0	0.0	0	0.0	46	46	46	4	0
78	0	0.0	0	0.0	45	45	45	3	0
79	0	0.0	0	0.0	46	46	46	5	0
80	13	0.1	0	0.0	45	44	44	12	0
91	0	0.0	0	0.0	49	49	49	4	0
82	0	0.0	0	0.0	45	45	45	7	1
83	444	3.0	0	0.0	46	45	45	5	0
84	0	0.0	0	0.0	45	45	45	24	0
372	0	0.0	0	0.0	45	45	45	0	6
86	0	0.0	0	0.0	47	47	47	11	0
87	5	0.0	0	0.0	45	45	45	75	0
88	0	0.0	0	0.0	46	46	46	1	0
89	0	0.0	0	0.0	46	46	46	0	0
371	0	0.0	0	0.0	44	44	44	0	0
361	0	0.0	0	0.0	49	49	49	3	7
254	0	0.0	0	0.0	48	48	48	10	0
355	0	0.0	0	0.0	44	44	44	6	0
439	0	0.0	0	0.0	45	45	45	4	5
438	0	0.0	0	0.0	45	45	45	14	0
357	4	0.0	17	0.1	47	45	46	10	0
405	0	0.0	0	0.0	46	45	45	1	33
404	3	0.0	6	0.0	45	44	44	3	0
449	350	2.9	262	2.2	52	45	47	31	0
448	298	2.0	302	2.0	48	45	45	244	1
450	0	0.0	0	0.0	45	45	45	2	0

```
*********************************************
***** Process Resource Monitor Phase 2 *****
*****       Analysis Report             *****
*********************************************
```

LogTime from: Thu Aug 16 16:59:34 1990
to: Thu Aug 16 17:02:04 1990
Host Id: aries
CPU type: 68030
number: 2

COMMAND	PROCESS ID.	80% CPU UASAGE		50% CPU UASAGE		MEMORY COMSUMPTION		
		Milli Sec.	% of Total	Milli Sec.	% of Total	Phys	Virt	Shr
SWAPPER	0	0	0.00	0	0.00	0	0	0.0
init	1	0	0.00	0	0.00	15	20	0.0
PAGEDAEMON	2	0	0.00	0	0.00	8	2056	0.0
lwp1	3	0	0.00	0	0.00	8	8	0.0
lwp2	4	0	0.00	0	0.00	8	8	0.0
getty	72	0	0.00	0	0.00	12	18	0.0
update	34	0	0.00	0	0.00	3	8	0.0
getty	71	0	0.00	0	0.00	12	18	0.0
getty	73	0	0.00	0	0.00	12	18	0.0
errdemon	50	0	0.00	0	0.00	10	16	0.0
cron	54	0	0.00	0	0.00	31	36	0.0
Xgcm	69	0	0.00	0	0.00	170	196	0.0
Xgcm	70	150	7.50	133	6.65	1839	3118	0.0
portmap	74	0	0.00	0	0.00	17	23	0.0
netd	75	0	0.00	0	0.00	24	31	0.0
rwhod	76	0	0.00	0	0.00	24	29	0.0
talkd	77	0	0.00	0	0.00	25	32	0.0
rwalld	78	0	0.00	0	0.00	16	22	0.0
rusersd	79	0	0.00	0	0.00	22	29	0.0
timed	80	0	0.00	0	0.00	31	39	0.0
xdm	91	0	0.00	0	0.00	104	115	2.0
ypbind	82	0	0.00	0	0.00	18	24	0.0
biod	83	0	0.00	0	0.00	11	11	0.0
mountd	84	0	0.00	0	0.00	24	31	0.0
csh	122	0	0.00	0	0.00	43	50	0.0
xdm	86	0	0.00	0	0.00	81	92	0.0
nfsd	87	0	0.00	0	0.00	13	19	0.0
xdm	88	0	0.00	0	0.00	82	97	2.0
xdm	89	0	0.00	0	0.00	82	97	2.0
xcterm	121	0	0.00	0	0.00	215	228	2.0
xdm	94	0	0.00	0	0.00	105	116	0.0
mwm	107	0	0.00	0	0.00	215	226	2.0
csh	111	0	0.00	0	0.00	43	50	0.0
xcterm	105	0	0.00	0	0.00	212	225	2.0
go	207	0	0.00	0	0.00	34	43	0.0
stflmgc	221	184	9.20	167	8.35	142	188	2.0
prm	220	100	5.00	83	4.15	54	76	0.0
sleep	222	0	0.00	0	0.00	34	43	2.0

PROCESS ID.	CONTEXT SWITCHING VOL. Total	VOL. Avg.	INVOL. Total	INVOL. Avg.	PROCESS PRIORITY High	Low	Avg.	TOTAL I/O ACT.	SIGNALS RECEIVED
0	0	0.0	0	0.0	35	35	35	0	0
1	4	0.0	0	0.0	45	45	45	174	27
2	0	0.0	0	0.0	-25	-25	-25	0	0
3	74	0.5	0	0.0	35	35	35	0	0
4	0	0.0	0	0.0	44	44	44	0	0
72	0	0.0	0	0.0	45	45	45	0	1
34	16	0.1	6	0.0	45	45	45	887	113
71	0	0.0	0	0.0	46	46	46	3	0
73	0	0.0	0	0.0	45	45	45	1	1
50	0	0.0	0	0.0	46	46	46	4	0
54	0	0.0	0	0.0	45	45	45	16	2
69	0	0.0	0	0.0	40	40	40	51	1
70	658	4.4	8	0.1	44	41	42	40	0
74	0	0.0	0	0.0	45	45	45	6	0
75	0	0.0	0	0.0	44	44	44	15	1
76	3	0.0	0	0.0	45	45	45	161	18
77	0	0.0	0	0.0	46	46	46	4	0
78	0	0.0	0	0.0	45	45	45	3	0
79	0	0.0	0	0.0	46	46	46	5	0
80	0	0.0	0	0.0	45	45	45	6	0
91	0	0.0	0	0.0	49	49	49	4	0
82	0	0.0	0	0.0	45	45	45	7	1
83	444	3.0	0	0.0	46	45	45	5	0
84	0	0.0	0	0.0	46	46	46	6	0
122	0	0.0	0	0.0	45	45	45	0	25
86	0	0.0	0	0.0	47	47	47	11	0
87	0	0.0	0	0.0	45	45	45	14	0
88	0	0.0	0	0.0	46	46	46	0	0
89	0	0.0	0	0.0	46	46	46	0	0
121	0	0.0	0	0.0	45	45	45	2	0
94	0	0.0	0	0.0	52	52	52	3	0
107	2	0.0	4	0.0	46	45	45	17	0
111	0	0.0	0	0.0	56	56	56	4	7
105	0	0.0	0	0.0	50	50	50	17	0
207	3	0.0	2	0.0	45	45	45	0	10
221	454	3.8	412	3.4	49	47	47	32	0
220	297	2.0	325	2.2	47	45	45	250	1
222	0	0.0	0	0.0	45	45	45	0	0

```
*********************************************
***** Process Resource Monitor Phase 2 *****
*****        Analysis Report            *****
*********************************************
```

LogTime from: Thu Aug 16 16:18:30 1990
to: Thu Aug 16 16:20:59 1990
Host Id: aries
CPU type: 68030
number: 2

COMMAND	PROCESS ID.	80% CPU UASAGE		50% CPU UASAGE		MEMORY COMSUMPTION		
		Milli Sec.	% of Total	Milli Sec.	% of Total	Phys	Virt	Shr
SWAPPER	0	0	0.00	0	0.00	0	0	0.0
init	1	0	0.00	0	0.00	15	20	0.0
PAGEDAEMON	2	0	0.00	0	0.00	8	2056	0.0
lwp1	3	0	0.00	0	0.00	8	8	0.0
lwp2	4	0	0.00	0	0.00	8	8	0.0
getty	72	0	0.00	0	0.00	12	18	0.0
update	34	0	0.00	0	0.00	3	8	0.0
getty	71	0	0.00	0	0.00	12	18	0.0
getty	73	0	0.00	0	0.00	12	18	0.0
errdemon	50	0	0.00	0	0.00	10	16	0.0
cron	54	0	0.00	0	0.00	30	36	0.0
Xgcm	69	0	0.00	0	0.00	170	196	0.0
Xgcm	70	200	10.00	183	9.15	1834	3115	0.0
portmap	74	0	0.00	0	0.00	17	23	0.0
netd	75	0	0.00	0	0.00	24	31	0.0
rwhod	76	0	0.00	0	0.00	24	29	0.0
talkd	77	0	0.00	0	0.00	25	32	0.0
rwalld	78	0	0.00	0	0.00	16	22	0.0
rusersd	79	0	0.00	0	0.00	22	29	0.0
timed	80	0	0.00	0	0.00	31	39	0.0
xdm	91	0	0.00	0	0.00	104	115	2.0
ypbind	82	0	0.00	0	0.00	18	24	0.0
biod	83	16	0.80	0	0.00	11	11	0.0
mountd	84	0	0.00	0	0.00	24	31	0.0
csh	122	0	0.00	0	0.00	43	50	0.0
xdm	86	0	0.00	0	0.00	81	92	0.0
nfsd	87	0	0.00	0	0.00	13	19	0.0
xdm	88	0	0.00	0	0.00	82	97	2.0
xdm	89	0	0.00	0	0.00	82	97	2.0
xcterm	121	0	0.00	0	0.00	215	228	2.0
xdm	94	0	0.00	0	0.00	105	116	0.0
mwm	107	0	0.00	0	0.00	215	226	2.0
csh	111	0	0.00	0	0.00	43	50	0.0
prm	150	100	5.00	84	4.20	54	76	0.0
go	149	0	0.00	0	0.00	34	43	0.0
xcterm	105	0	0.00	0	0.00	212	225	2.0
stflmgc	151	183	9.15	149	7.45	142	188	2.0

PROCESS ID.	CONTEXT SWITCHING VOL.		INVOL.		PROCESS PRIORITY			TOTAL I/O ACT.	SIGNALS RECEIVED
	Total	Avg.	Total	Avg.	High	Low	Avg.		
0	0	0.0	0	0.0	35	35	35	0	0
1	0	0.0	0	0.0	45	45	45	140	15
2	0	0.0	0	0.0	-25	-25	-25	0	0
3	73	0.5	0	0.0	35	35	35	0	0
4	0	0.0	0	0.0	44	44	44	0	0
72	0	0.0	0	0.0	45	45	45	0	1
34	15	0.1	4	0.0	45	45	45	280	32
71	0	0.0	0	0.0	46	46	46	3	0
73	0	0.0	0	0.0	45	45	45	1	1
50	0	0.0	0	0.0	46	46	46	4	0
54	0	0.0	0	0.0	46	46	46	5	0
69	0	0.0	0	0.0	44	44	44	51	0
70	759	5.1	207	1.4	50	42	44	20	0
74	14	0.1	2	0.0	45	45	45	6	0
75	0	0.0	0	0.0	45	45	45	9	1
76	0	0.0	0	0.0	45	45	45	48	5
77	0	0.0	0	0.0	46	46	46	4	0
78	0	0.0	0	0.0	45	45	45	3	0
79	0	0.0	0	0.0	46	46	46	5	0
80	15	0.1	0	0.0	45	45	45	6	0
91	0	0.0	0	0.0	49	49	49	4	0
82	0	0.0	0	0.0	44	44	44	7	1
83	442	3.0	0	0.0	45	45	45	5	0
84	0	0.0	0	0.0	46	46	46	6	0
122	0	0.0	0	0.0	45	45	45	0	7
86	0	0.0	0	0.0	47	47	47	11	0
87	31	0.2	0	0.0	45	45	45	9	0
88	0	0.0	0	0.0	46	46	46	0	0
89	0	0.0	0	0.0	46	46	46	0	0
121	0	0.0	0	0.0	45	45	45	1	0
94	0	0.0	0	0.0	52	52	52	3	0
107	101	0.7	248	1.7	54	48	48	15	0
111	0	0.0	0	0.0	56	56	56	4	7
150	291	2.0	275	1.8	48	46	46	226	1
149	0	0.0	0	0.0	48	48	48	7	0
105	0	0.0	0	0.0	50	50	50	17	0
151	192	1.3	544	3.7	51	46	49	0	0

```
*********************************************
***** Process Resource Monitor Phase 2 *****
*****        Analysis Report            *****
*********************************************
```

LogTime from: Fri Aug 17 08:43:15 1990
to: Fri Aug 17 08:45:46 1990
Host Id: aries
CPU type: 68030
number: 2

COMMAND	PROCESS ID.	80% CPU UASAGE		50% CPU UASAGE		MEMORY COMSUMPTION		
		Milli Sec.	% of Total	Milli Sec.	% of Total	Phys	Virt	Shr
SWAPPER	0	0	0.00	0	0.00	0	0	0.0
init	1	0	0.00	0	0.00	15	20	0.0
PAGEDAEMON	2	0	0.00	0	0.00	8	2056	0.0
lwp1	3	0	0.00	0	0.00	8	8	0.0
lwp2	4	0	0.00	0	0.00	8	8	0.0
getty	72	0	0.00	0	0.00	12	18	0.0
update	34	0	0.00	0	0.00	3	8	0.0
getty	71	0	0.00	0	0.00	12	18	0.0
getty	73	0	0.00	0	0.00	12	18	0.0
errdemon	50	0	0.00	0	0.00	10	16	0.0
cron	54	0	0.00	0	0.00	31	36	0.0
Xgcm	69	0	0.00	0	0.00	171	198	0.0
Xgcm	70	167	8.35	101	5.05	1858	3136	0.0
portmap	74	0	0.00	0	0.00	22	29	0.0
netd	75	0	0.00	0	0.00	24	31	0.0
rwhod	76	0	0.00	0	0.00	24	29	0.0
talkd	77	0	0.00	0	0.00	25	32	0.0
rwalld	78	0	0.00	0	0.00	16	22	0.0
rusersd	79	0	0.00	0	0.00	22	29	0.0
timed	80	0	0.00	0	0.00	31	39	0.0
xdm	91	0	0.00	0	0.00	104	115	2.0
ypbind	82	0	0.00	0	0.00	18	24	0.0
biod	83	0	0.00	0	0.00	11	11	0.0
mountd	84	0	0.00	0	0.00	25	31	0.0
csh	372	0	0.00	0	0.00	43	50	0.0
xdm	86	0	0.00	0	0.00	81	92	0.0
nfsd	87	0	0.00	0	0.00	13	19	0.0
xdm	88	0	0.00	0	0.00	82	97	2.0
xdm	89	0	0.00	0	0.00	82	97	2.0
xcterm	371	0	0.00	0	0.00	193	227	2.0
csh	361	0	0.00	0	0.00	43	50	0.0
xdm	254	0	0.00	0	0.00	105	116	0.0
xcterm	355	0	0.00	0	0.00	190	224	2.0
stflmgc	429	200	10.00	183	9.15	142	188	2.0
mwm	357	0	0.00	0	0.00	204	226	2.0
csh	405	0	0.00	0	0.00	43	50	2.0
xcterm	404	0	0.00	0	0.00	193	227	2.0
prm	428	84	4.20	83	4.15	54	76	2.0

PROCESS ID.	CONTEXT SWITCHING VOL. Total	VOL. Avg.	INVOL. Total	INVOL. Avg.	PROCESS PRIORITY High	Low	Avg.	TOTAL I/O ACT.	SIGNALS RECEIVED
0	0	0.0	0	0.0	35	35	35	0	0
1	3	0.0	0	0.0	45	45	45	201	255
2	0	0.0	0	0.0	-25	-25	-25	0	0
3	75	0.5	0	0.0	35	35	35	0	0
4	0	0.0	0	0.0	44	44	44	0	0
72	0	0.0	0	0.0	45	45	45	0	1
34	9	0.1	0	0.0	46	45	45	5574	1998
71	0	0.0	0	0.0	46	46	46	3	0
73	0	0.0	0	0.0	45	45	45	1	1
50	0	0.0	0	0.0	46	46	46	4	0
54	0	0.0	0	0.0	45	45	45	57	33
69	0	0.0	0	0.0	45	45	45	51	33
70	861	5.8	79	0.5	48	41	43	89	30
74	14	0.1	0	0.0	45	45	45	6	0
75	0	0.0	0	0.0	45	45	45	15	1
76	6	0.0	0	0.0	45	45	45	1697	333
77	0	0.0	0	0.0	46	46	46	4	0
78	0	0.0	0	0.0	45	45	45	3	0
79	0	0.0	0	0.0	46	46	46	5	0
80	0	0.0	0	0.0	45	45	45	11	0
91	0	0.0	0	0.0	49	49	49	4	0
82	0	0.0	0	0.0	45	45	45	7	1
83	437	2.9	0	0.0	45	45	45	5	0
84	0	0.0	0	0.0	45	45	45	24	0
372	0	0.0	0	0.0	45	45	45	0	6
86	0	0.0	0	0.0	47	47	47	11	0
87	162	1.1	0	0.0	49	45	45	75	0
88	0	0.0	0	0.0	46	46	46	1	0
89	0	0.0	0	0.0	46	46	46	0	0
371	0	0.0	0	0.0	44	44	44	0	0
361	0	0.0	0	0.0	49	49	49	3	7
254	0	0.0	0	0.0	48	48	48	10	0
355	0	0.0	0	0.0	44	44	44	6	0
429	349	3.0	254	2.2	50	44	48	31	0
357	35	0.2	433	2.9	53	44	47	8	0
405	0	0.0	0	0.0	46	45	45	1	23
404	3	0.0	10	0.1	45	44	44	3	0
428	283	1.9	311	2.1	47	45	45	230	1

```
*********************************************
***** Process Resource Monitor Phase 2 *****
*****       Analysis Report             *****
*********************************************
```

LogTime from: Thu Aug 16 16:14:55 1990
to: Thu Aug 16 16:17:24 1990
Host Id: aries
CPU type: 68030
number: 2

COMMAND	PROCESS ID.	80% CPU UASAGE		50% CPU UASAGE		MEMORY COMSUMPTION		
		Milli Sec.	% of Total	Milli Sec.	% of Total	Phys	Virt	Shr
SWAPPER	0	0	0.00	0	0.00	0	0	0.0
init	1	0	0.00	0	0.00	15	20	0.0
PAGEDAEMON	2	0	0.00	0	0.00	8	2056	0.0
lwp1	3	0	0.00	0	0.00	8	8	0.0
lwp2	4	0	0.00	0	0.00	8	8	0.0
getty	72	0	0.00	0	0.00	12	18	0.0
update	34	0	0.00	0	0.00	3	8	0.0
getty	71	0	0.00	0	0.00	12	18	0.0
getty	73	0	0.00	0	0.00	12	18	0.0
errdemon	50	0	0.00	0	0.00	10	16	0.0
cron	54	0	0.00	0	0.00	30	36	0.0
Xgcm	69	0	0.00	0	0.00	170	196	0.0
Xgcm	70	133	6.65	116	5.80	690	1067	0.0
portmap	74	0	0.00	0	0.00	17	23	0.0
netd	75	0	0.00	0	0.00	24	31	0.0
rwhod	76	0	0.00	0	0.00	24	29	0.0
talkd	77	0	0.00	0	0.00	25	32	0.0
rwalld	78	0	0.00	0	0.00	16	22	0.0
rusersd	79	0	0.00	0	0.00	22	29	0.0
timed	80	0	0.00	0	0.00	31	39	0.0
xdm	91	0	0.00	0	0.00	104	115	2.0
ypbind	82	0	0.00	0	0.00	18	24	0.0
biod	83	0	0.00	0	0.00	11	11	0.0
mountd	84	0	0.00	0	0.00	24	31	0.0
csh	122	0	0.00	0	0.00	43	50	0.0
xdm	86	0	0.00	0	0.00	81	92	0.0
nfsd	87	0	0.00	0	0.00	13	19	0.0
xdm	88	0	0.00	0	0.00	82	97	2.0
xdm	89	0	0.00	0	0.00	82	97	2.0
xcterm	121	0	0.00	0	0.00	215	228	2.0
xdm	94	0	0.00	0	0.00	105	116	0.0
mwm	107	0	0.00	0	0.00	215	226	2.0
csh	111	0	0.00	0	0.00	43	50	0.0
go	128	0	0.00	0	0.00	34	43	0.0
prm	141	84	4.20	83	4.15	54	76	0.0
xcterm	105	0	0.00	0	0.00	212	225	2.0
stflmgc	142	200	10.00	167	8.35	142	188	2.0
sleep	144	0	0.00	0	0.00	5	13	2.0

PROCESS ID.	CONTEXT SWITCHING VOL.		INVOL.		PROCESS PRIORITY			TOTAL I/O ACT.	SIGNALS RECEIVED
	Total	Avg.	Total	Avg.	High	Low	Avg.		
0	0	0.0	0	0.0	35	35	35	0	0
1	0	0.0	0	0.0	45	45	45	138	13
2	0	0.0	0	0.0	-25	-25	-25	0	0
3	74	0.5	0	0.0	35	35	35	0	0
4	0	0.0	0	0.0	44	44	44	0	0
72	0	0.0	0	0.0	45	45	45	0	1
34	22	0.1	0	0.0	45	45	45	219	25
71	0	0.0	0	0.0	46	46	46	3	0
73	0	0.0	0	0.0	45	45	45	1	1
50	0	0.0	0	0.0	46	46	46	4	0
54	0	0.0	0	0.0	46	46	46	5	0
69	0	0.0	0	0.0	44	44	44	51	0
70	674	4.5	9	0.1	46	41	42	15	0
74	14	0.1	0	0.0	45	45	45	6	0
75	9	0.1	0	0.0	45	44	44	9	1
76	0	0.0	0	0.0	45	45	45	39	4
77	0	0.0	0	0.0	46	46	46	4	0
78	0	0.0	0	0.0	45	45	45	3	0
79	0	0.0	0	0.0	46	46	46	5	0
80	0	0.0	0	0.0	45	45	45	6	0
91	0	0.0	0	0.0	49	49	49	4	0
82	0	0.0	0	0.0	44	44	44	7	1
83	442	3.0	0	0.0	45	45	45	5	0
84	0	0.0	0	0.0	46	46	46	6	0
122	0	0.0	0	0.0	45	45	45	0	5
86	0	0.0	0	0.0	47	47	47	11	0
87	23	0.2	0	0.0	45	45	45	7	0
88	0	0.0	0	0.0	46	46	46	0	0
89	0	0.0	0	0.0	46	46	46	0	0
121	1	0.0	0	0.0	45	44	44	1	0
94	0	0.0	0	0.0	52	52	52	3	0
107	7	0.0	20	0.1	49	45	45	15	0
111	0	0.0	0	0.0	56	56	56	4	7
128	3	0.0	4	0.0	45	45	45	0	10
141	297	2.0	259	1.7	47	45	45	226	1
105	0	0.0	0	0.0	50	50	50	17	0
142	393	3.1	374	3.0	50	46	48	1	0
144	0	0.0	0	0.0	45	45	45	0	0

```
*********************************************
***** Process Resource Monitor Phase 2 *****
*****       Analysis Report             *****
*********************************************
```

LogTime from: Thu Aug 16 15:53:02 1990
to: Thu Aug 16 15:55:31 1990
Host Id: aries
CPU type: 68030
number: 2

COMMAND	PROCESS ID.	80% CPU UASAGE		50% CPU UASAGE		MEMORY COMSUMPTION		
		Milli Sec.	% of Total	Milli Sec.	% of Total	Phys	Virt	Shr
SWAPPER	0	0	0.00	0	0.00	0	0	0.0
init	1	0	0.00	0	0.00	15	20	0.0
PAGEDAEMON	2	0	0.00	0	0.00	8	2056	0.0
lwp1	3	0	0.00	0	0.00	8	8	0.0
lwp2	4	0	0.00	0	0.00	8	8	0.0
getty	72	0	0.00	0	0.00	12	18	0.0
update	34	0	0.00	0	0.00	3	8	0.0
getty	71	0	0.00	0	0.00	12	18	0.0
getty	73	0	0.00	0	0.00	12	18	0.0
errdemon	50	0	0.00	0	0.00	10	16	0.0
cron	54	0	0.00	0	0.00	31	36	0.0
Xgcm	69	0	0.00	0	0.00	170	196	0.0
Xgcm	70	150	7.50	116	5.80	503	687	0.0
portmap	74	0	0.00	0	0.00	17	23	0.0
netd	75	0	0.00	0	0.00	25	31	0.0
rwhod	76	0	0.00	0	0.00	24	29	0.0
talkd	77	0	0.00	0	0.00	25	32	0.0
rwalld	78	0	0.00	0	0.00	16	22	0.0
rusersd	79	0	0.00	0	0.00	22	29	0.0
timed	80	0	0.00	0	0.00	31	39	0.0
xdm	302	0	0.00	0	0.00	105	116	0.0
ypbind	82	0	0.00	0	0.00	18	24	0.0
biod	83	17	0.85	16	0.80	11	11	0.0
mountd	84	0	0.00	0	0.00	24	31	0.0
xcterm	326	0	0.00	0	0.00	190	226	2.0
xdm	86	0	0.00	0	0.00	81	92	0.0
nfsd	87	0	0.00	0	0.00	13	19	0.0
xdm	88	0	0.00	0	0.00	82	97	2.0
xdm	89	0	0.00	0	0.00	82	97	2.0
csh	327	0	0.00	0	0.00	43	50	0.0
xdm	94	0	0.00	0	0.00	104	115	2.0
xcterm	310	0	0.00	0	0.00	188	224	2.0
mwm	312	0	0.00	0	0.00	204	226	2.0
go	346	0	0.00	0	0.00	34	43	0.0
csh	316	0	0.00	0	0.00	43	50	0.0
prm	377	1000	50.00	316	15.80	78	326	0.0
stfloat	381	233	11.65	200	10.00	142	188	2.0
prm	380	83	4.15	67	3.35	59	76	2.0
sleep	382	0	0.00	0	0.00	5	13	0.0
stnofmt	387	383	19.15	0	0.00	141	188	2.0
prm	386	0	0.00	0	0.00	50	76	2.0

PROCESS ID.	CONTEXT SWITCHING VOL. Total	VOL. Avg.	INVOL. Total	INVOL. Avg.	PROCESS PRIORITY High	Low	Avg.	TOTAL I/O ACT.	SIGNALS RECEIVED
0	0	0.0	0	0.0	35	35	35	0	0
1	3	0.0	0	0.0	45	45	45	284	59
2	0	0.0	0	0.0	-25	-25	-25	0	0
3	73	0.5	0	0.0	35	35	35	0	0
4	0	0.0	0	0.0	44	44	44	0	0
72	0	0.0	0	0.0	45	45	45	0	1
34	31	0.2	4	0.0	45	45	45	2216	270
71	0	0.0	0	0.0	47	47	47	4	0
73	0	0.0	0	0.0	45	45	45	1	1
50	0	0.0	0	0.0	46	46	46	4	0
54	0	0.0	0	0.0	46	45	45	29	6
69	0	0.0	0	0.0	40	40	40	30	4
70	830	5.6	127	0.9	48	41	42	99	0
74	14	0.1	0	0.0	45	45	45	5	0
75	0	0.0	0	0.0	45	44	44	22	1
76	7	0.0	0	0.0	45	45	45	371	45
77	0	0.0	0	0.0	45	45	45	4	0
78	0	0.0	0	0.0	45	45	45	3	0
79	0	0.0	0	0.0	45	45	45	5	0
80	4	0.0	0	0.0	45	45	45	12	0
302	0	0.0	0	0.0	51	51	51	12	0
82	0	0.0	0	0.0	44	44	44	5	1
83	444	3.0	0	0.0	46	45	45	15	0
84	0	0.0	0	0.0	44	44	44	6	0
326	15	0.1	31	0.2	46	44	45	3	0
86	0	0.0	0	0.0	48	48	48	12	0
87	0	0.0	0	0.0	45	45	45	22	0
88	0	0.0	0	0.0	46	46	46	1	0
89	0	0.0	0	0.0	46	46	46	0	0
327	0	0.0	0	0.0	45	45	45	0	18
94	0	0.0	0	0.0	44	44	44	2	0
310	0	0.0	0	0.0	44	44	44	14	0
312	3	0.0	52	0.3	47	45	45	8	0
346	6	0.0	4	0.0	46	45	45	2	32
316	0	0.0	0	0.0	52	52	52	9	7
377	3	0.6	54	10.8	61	47	51	231	1
381	476	4.5	380	3.6	52	44	48	30	0
380	277	1.9	312	2.1	47	45	45	226	1
382	13	0.3	8	0.2	45	45	45	0	1
387	0	0.0	0	0.0	48	44	45	0	0
386	0	0.0	0	0.0	50	50	50	1	0

```
*********************************************
***** Process Resource Monitor Phase 2 *****
*****       Analysis Report             *****
*********************************************
```

LogTime from: Thu Aug 16 15:42:10 1990
to: Thu Aug 16 15:44:40 1990
Host Id: aries
CPU type: 68030
number: 2

COMMAND	PROCESS ID.	80% CPU UASAGE		50% CPU UASAGE		MEMORY COMSUMPTION		
		Milli Sec.	% of Total	Milli Sec.	% of Total	Phys	Virt	Shr
SWAPPER	0	0	0.00	0	0.00	0	0	0.0
init	1	0	0.00	0	0.00	15	20	0.0
PAGEDAEMON	2	0	0.00	0	0.00	8	2056	0.0
lwp1	3	0	0.00	0	0.00	8	8	0.0
lwp2	4	0	0.00	0	0.00	8	8	0.0
getty	72	0	0.00	0	0.00	12	18	0.0
update	34	0	0.00	0	0.00	3	8	0.0
getty	71	0	0.00	0	0.00	12	18	0.0
getty	73	0	0.00	0	0.00	12	18	0.0
errdemon	50	0	0.00	0	0.00	10	16	0.0
cron	54	0	0.00	0	0.00	31	36	0.0
Xgcm	69	0	0.00	0	0.00	170	196	0.0
Xgcm	70	134	6.70	117	5.85	412	559	0.0
portmap	74	0	0.00	0	0.00	17	23	0.0
netd	75	0	0.00	0	0.00	25	31	0.0
rwhod	76	0	0.00	0	0.00	24	29	0.0
talkd	77	0	0.00	0	0.00	25	32	0.0
rwalld	78	0	0.00	0	0.00	16	22	0.0
rusersd	79	0	0.00	0	0.00	22	29	0.0
timed	80	0	0.00	0	0.00	31	39	0.0
xdm	302	0	0.00	0	0.00	105	116	0.0
ypbind	82	0	0.00	0	0.00	18	24	0.0
biod	83	17	0.85	0	0.00	11	11	0.0
mountd	84	0	0.00	0	0.00	24	31	0.0
xcterm	326	0	0.00	0	0.00	190	226	2.0
xdm	86	0	0.00	0	0.00	81	92	0.0
nfsd	87	0	0.00	0	0.00	13	19	0.0
xdm	88	0	0.00	0	0.00	82	97	2.0
xdm	89	0	0.00	0	0.00	82	97	2.0
csh	327	0	0.00	0	0.00	43	50	0.0
xdm	94	0	0.00	0	0.00	104	115	2.0
xcterm	310	0	0.00	0	0.00	188	224	2.0
mwm	312	0	0.00	0	0.00	204	226	2.0
go	346	0	0.00	0	0.00	34	43	0.0
csh	316	0	0.00	0	0.00	43	50	0.0
prm	362	100	5.00	99	4.95	81	332	0.0
stfloat	366	184	9.20	167	8.35	142	188	2.0
prm	365	100	5.00	83	4.15	59	76	2.0
sleep	367	0	0.00	0	0.00	5	13	0.0

PROCESS ID.	CONTEXT SWITCHING VOL. Total	VOL. Avg.	INVOL. Total	INVOL. Avg.	PROCESS PRIORITY High	Low	Avg.	TOTAL I/O ACT.	SIGNALS RECEIVED
0	0	0.0	0	0.0	35	35	35	0	0
1	0	0.0	0	0.0	45	45	45	278	54
2	0	0.0	0	0.0	-25	-25	-25	0	0
3	73	0.5	0	0.0	35	35	35	0	0
4	0	0.0	0	0.0	44	44	44	0	0
72	0	0.0	0	0.0	45	45	45	0	1
34	11	0.1	0	0.0	45	44	44	2074	248
71	0	0.0	0	0.0	47	47	47	4	0
73	0	0.0	0	0.0	45	45	45	1	1
50	0	0.0	0	0.0	46	46	46	4	0
54	0	0.0	0	0.0	45	45	45	19	4
69	0	0.0	0	0.0	40	40	40	30	4
70	690	4.6	33	0.2	48	41	44	81	0
74	0	0.0	0	0.0	45	45	45	5	0
75	0	0.0	0	0.0	44	44	44	18	1
76	15	0.1	0	0.0	45	45	45	335	41
77	0	0.0	0	0.0	45	45	45	4	0
78	0	0.0	0	0.0	45	45	45	3	0
79	0	0.0	0	0.0	45	45	45	5	0
80	0	0.0	0	0.0	45	45	45	11	0
302	0	0.0	0	0.0	51	51	51	12	0
82	0	0.0	0	0.0	44	44	44	5	1
83	559	3.8	0	0.0	45	45	45	15	0
84	0	0.0	0	0.0	44	44	44	6	0
326	4	0.0	9	0.1	45	44	44	3	0
86	0	0.0	0	0.0	48	48	48	12	0
87	0	0.0	0	0.0	45	45	45	19	0
88	0	0.0	0	0.0	46	46	46	1	0
89	0	0.0	0	0.0	46	46	46	0	0
327	0	0.0	0	0.0	45	45	45	0	18
94	0	0.0	0	0.0	44	44	44	2	0
310	0	0.0	0	0.0	44	44	44	14	0
312	1	0.0	81	0.5	47	45	45	8	0
346	3	0.0	5	0.0	46	45	45	1	19
316	0	0.0	0	0.0	52	52	52	9	7
362	81	2.0	93	2.3	72	47	48	239	1
366	468	3.8	445	3.6	54	44	51	30	0
365	268	1.8	372	2.5	49	46	47	227	1
367	0	0.0	0	0.0	45	45	45	0	0

```
*********************************************
***** Process Resource Monitor Phase 2 *****
*****       Analysis Report             *****
*********************************************
```

LogTime from: Fri Aug 17 08:35:34 1990
to: Fri Aug 17 08:37:32 1990
Host Id: aries
CPU type: 68030
number: 2

COMMAND	PROCESS ID.	80% CPU UASAGE		50% CPU UASAGE		MEMORY COMSUMPTION		
		Milli Sec.	% of Total	Milli Sec.	% of Total	Phys	Virt	Shr
SWAPPER	0	0	0.00	0	0.00	0	0	0.0
init	1	0	0.00	0	0.00	15	20	0.0
PAGEDAEMON	2	0	0.00	0	0.00	8	2056	0.0
lwp1	3	0	0.00	0	0.00	8	8	0.0
lwp2	4	0	0.00	0	0.00	8	8	0.0
getty	72	0	0.00	0	0.00	12	18	0.0
update	34	0	0.00	0	0.00	3	8	0.0
getty	71	0	0.00	0	0.00	12	18	0.0
getty	73	0	0.00	0	0.00	12	18	0.0
errdemon	50	0	0.00	0	0.00	10	16	0.0
cron	54	0	0.00	0	0.00	31	36	0.0
Xgcm	69	0	0.00	0	0.00	171	198	0.0
Xgcm	70	200	10.00	166	8.30	1858	3136	0.0
portmap	74	0	0.00	0	0.00	22	29	0.0
netd	75	0	0.00	0	0.00	24	31	0.0
rwhod	76	0	0.00	0	0.00	24	29	0.0
talkd	77	0	0.00	0	0.00	25	32	0.0
rwalld	78	0	0.00	0	0.00	16	22	0.0
rusersd	79	0	0.00	0	0.00	22	29	0.0
timed	80	0	0.00	0	0.00	31	39	0.0
xdm	91	0	0.00	0	0.00	104	115	2.0
ypbind	82	0	0.00	0	0.00	18	24	0.0
biod	83	16	0.80	0	0.00	11	11	0.0
mountd	84	0	0.00	0	0.00	25	31	0.0
csh	372	0	0.00	0	0.00	43	50	0.0
xdm	86	0	0.00	0	0.00	81	92	0.0
nfsd	87	0	0.00	0	0.00	13	19	0.0
xdm	88	0	0.00	0	0.00	82	97	2.0
xdm	89	0	0.00	0	0.00	82	97	2.0
xcterm	371	0	0.00	0	0.00	193	227	2.0
csh	361	0	0.00	0	0.00	43	50	0.0
xdm	254	0	0.00	0	0.00	105	116	0.0
xcterm	355	0	0.00	0	0.00	190	224	2.0
stingc	401	116	5.80	84	4.20	141	188	2.0
mwm	357	0	0.00	0	0.00	204	226	2.0
xcterm	380	0	0.00	0	0.00	190	224	2.0
csh	381	0	0.00	0	0.00	43	50	0.0
prm	400	100	5.00	83	4.15	67	76	2.0
xcterm	404	67	3.35	33	1.65	190	224	2.0
csh	405	0	0.00	0	0.00	43	50	2.0
ls	408	0	0.00	0	0.00	19	39	2.0
ls	409	0	0.00	0	0.00	43	50	2.0
prm	411	67	3.35	0	0.00	80	91	2.0

PROCESS ID.	CONTEXT SWITCHING VOL. Total	VOL. Avg.	INVOL. Total	INVOL. Avg.	PROCESS PRIORITY High	Low	Avg.	TOTAL I/O ACT.	SIGNALS RECEIVED
0	0	0.0	0	0.0	35	35	35	0	0
1	482	4.1	79	0.7	45	45	45	197	252
2	1	0.0	0	0.0	-25	-25	-25	0	0
3	58539	496.1	0	0.0	35	35	35	0	0
4	1	0.0	0	0.0	44	44	44	0	0
72	14	0.1	22	0.2	45	45	45	0	1
34	5046	42.8	396	3.4	46	44	45	5449	1981
71	9	0.1	24	0.2	46	46	46	3	0
73	11	0.1	17	0.1	45	45	45	1	1
50	5	0.0	6	0.1	46	46	46	4	0
54	212	1.8	111	0.9	45	45	45	57	33
69	264	2.2	149	1.3	40	40	40	51	32
70	76881	651.5	8198	69.5	49	40	42	82	30
74	898	7.6	107	0.9	45	44	44	6	0
75	2170	18.4	105	0.9	45	45	45	15	1
76	3693	31.3	80	0.7	45	45	45	1671	330
77	35	0.3	30	0.3	46	46	46	4	0
78	41	0.3	32	0.3	45	45	45	3	0
79	65	0.6	51	0.4	46	46	46	5	0
80	2846	24.1	56	0.5	45	44	44	11	0
91	35	0.3	95	0.8	49	49	49	4	0
82	316	2.7	65	0.6	45	45	45	7	1
83	17382	147.3	14	0.1	45	45	45	5	0
84	119	1.0	30	0.3	45	45	45	24	0
372	141	1.2	187	1.6	45	45	45	0	6
86	25	0.2	46	0.4	47	47	47	11	0
87	1313	11.1	16	0.1	45	45	45	18	0
88	10	0.1	60	0.5	46	46	46	1	0
89	4	0.0	6	0.1	46	46	46	0	0
371	366	3.1	1489	12.6	44	44	44	0	0
361	151	1.3	155	1.3	49	49	49	3	7
254	172	1.5	304	2.6	48	48	48	10	0
355	144	1.2	316	2.7	44	44	44	6	0
401	138	2.1	380	5.8	47	44	46	31	0
357	737	6.2	11593	98.2	52	44	49	8	0
380	3	0.0	16	0.2	44	44	44	0	0
381	0	0.0	0	0.0	47	47	47	0	17
400	221	1.9	326	2.8	46	45	45	181	1
404	97	2.1	321	6.8	50	44	45	3	0
405	20	0.4	22	0.5	52	44	45	1	5
408	0	0.0	0	0.0	48	48	48	0	0
409	0	0.0	0	0.0	45	45	45	0	0
411	25	1.2	85	4.0	48	45	46	5	1

```
*********************************************
***** Process Resource Monitor Phase 2 *****
*****       Analysis Report             *****
*********************************************
```

LogTime from: Fri Aug 17 08:39:25 1990
to: Fri Aug 17 08:41:54 1990
Host Id: aries
CPU type: 68030
number: 2

COMMAND	PROCESS ID.	80% CPU UASAGE		50% CPU UASAGE		MEMORY COMSUMPTION		
		Milli Sec.	% of Total	Milli Sec.	% of Total	Phys	Virt	Shr
SWAPPER	0	0	0.00	0	0.00	0	0	0.0
init	1	0	0.00	0	0.00	15	20	0.0
PAGEDAEMON	2	0	0.00	0	0.00	8	2056	0.0
lwp1	3	0	0.00	0	0.00	8	8	0.0
lwp2	4	0	0.00	0	0.00	8	8	0.0
getty	72	0	0.00	0	0.00	12	18	0.0
update	34	0	0.00	0	0.00	3	8	0.0
getty	71	0	0.00	0	0.00	12	18	0.0
getty	73	0	0.00	0	0.00	12	18	0.0
errdemon	50	0	0.00	0	0.00	10	16	0.0
cron	54	0	0.00	0	0.00	31	36	0.0
Xgcm	69	0	0.00	0	0.00	171	198	0.0
Xgcm	70	216	10.80	0	0.00	1858	3136	0.0
portmap	74	0	0.00	0	0.00	22	29	0.0
netd	75	0	0.00	0	0.00	24	31	0.0
rwhod	76	0	0.00	0	0.00	24	29	0.0
talkd	77	0	0.00	0	0.00	25	32	0.0
rwalld	78	0	0.00	0	0.00	16	22	0.0
rusersd	79	0	0.00	0	0.00	22	29	0.0
timed	80	0	0.00	0	0.00	31	39	0.0
xdm	91	0	0.00	0	0.00	104	115	2.0
ypbind	82	0	0.00	0	0.00	18	24	0.0
biod	83	17	0.85	0	0.00	11	11	0.0
mountd	84	0	0.00	0	0.00	25	31	0.0
csh	372	0	0.00	0	0.00	43	50	0.0
xdm	86	0	0.00	0	0.00	81	92	0.0
nfsd	87	0	0.00	0	0.00	13	19	0.0
xdm	88	0	0.00	0	0.00	82	97	2.0
xdm	89	0	0.00	0	0.00	82	97	2.0
xcterm	371	0	0.00	0	0.00	193	227	2.0
csh	361	0	0.00	0	0.00	43	50	0.0
xdm	254	0	0.00	0	0.00	105	116	0.0
xcterm	355	0	0.00	0	0.00	190	224	2.0
stinmgc	421	116	5.80	100	5.00	141	188	2.0
mwm	357	0	0.00	0	0.00	204	226	2.0
csh	405	0	0.00	0	0.00	43	50	2.0
xcterm	404	0	0.00	0	0.00	190	224	2.0
prm	420	84	4.20	83	4.15	54	76	2.0

PROCESS ID.	CONTEXT SWITCHING VOL.		INVOL.		PROCESS PRIORITY			TOTAL I/O ACT.	SIGNALS RECEIVED
	Total	Avg.	Total	Avg.	High	Low	Avg.		
0	0	0.0	0	0.0	35	35	35	0	0
1	3	0.0	0	0.0	45	45	45	199	254
2	0	0.0	0	0.0	-25	-25	-25	0	0
3	73	0.5	0	0.0	35	35	35	0	0
4	0	0.0	0	0.0	44	44	44	0	0
72	0	0.0	0	0.0	45	45	45	0	1
34	29	0.2	4	0.0	46	45	45	5511	1990
71	0	0.0	0	0.0	46	46	46	3	0
73	0	0.0	0	0.0	45	45	45	1	1
50	0	0.0	0	0.0	46	46	46	4	0
54	0	0.0	0	0.0	45	45	45	57	33
69	0	0.0	0	0.0	45	40	43	51	33
70	626	4.2	490	3.3	52	40	44	83	30
74	0	0.0	0	0.0	44	44	44	6	0
75	9	0.1	0	0.0	45	45	45	15	1
76	8	0.1	0	0.0	45	45	45	1684	332
77	0	0.0	0	0.0	46	46	46	4	0
78	0	0.0	0	0.0	45	45	45	3	0
79	0	0.0	0	0.0	46	46	46	5	0
80	4	0.0	0	0.0	45	45	45	11	0
91	0	0.0	0	0.0	49	49	49	4	0
82	0	0.0	0	0.0	45	45	45	7	1
83	438	2.9	0	0.0	46	45	45	5	0
84	0	0.0	0	0.0	45	45	45	24	0
372	0	0.0	0	0.0	45	45	45	0	6
86	0	0.0	0	0.0	47	47	47	11	0
87	0	0.0	0	0.0	45	45	45	18	0
88	0	0.0	0	0.0	46	46	46	1	0
89	0	0.0	0	0.0	46	46	46	0	0
371	0	0.0	0	0.0	44	44	44	0	0
361	0	0.0	0	0.0	49	49	49	3	7
254	0	0.0	0	0.0	48	48	48	10	0
355	0	0.0	0	0.0	44	44	44	6	0
421	266	4.0	401	6.1	49	44	47	31	0
357	35	0.2	451	3.0	53	45	46	8	0
405	2	0.0	0	0.0	46	45	45	1	15
404	3	0.0	6	0.0	45	45	45	3	0
420	279	1.9	348	2.3	47	45	45	227	1

```
*********************************************
***** Process Resource Monitor Phase 2 *****
*****        Analysis Report            *****
*********************************************
```

LogTime from: Thu Aug 16 15:36:20 1990
to: Thu Aug 16 15:38:49 1990
Host Id: aries
CPU type: 68030
number: 2

COMMAND	PROCESS ID.	80% CPU UASAGE		50% CPU UASAGE		MEMORY COMSUMPTION		
		Milli Sec.	% of Total	Milli Sec.	% of Total	Phys	Virt	Shr
SWAPPER	0	0	0.00	0	0.00	0	0	0.0
init	1	0	0.00	0	0.00	15	20	0.0
PAGEDAEMON	2	0	0.00	0	0.00	8	2056	0.0
lwp1	3	0	0.00	0	0.00	8	8	0.0
lwp2	4	0	0.00	0	0.00	8	8	0.0
getty	72	0	0.00	0	0.00	12	18	0.0
update	34	0	0.00	0	0.00	3	8	0.0
getty	71	0	0.00	0	0.00	12	18	0.0
getty	73	0	0.00	0	0.00	12	18	0.0
errdemon	50	0	0.00	0	0.00	10	16	0.0
cron	54	0	0.00	0	0.00	31	36	0.0
Xgcm	69	0	0.00	0	0.00	170	196	0.0
Xgcm	70	233	11.65	183	9.15	412	559	0.0
portmap	74	0	0.00	0	0.00	17	23	0.0
netd	75	0	0.00	0	0.00	25	31	0.0
rwhod	76	0	0.00	0	0.00	24	29	0.0
talkd	77	0	0.00	0	0.00	25	32	0.0
rwalld	78	0	0.00	0	0.00	16	22	0.0
rusersd	79	0	0.00	0	0.00	22	29	0.0
timed	80	0	0.00	0	0.00	31	39	0.0
xdm	302	0	0.00	0	0.00	105	116	0.0
ypbind	82	0	0.00	0	0.00	18	24	0.0
biod	83	17	0.85	0	0.00	11	11	0.0
mountd	84	0	0.00	0	0.00	24	31	0.0
xcterm	326	0	0.00	0	0.00	190	226	2.0
xdm	86	0	0.00	0	0.00	81	92	0.0
nfsd	87	0	0.00	0	0.00	13	19	0.0
xdm	88	0	0.00	0	0.00	82	97	2.0
xdm	89	0	0.00	0	0.00	82	97	2.0
csh	327	0	0.00	0	0.00	43	50	0.0
xdm	94	0	0.00	0	0.00	104	115	2.0
xcterm	310	0	0.00	0	0.00	188	224	2.0
mwm	312	0	0.00	0	0.00	204	226	2.0
go	346	0	0.00	0	0.00	34	43	0.0
csh	316	0	0.00	0	0.00	43	50	0.0
stint	357	100	5.00	67	3.35	142	188	2.0
prm	356	84	4.20	83	4.15	54	76	0.0
go	358	0	0.00	0	0.00	34	43	2.0
go	360	101	5.05	67	3.35	178	188	2.0
prm	359	83	4.15	67	3.35	54	76	0.0

PROCESS ID.	CONTEXT SWITCHING VOL. Total	VOL. Avg.	INVOL. Total	INVOL. Avg.	PROCESS PRIORITY High	Low	Avg.	TOTAL I/O ACT.	SIGNALS RECEIVED
0	0	0.0	0	0.0	35	35	35	0	0
1	0	0.0	0	0.0	45	45	45	276	53
2	0	0.0	0	0.0	-25	-25	-25	0	0
3	73	0.5	0	0.0	35	35	35	0	0
4	0	0.0	0	0.0	44	44	44	0	0
72	0	0.0	0	0.0	45	45	45	0	1
34	26	0.2	2	0.0	45	44	44	2000	237
71	0	0.0	0	0.0	47	47	47	4	0
73	0	0.0	0	0.0	45	45	45	1	1
50	0	0.0	0	0.0	46	46	46	4	0
54	0	0.0	0	0.0	45	45	45	19	4
69	0	0.0	0	0.0	40	40	40	30	3
70	793	5.3	53	0.4	47	40	43	79	0
74	0	0.0	0	0.0	45	45	45	5	0
75	0	0.0	0	0.0	44	44	44	16	1
76	8	0.1	0	0.0	45	45	45	314	39
77	0	0.0	0	0.0	45	45	45	4	0
78	0	0.0	0	0.0	45	45	45	3	0
79	0	0.0	0	0.0	45	45	45	5	0
80	17	0.1	0	0.0	45	44	44	11	0
302	0	0.0	0	0.0	51	51	51	12	0
82	0	0.0	0	0.0	44	44	44	5	1
83	696	4.7	0	0.0	47	45	45	15	0
84	0	0.0	0	0.0	44	44	44	6	0
326	3	0.0	4	0.0	45	44	44	3	0
86	0	0.0	0	0.0	48	48	48	12	0
87	22	0.1	0	0.0	45	45	45	19	0
88	0	0.0	0	0.0	46	46	46	1	0
89	0	0.0	0	0.0	46	46	46	0	0
327	0	0.0	0	0.0	45	45	45	0	18
94	0	0.0	0	0.0	44	44	44	2	0
310	0	0.0	0	0.0	44	44	44	14	0
312	14	0.1	155	1.0	48	45	46	8	0
346	0	0.0	0	0.0	46	45	45	0	11
316	0	0.0	0	0.0	52	52	52	9	7
357	137	1.7	333	4.2	48	44	46	30	0
356	299	2.0	274	1.8	48	45	45	227	1
358	13	0.3	6	0.2	45	45	45	0	0
360	313	10.4	115	3.8	54	45	47	145	0
359	66	2.2	65	2.2	48	45	46	41	1

```
*********************************************
***** Process Resource Monitor Phase 2 *****
*****       Analysis Report             *****
*********************************************
```

LogTime from: Thu Aug 16 15:30:51 1990
to: Thu Aug 16 15:33:20 1990
Host Id: aries
CPU type: 68030
number: 2

COMMAND	PROCESS ID.	50% CPU UASAGE		50% CPU UASAGE		MEMORY COMSUMPTION		
		Milli Sec.	% of Total	Milli Sec.	% of Total	Phys	Virt	Shr
SWAPPER	0	0	0.00	0	0.00	0	0	0.0
init	1	0	0.00	0	0.00	15	20	0.0
PAGEDAEMON	2	0	0.00	0	0.00	8	2056	0.0
lwp1	3	0	0.00	0	0.00	8	8	0.0
lwp2	4	0	0.00	0	0.00	8	8	0.0
getty	72	0	0.00	0	0.00	12	18	0.0
update	34	0	0.00	0	0.00	3	8	0.0
getty	71	0	0.00	0	0.00	12	18	0.0
getty	73	0	0.00	0	0.00	12	18	0.0
errdemon	50	0	0.00	0	0.00	10	16	0.0
cron	54	0	0.00	0	0.00	31	36	0.0
Xgcm	69	0	0.00	0	0.00	170	196	0.0
Xgcm	70	117	5.85	101	5.05	412	559	0.0
portmap	74	0	0.00	0	0.00	17	23	0.0
netd	75	0	0.00	0	0.00	25	31	0.0
rwhod	76	0	0.00	0	0.00	24	29	0.0
talkd	77	0	0.00	0	0.00	25	32	0.0
rwalld	78	0	0.00	0	0.00	16	22	0.0
rusersd	79	0	0.00	0	0.00	22	29	0.0
timed	80	0	0.00	0	0.00	31	39	0.0
xdm	302	0	0.00	0	0.00	105	116	0.0
ypbind	82	0	0.00	0	0.00	18	24	0.0
biod	83	17	0.85	0	0.00	11	11	0.0
mountd	84	0	0.00	0	0.00	24	31	0.0
xcterm	326	0	0.00	0	0.00	190	226	2.0
xdm	86	0	0.00	0	0.00	81	92	0.0
nfsd	87	0	0.00	0	0.00	13	19	0.0
xdm	88	0	0.00	0	0.00	82	97	2.0
xdm	89	0	0.00	0	0.00	82	97	2.0
csh	327	0	0.00	0	0.00	43	50	0.0
xdm	94	0	0.00	0	0.00	104	115	2.0
xcterm	310	0	0.00	0	0.00	188	224	2.0
mwm	312	0	0.00	0	0.00	204	226	2.0
go	346	0	0.00	0	0.00	34	43	0.0
csh	316	0	0.00	0	0.00	43	50	0.0
stnfgc	351	34	1.70	33	1.65	141	188	2.0
prm	350	83	4.15	67	3.35	54	76	0.0
sleep	352	0	0.00	0	0.00	5	13	2.0

PROCESS ID.	CONTEXT SWITCHING VOL. Total	VOL. Avg.	INVOL. Total	INVOL. Avg.	PROCESS PRIORITY High	Low	Avg.	TOTAL I/O ACT.	SIGNALS RECEIVED
0	0	0.0	0	0.0	35	35	35	0	0
1	0	0.0	0	0.0	45	45	45	274	52
2	0	0.0	0	0.0	-25	-25	-25	0	0
3	72	0.5	0	0.0	35	35	35	0	0
4	0	0.0	0	0.0	44	44	44	0	0
72	0	0.0	0	0.0	45	45	45	0	1
34	19	0.1	2	0.0	46	45	45	1925	226
71	0	0.0	0	0.0	47	47	47	4	0
73	0	0.0	0	0.0	45	45	45	1	1
50	0	0.0	0	0.0	46	46	46	4	0
54	0	0.0	0	0.0	45	45	45	19	4
69	0	0.0	0	0.0	40	40	40	30	3
70	735	4.9	6	0.0	45	41	42	76	0
74	0	0.0	0	0.0	45	45	45	5	0
75	0	0.0	0	0.0	45	45	45	14	1
76	3	0.0	0	0.0	45	45	45	302	37
77	0	0.0	0	0.0	45	45	45	4	0
78	0	0.0	0	0.0	45	45	45	3	0
79	0	0.0	0	0.0	45	45	45	5	0
80	4	0.0	0	0.0	45	45	45	10	0
302	0	0.0	0	0.0	51	51	51	12	0
82	0	0.0	0	0.0	44	44	44	5	1
83	444	3.0	0	0.0	46	45	45	15	0
84	0	0.0	0	0.0	44	44	44	6	0
326	1	0.0	2	0.0	46	44	45	3	0
86	0	0.0	0	0.0	48	48	48	12	0
87	0	0.0	0	0.0	45	45	45	16	0
88	0	0.0	0	0.0	46	46	46	1	0
89	0	0.0	0	0.0	46	46	46	0	0
327	0	0.0	0	0.0	45	45	45	0	18
94	0	0.0	0	0.0	44	44	44	2	0
310	0	0.0	0	0.0	44	44	44	14	0
312	4	0.0	4	0.0	46	45	45	8	0
346	3	0.0	3	0.0	46	45	45	0	4
316	0	0.0	0	0.0	52	52	52	9	7
351	364	3.0	110	0.9	48	44	45	30	0
350	293	2.0	336	2.3	47	45	45	226	1
352	13	0.5	4	0.1	45	45	45	0	0

```
*********************************************
***** Process Resource Monitor Phase 2 *****
*****       Analysis Report             *****
*********************************************
```

LogTime from: Thu Aug 16 15:33:35 1990
to: Thu Aug 16 15:36:05 1990
Host Id: aries
CPU type: 68030
number: 2

COMMAND	PROCESS ID.	80% CPU UASAGE		50% CPU UASAGE		MEMORY COMSUMPTION		
		Milli Sec.	% of Total	Milli Sec.	% of Total	Phys	Virt	Shr
SWAPPER	0	0	0.00	0	0.00	0	0	0.0
init	1	0	0.00	0	0.00	15	20	0.0
PAGEDAEMON	2	0	0.00	0	0.00	8	2056	0.0
lwp1	3	0	0.00	0	0.00	8	8	0.0
lwp2	4	0	0.00	0	0.00	8	8	0.0
getty	72	0	0.00	0	0.00	12	18	0.0
update	34	0	0.00	0	0.00	3	8	0.0
getty	71	0	0.00	0	0.00	12	18	0.0
getty	73	0	0.00	0	0.00	12	18	0.0
errdemon	50	0	0.00	0	0.00	10	16	0.0
cron	54	0	0.00	0	0.00	31	36	0.0
Xgcm	69	0	0.00	0	0.00	170	196	0.0
Xgcm	70	150	7.50	133	6.65	412	559	0.0
portmap	74	0	0.00	0	0.00	17	23	0.0
netd	75	0	0.00	0	0.00	25	31	0.0
rwhod	76	0	0.00	0	0.00	24	29	0.0
talkd	77	0	0.00	0	0.00	25	32	0.0
rwalld	78	0	0.00	0	0.00	16	22	0.0
rusersd	79	0	0.00	0	0.00	22	29	0.0
timed	80	0	0.00	0	0.00	31	39	0.0
xdm	302	0	0.00	0	0.00	105	116	0.0
ypbind	82	0	0.00	0	0.00	18	24	0.0
biod	83	17	0.85	0	0.00	11	11	0.0
mountd	84	0	0.00	0	0.00	24	31	0.0
xcterm	326	0	0.00	0	0.00	190	226	2.0
xdm	86	0	0.00	0	0.00	81	92	0.0
nfsd	87	0	0.00	0	0.00	13	19	0.0
xdm	88	0	0.00	0	0.00	82	97	2.0
xdm	89	0	0.00	0	0.00	82	97	2.0
csh	327	0	0.00	0	0.00	43	50	0.0
xdm	94	0	0.00	0	0.00	104	115	2.0
xcterm	310	0	0.00	0	0.00	188	224	2.0
mwm	312	0	0.00	0	0.00	204	226	2.0
go	346	0	0.00	0	0.00	34	43	0.0
csh	316	0	0.00	0	0.00	43	50	0.0
prm	353	100	5.00	83	4.15	54	76	0.0
stnfmgc	354	34	1.70	33	1.65	141	188	2.0
sleep	355	0	0.00	0	0.00	5	13	2.0

PROCESS ID.	CONTEXT SWITCHING VOL.		INVOL.		PROCESS PRIORITY			TOTAL I/O ACT.	SIGNALS RECEIVED
	Total	Avg.	Total	Avg.	High	Low	Avg.		
0	0	0.0	0	0.0	35	35	35	0	0
1	0	0.0	0	0.0	45	45	45	276	53
2	0	0.0	0	0.0	-25	-25	-25	0	0
3	72	0.5	0	0.0	35	35	35	0	0
4	0	0.0	0	0.0	44	44	44	0	0
72	0	0.0	0	0.0	45	45	45	0	1
34	29	0.2	0	0.0	46	44	45	1955	231
71	0	0.0	0	0.0	47	47	47	4	0
73	0	0.0	0	0.0	45	45	45	1	1
50	0	0.0	0	0.0	46	46	46	4	0
54	0	0.0	0	0.0	45	45	45	19	4
69	0	0.0	0	0.0	40	40	40	30	3
70	747	5.0	17	0.1	47	40	42	77	0
74	0	0.0	0	0.0	45	45	45	5	0
75	11	0.1	0	0.0	45	44	44	16	1
76	0	0.0	0	0.0	45	45	45	306	38
77	0	0.0	0	0.0	45	45	45	4	0
78	0	0.0	0	0.0	45	45	45	3	0
79	0	0.0	0	0.0	45	45	45	5	0
80	0	0.0	0	0.0	45	45	45	10	0
302	0	0.0	0	0.0	51	51	51	12	0
82	0	0.0	0	0.0	44	44	44	5	1
83	443	3.0	0	0.0	45	45	45	15	0
84	0	0.0	0	0.0	44	44	44	6	0
326	2	0.0	10	0.1	45	44	44	3	0
86	0	0.0	0	0.0	48	48	48	12	0
87	0	0.0	0	0.0	45	45	45	16	0
88	0	0.0	0	0.0	46	46	46	1	0
89	0	0.0	0	0.0	46	46	46	0	0
327	0	0.0	0	0.0	45	45	45	0	18
94	0	0.0	0	0.0	44	44	44	2	0
310	0	0.0	0	0.0	44	44	44	14	0
312	4	0.0	4	0.0	49	45	45	8	0
346	0	0.0	0	0.0	46	45	45	0	7
316	0	0.0	0	0.0	52	52	52	9	7
353	297	2.0	364	2.4	47	45	45	226	1
354	347	2.9	118	1.0	47	44	45	30	0
355	13	0.5	4	0.1	45	45	45	0	0

```
*********************************************
***** Process Resource Monitor Phase 2 *****
*****        Analysis Report            *****
*********************************************
```

LogTime from: Fri Aug 17 10:17:35 1990
to: Fri Aug 17 10:20:04 1990
Host Id: aries
CPU type: 68030
number: 2

COMMAND	PROCESS ID.	80% CPU UASAGE		50% CPU UASAGE		MEMORY COMSUMPTION		
		Milli Sec.	% of Total	Milli Sec.	% of Total	Phys	Virt	Shr
SWAPPER	0	0	0.00	0	0.00	0	0	0.0
init	1	0	0.00	0	0.00	15	20	0.0
PAGEDAEMON	2	0	0.00	0	0.00	8	2056	0.0
lwp1	3	0	0.00	0	0.00	8	8	0.0
lwp2	4	0	0.00	0	0.00	8	8	0.0
getty	72	0	0.00	0	0.00	12	18	0.0
update	34	0	0.00	0	0.00	3	8	0.0
csh	71	0	0.00	0	0.00	41	50	0.0
getty	73	0	0.00	0	0.00	12	18	0.0
errdemon	50	0	0.00	0	0.00	10	16	0.0
cron	54	0	0.00	0	0.00	31	36	0.0
Xgcm	69	0	0.00	0	0.00	171	198	0.0
Xgcm	70	133	6.65	117	5.85	1955	3286	0.0
portmap	74	0	0.00	0	0.00	22	29	0.0
netd	75	0	0.00	0	0.00	24	31	0.0
rwhod	76	0	0.00	0	0.00	24	29	0.0
talkd	77	0	0.00	0	0.00	25	32	0.0
rwalld	78	0	0.00	0	0.00	16	22	0.0
rusersd	79	0	0.00	0	0.00	22	29	0.0
timed	80	0	0.00	0	0.00	31	39	0.0
xdm	91	0	0.00	0	0.00	104	115	2.0
ypbind	82	0	0.00	0	0.00	18	24	0.0
biod	83	16	0.80	0	0.00	11	11	0.0
mountd	84	0	0.00	0	0.00	25	31	0.0
csh	372	0	0.00	0	0.00	43	50	0.0
xdm	86	0	0.00	0	0.00	81	92	0.0
nfsd	87	0	0.00	0	0.00	13	19	0.0
xdm	88	0	0.00	0	0.00	82	97	2.0
xdm	89	0	0.00	0	0.00	82	97	2.0
xcterm	371	0	0.00	0	0.00	193	227	2.0
csh	361	0	0.00	0	0.00	43	50	0.0
xdm	254	0	0.00	0	0.00	105	116	0.0
xcterm	355	0	0.00	0	0.00	190	224	2.0
stnofmt	504	49	2.45	33	1.65	143	188	2.0
mwm	357	0	0.00	0	0.00	209	231	2.0
csh	405	0	0.00	0	0.00	43	50	2.0
xcterm	404	0	0.00	0	0.00	194	228	2.0
csh	484	0	0.00	0	0.00	43	50	2.0
prm	503	100	5.00	84	4.20	54	76	0.0
xcterm	483	0	0.00	0	0.00	190	224	2.0

PROCESS ID.	CONTEXT SWITCHING VOL.		INVOL.		PROCESS PRIORITY			TOTAL I/O ACT.	SIGNALS RECEIVED
	Total	Avg.	Total	Avg.	High	Low	Avg.		
0	0	0.0	0	0.0	35	35	35	0	0
1	0	0.0	0	0.0	45	45	45	305	281
2	0	0.0	0	0.0	-25	-25	-25	0	0
3	73	0.5	0	0.0	35	35	35	0	0
4	0	0.0	0	0.0	44	44	44	0	0
72	0	0.0	0	0.0	45	45	45	0	1
34	32	0.2	2	0.0	45	45	45	7189	2185
71	0	0.0	0	0.0	45	45	45	19	17
73	0	0.0	0	0.0	45	45	45	1	1
50	0	0.0	0	0.0	46	46	46	4	0
54	0	0.0	0	0.0	45	45	45	72	37
69	0	0.0	0	0.0	40	40	40	51	36
70	748	5.0	6	0.0	43	40	42	96	30
74	0	0.0	0	0.0	45	45	45	6	0
75	0	0.0	0	0.0	45	45	45	15	1
76	0	0.0	0	0.0	45	45	45	1915	364
77	0	0.0	0	0.0	46	46	46	4	0
78	0	0.0	0	0.0	45	45	45	3	0
79	0	0.0	0	0.0	46	46	46	5	0
80	13	0.1	0	0.0	45	44	44	18	0
91	0	0.0	0	0.0	49	49	49	4	0
82	0	0.0	0	0.0	45	45	45	7	3
83	438	2.9	0	0.0	46	45	45	5	0
84	0	0.0	0	0.0	45	45	45	24	0
372	0	0.0	0	0.0	45	45	45	0	6
86	0	0.0	0	0.0	47	47	47	11	0
87	5	0.0	0	0.0	45	45	45	380	0
88	0	0.0	0	0.0	46	46	46	1	0
89	0	0.0	0	0.0	46	46	46	0	0
371	0	0.0	0	0.0	44	44	44	0	0
361	0	0.0	0	0.0	49	49	49	3	7
254	0	0.0	0	0.0	48	48	48	10	0
355	0	0.0	0	0.0	44	44	44	6	0
504	490	4.7	261	2.5	46	44	45	37	0
357	3	0.0	19	0.1	46	44	45	10	0
405	5	0.0	0	0.0	46	45	45	1	42
404	6	0.0	16	0.1	45	44	44	3	0
484	0	0.0	0	0.0	45	45	45	0	5
503	306	2.1	320	2.1	47	45	45	240	1
483	0	0.0	0	0.0	45	45	45	0	0

```
*********************************************
***** Process Resource Monitor Phase 2 *****
*****        Analysis Report            *****
*********************************************
```

LogTime from: Thu Aug 16 15:28:07 1990
to: Thu Aug 16 15:30:38 1990
Host Id: aries
CPU type: 68030
number: 2

COMMAND	PROCESS ID.	80% CPU UASAGE		50% CPU UASAGE		MEMORY COMSUMPTION		
		Milli Sec.	% of Total	Milli Sec.	% of Total	Phys	Virt	Shr
SWAPPER	0	0	0.00	0	0.00	0	0	0.0
init	1	0	0.00	0	0.00	15	20	0.0
PAGEDAEMON	2	0	0.00	0	0.00	8	2056	0.0
lwp1	3	0	0.00	0	0.00	8	8	0.0
lwp2	4	0	0.00	0	0.00	8	8	0.0
getty	72	0	0.00	0	0.00	12	18	0.0
update	34	0	0.00	0	0.00	3	8	0.0
getty	71	0	0.00	0	0.00	12	18	0.0
getty	73	0	0.00	0	0.00	12	18	0.0
errdemon	50	0	0.00	0	0.00	10	16	0.0
cron	54	0	0.00	0	0.00	31	36	0.0
Xgcm	69	0	0.00	0	0.00	170	196	0.0
Xgcm	70	133	6.65	116	5.80	412	559	0.0
portmap	74	0	0.00	0	0.00	17	23	0.0
netd	75	0	0.00	0	0.00	25	31	0.0
rwhod	76	0	0.00	0	0.00	24	29	0.0
talkd	77	0	0.00	0	0.00	25	32	0.0
rwalld	78	0	0.00	0	0.00	16	22	0.0
rusersd	79	0	0.00	0	0.00	22	29	0.0
timed	80	0	0.00	0	0.00	31	39	0.0
xdm	302	0	0.00	0	0.00	105	116	0.0
ypbind	82	0	0.00	0	0.00	18	24	0.0
biod	83	17	0.85	0	0.00	11	11	0.0
mountd	84	0	0.00	0	0.00	24	31	0.0
xcterm	326	0	0.00	0	0.00	190	226	2.0
xdm	86	0	0.00	0	0.00	81	92	0.0
nfsd	87	0	0.00	0	0.00	13	19	0.0
xdm	88	0	0.00	0	0.00	82	97	2.0
xdm	89	0	0.00	0	0.00	82	97	2.0
csh	327	0	0.00	0	0.00	43	50	0.0
xdm	94	0	0.00	0	0.00	104	115	2.0
xcterm	310	0	0.00	0	0.00	188	224	2.0
mwm	312	0	0.00	0	0.00	204	226	2.0
go	346	0	0.00	0	0.00	34	43	0.0
csh	316	0	0.00	0	0.00	43	50	0.0
prm	347	84	4.20	83	4.15	68	76	0.0
stnofmt	348	34	1.70	33	1.65	143	188	2.0
sleep	349	0	0.00	0	0.00	5	13	2.0

PROCESS ID.	CONTEXT SWITCHING VOL.		INVOL.		PROCESS PRIORITY			TOTAL I/O ACT.	SIGNALS RECEIVED
	Total	Avg.	Total	Avg.	High	Low	Avg.		
0	0	0.0	0	0.0	35	35	35	0	0
1	3	0.0	0	0.0	45	45	45	274	52
2	0	0.0	0	0.0	-25	-25	-25	0	0
3	73	0.5	0	0.0	35	35	35	0	0
4	0	0.0	0	0.0	44	44	44	0	0
72	0	0.0	0	0.0	45	45	45	0	1
34	0	0.0	0	0.0	46	45	45	1890	220
71	0	0.0	0	0.0	47	47	47	4	0
73	0	0.0	0	0.0	45	45	45	1	1
50	0	0.0	0	0.0	46	46	46	4	0
54	0	0.0	0	0.0	45	45	45	19	4
69	0	0.0	0	0.0	40	40	40	30	3
70	752	5.0	2	0.0	46	40	42	75	0
74	0	0.0	0	0.0	45	45	45	5	0
75	0	0.0	0	0.0	45	45	45	14	1
76	0	0.0	0	0.0	45	45	45	293	36
77	0	0.0	0	0.0	45	45	45	4	0
78	0	0.0	0	0.0	45	45	45	3	0
79	0	0.0	0	0.0	45	45	45	5	0
80	15	0.1	0	0.0	45	45	45	10	0
302	0	0.0	0	0.0	51	51	51	12	0
82	0	0.0	0	0.0	44	44	44	5	1
83	444	3.0	0	0.0	45	45	45	15	0
84	0	0.0	0	0.0	44	44	44	6	0
326	2	0.0	8	0.1	45	44	44	3	0
86	0	0.0	0	0.0	48	48	48	12	0
87	0	0.0	0	0.0	45	45	45	16	0
88	0	0.0	0	0.0	46	46	46	1	0
89	0	0.0	0	0.0	46	46	46	0	0
327	0	0.0	0	0.0	45	45	45	0	18
94	0	0.0	0	0.0	44	44	44	2	0
310	0	0.0	0	0.0	44	44	44	14	0
312	4	0.0	20	0.1	49	45	45	8	0
346	0	0.0	0	0.0	46	45	45	0	1
316	0	0.0	0	0.0	52	52	52	9	7
347	293	2.0	307	2.1	47	45	45	258	1
348	357	3.0	103	0.9	46	44	45	32	0
349	18	0.6	4	0.1	45	45	45	3	0

```
*********************************************
***** Process Resource Monitor Phase 2 *****
*****       Analysis Report             *****
*********************************************
```

LogTime from: Thu Aug 16 16:06:39 1990
to: Thu Aug 16 16:09:08 1990
Host Id: aries
CPU type: 68030
number: 2

COMMAND	PROCESS ID.	80% CPU UASAGE		50% CPU UASAGE		MEMORY COMSUMPTION		
		Milli Sec.	% of Total	Milli Sec.	% of Total	Phys	Virt	Shr
SWAPPER	0	0	0.00	0	0.00	0	0	0.0
init	1	0	0.00	0	0.00	15	20	0.0
PAGEDAEMON	2	0	0.00	0	0.00	8	2056	0.0
lwp1	3	0	0.00	0	0.00	8	8	0.0
lwp2	4	0	0.00	0	0.00	8	8	0.0
getty	72	0	0.00	0	0.00	12	18	0.0
update	34	0	0.00	0	0.00	3	8	0.0
getty	71	0	0.00	0	0.00	12	18	0.0
getty	73	0	0.00	0	0.00	12	18	0.0
errdemon	50	0	0.00	0	0.00	10	16	0.0
cron	54	0	0.00	0	0.00	30	36	0.0
Xgcm	69	0	0.00	0	0.00	170	196	0.0
Xgcm	70	134	6.70	117	5.85	417	552	0.0
portmap	74	0	0.00	0	0.00	17	23	0.0
netd	75	0	0.00	0	0.00	24	31	0.0
rwhod	76	0	0.00	0	0.00	24	29	0.0
talkd	77	0	0.00	0	0.00	25	32	0.0
rwalld	78	0	0.00	0	0.00	16	22	0.0
rusersd	79	0	0.00	0	0.00	22	29	0.0
timed	80	0	0.00	0	0.00	31	39	0.0
xdm	91	0	0.00	0	0.00	104	115	2.0
ypbind	82	0	0.00	0	0.00	18	24	0.0
biod	83	0	0.00	0	0.00	11	11	0.0
mountd	84	0	0.00	0	0.00	24	31	0.0
csh	122	0	0.00	0	0.00	43	50	0.0
xdm	86	0	0.00	0	0.00	81	92	0.0
nfsd	87	0	0.00	0	0.00	13	19	0.0
xdm	88	0	0.00	0	0.00	82	97	2.0
xdm	89	0	0.00	0	0.00	82	97	2.0
xcterm	121	0	0.00	0	0.00	215	228	2.0
xdm	94	0	0.00	0	0.00	105	116	0.0
mwm	107	0	0.00	0	0.00	215	226	2.0
csh	111	0	0.00	0	0.00	43	50	0.0
go	128	0	0.00	0	0.00	34	43	0.0
stnofmt	130	50	2.50	17	0.85	178	188	2.0
xcterm	105	0	0.00	0	0.00	212	225	2.0
prm	129	100	5.00	83	4.15	68	76	0.0
sleep	132	0	0.00	0	0.00	7	13	2.0

PROCESS ID.	CONTEXT SWITCHING VOL. Total	CONTEXT SWITCHING VOL. Avg.	CONTEXT SWITCHING INVOL. Total	CONTEXT SWITCHING INVOL. Avg.	PROCESS PRIORITY High	PROCESS PRIORITY Low	PROCESS PRIORITY Avg.	TOTAL I/O ACT.	SIGNALS RECEIVED
0	0	0.0	0	0.0	35	35	35	0	0
1	0	0.0	0	0.0	46	46	46	133	11
2	0	0.0	0	0.0	-25	-25	-25	0	0
3	74	0.5	0	0.0	35	35	35	0	0
4	0	0.0	0	0.0	44	44	44	0	0
72	0	0.0	0	0.0	45	45	45	0	1
34	5	0.0	0	0.0	45	45	45	107	8
71	0	0.0	0	0.0	46	46	46	3	0
73	0	0.0	0	0.0	45	45	45	1	1
50	0	0.0	0	0.0	46	46	46	4	0
54	0	0.0	0	0.0	46	46	46	5	0
69	0	0.0	0	0.0	44	44	44	51	0
70	903	6.1	322	2.2	53	40	42	8	0
74	0	0.0	0	0.0	45	45	45	6	0
75	0	0.0	0	0.0	49	49	49	7	1
76	8	0.1	0	0.0	45	45	45	15	1
77	0	0.0	0	0.0	46	46	46	4	0
78	0	0.0	0	0.0	45	45	45	3	0
79	0	0.0	0	0.0	46	46	46	5	0
80	0	0.0	0	0.0	44	44	44	6	0
91	0	0.0	0	0.0	49	49	49	4	0
82	0	0.0	0	0.0	44	44	44	7	1
83	443	3.0	0	0.0	47	45	45	5	0
84	0	0.0	0	0.0	46	46	46	6	0
122	0	0.0	0	0.0	45	45	45	0	5
86	0	0.0	0	0.0	47	47	47	11	0
87	5	0.0	0	0.0	45	45	45	3	0
88	0	0.0	0	0.0	46	46	46	0	0
89	0	0.0	0	0.0	46	46	46	0	0
121	5	0.0	18	0.1	45	44	44	1	0
94	0	0.0	0	0.0	52	52	52	3	0
107	91	0.6	337	2.3	59	45	49	15	0
111	0	0.0	0	0.0	56	56	56	4	7
128	2	0.0	1	0.0	46	45	45	0	1
130	681	5.8	229	1.9	55	45	45	145	0
105	0	0.0	0	0.0	50	50	50	17	0
129	293	2.0	293	2.0	48	45	46	254	1
132	0	0.0	0	0.0	45	45	45	3	0

```
*********************************************
***** Process Resource Monitor Phase 2 *****
*****       Analysis Report             *****
*********************************************

LogTime    from: Thu Aug 16 16:12:08 1990
             to: Thu Aug 16 16:14:38 1990
Host         Id: aries
CPU        type: 68030
         number: 2
```

COMMAND	PROCESS ID.	80% CPU UASAGE		50% CPU UASAGE		MEMORY COMSUMPTION		
		Milli Sec.	% of Total	Milli Sec.	% of Total	Phys	Virt	Shr
SWAPPER	0	0	0.00	0	0.00	0	0	0.0
init	1	0	0.00	0	0.00	15	20	0.0
PAGEDAEMON	2	0	0.00	0	0.00	8	2056	0.0
lwp1	3	0	0.00	0	0.00	8	8	0.0
lwp2	4	0	0.00	0	0.00	8	8	0.0
getty	72	0	0.00	0	0.00	12	18	0.0
update	34	0	0.00	0	0.00	3	8	0.0
getty	71	0	0.00	0	0.00	12	18	0.0
getty	73	0	0.00	0	0.00	12	18	0.0
errdemon	50	0	0.00	0	0.00	10	16	0.0
cron	54	0	0.00	0	0.00	30	36	0.0
Xgcm	69	0	0.00	0	0.00	170	196	0.0
Xgcm	70	100	5.00	84	4.20	689	1067	0.0
portmap	74	0	0.00	0	0.00	17	23	0.0
netd	75	0	0.00	0	0.00	24	31	0.0
rwhod	76	0	0.00	0	0.00	24	29	0.0
talkd	77	0	0.00	0	0.00	25	32	0.0
rwalld	78	0	0.00	0	0.00	16	22	0.0
rusersd	79	0	0.00	0	0.00	22	29	0.0
timed	80	0	0.00	0	0.00	31	39	0.0
xdm	91	0	0.00	0	0.00	104	115	2.0
ypbind	82	0	0.00	0	0.00	18	24	0.0
biod	83	16	0.80	0	0.00	11	11	0.0
mountd	84	0	0.00	0	0.00	24	31	0.0
csh	122	0	0.00	0	0.00	43	50	0.0
xdm	86	0	0.00	0	0.00	81	92	0.0
nfsd	87	0	0.00	0	0.00	13	19	0.0
xdm	88	0	0.00	0	0.00	82	97	2.0
xdm	89	0	0.00	0	0.00	82	97	2.0
xcterm	121	0	0.00	0	0.00	215	228	2.0
xdm	94	0	0.00	0	0.00	105	116	0.0
mwm	107	0	0.00	0	0.00	215	226	2.0
csh	111	0	0.00	0	0.00	43	50	0.0
go	128	0	0.00	0	0.00	34	43	0.0
stnofmt	138	50	2.50	33	1.65	143	188	2.0
xcterm	105	0	0.00	0	0.00	212	225	2.0
prm	137	100	5.00	84	4.20	54	76	0.0
sleep	140	0	0.00	0	0.00	5	13	2.0

PROCESS ID.	CONTEXT SWITCHING VOL. Total	VOL. Avg.	INVOL. Total	INVOL. Avg.	PROCESS PRIORITY High	Low	Avg.	TOTAL I/O ACT.	SIGNALS RECEIVED
0	0	0.0	0	0.0	35	35	35	0	0
1	0	0.0	0	0.0	45	45	45	135	12
2	0	0.0	0	0.0	-25	-25	-25	0	0
3	73	0.5	0	0.0	35	35	35	0	0
4	0	0.0	0	0.0	44	44	44	0	0
72	0	0.0	0	0.0	45	45	45	0	1
34	20	0.1	0	0.0	45	44	44	179	19
71	0	0.0	0	0.0	46	46	46	3	0
73	0	0.0	0	0.0	45	45	45	1	1
50	0	0.0	0	0.0	46	46	46	4	0
54	0	0.0	0	0.0	46	46	46	5	0
69	0	0.0	0	0.0	44	44	44	51	0
70	882	5.9	94	0.6	48	40	41	13	0
74	0	0.0	0	0.0	45	45	45	6	0
75	0	0.0	0	0.0	44	44	44	9	1
76	14	0.1	0	0.0	45	45	45	31	3
77	0	0.0	0	0.0	46	46	46	4	0
78	0	0.0	0	0.0	45	45	45	3	0
79	0	0.0	0	0.0	46	46	46	5	0
80	4	0.0	0	0.0	45	45	45	6	0
91	0	0.0	0	0.0	49	49	49	4	0
82	0	0.0	0	0.0	44	44	44	7	1
83	444	3.0	0	0.0	45	45	45	5	0
84	0	0.0	0	0.0	46	46	46	6	0
122	0	0.0	0	0.0	45	45	45	0	5
86	0	0.0	0	0.0	47	47	47	11	0
87	26	0.2	0	0.0	45	45	45	7	0
88	0	0.0	0	0.0	46	46	46	0	0
89	0	0.0	0	0.0	46	46	46	0	0
121	1	0.0	8	0.1	45	44	44	1	0
94	0	0.0	0	0.0	52	52	52	3	0
107	38	0.3	296	2.0	49	45	48	15	0
111	0	0.0	0	0.0	56	56	56	4	7
128	0	0.0	0	0.0	46	45	45	0	7
138	341	2.8	136	1.1	49	44	45	30	0
105	0	0.0	0	0.0	50	50	50	17	0
137	303	2.0	340	2.3	47	45	45	226	1
140	15	0.6	6	0.2	45	45	45	0	0