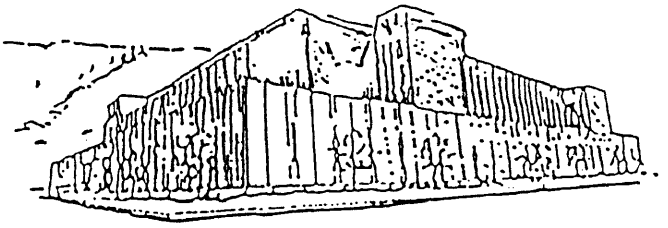
1997

Porting graphical user interfaces from X/Motif to Microsoft Windows

Kirk A. Moeller
The University of Montana



Maureen and Mike
MANSFIELD LIBRARY

The University of **MONTANA**

*** Please check "Yes" or "No" and provide signature ***

Yes, I grant permission P
No, I do not grant permission _____

Author's Signature Kirk D. Moeller

Date 9/29/97

PORTING GRAPHICAL USER INTERFACES FROM X/MOTIF TO MICROSOFT WINDOWS.

by

Kirk A. Moeller

B.S. The University of Montana, 1991

presented in partial fulfillment of the requirements

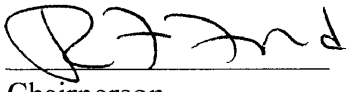
for the degree of

Master of Science

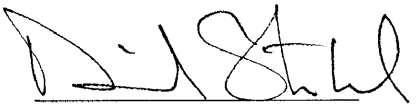
The University of Montana

1997

Approved by:



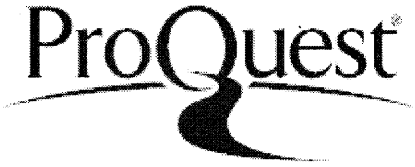
Chairperson



Dean, Graduate School

5-1-97

Date



ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

Moeller, Kirk A., M.S., May 1997 Computer Science

Porting Graphical User Interfaces from X/Motif to Microsoft Windows (239 pp.)

Director: Ray Ford

The primary goal of this project is to examine the process of porting X/Motif based software to run in the Microsoft Windows Environment. The outcome of the project includes a set of general guidelines that a programmer can use as an aid in performing such a port. These guidelines are illustrated in the successful port of a specific, complex X/Motif application.

The primary contribution of this paper is provide insight into the process of porting an application from X/Motif to Microsoft Windows using the class framework called The Object Windows Library. This paper attempts to give the reader a better understanding of the similarities and differences of the two systems. The paper also takes the reader through the process of doing such a port, detailing the problems encountered along the way. In short the goal of this paper is to aid programmers wishing to port programmers from X/Motif to the Object Windows Library, with the hopes of making their jobs a little easier.

Table of Contents.

List of Illustrations.

Chapter 1

INTRODUCTION

The primary goal of this project is to examine the process of porting X/Motif based software to run in the Microsoft Windows Environment. The outcome of the project includes a set of general guidelines that a programmer can use as an aid in performing such a port. These guidelines are illustrated in the successful port of a specific, complex X/Motif application. As discussed below, the portability of any X/Motif system all depends on the X/Motif "source" library functions used in creating the graphical user interface (GUI), and the Windows "target" library available to use in the Windows version. An X/Motif GUI can be created in many ways, so some X/Motif GUI's may be more inherently portable than others. The porting process also depends on the "target" Windows interface, and two specific alternatives are discussed below.

Topics Covered.

Generally, porting a GUI between windowing systems involves aspects of the original windowing system, the new target windowing system, and the application itself. The next chapter presents a brief overview of two applications, the "Ecosystem Information System" and "GIS2DX", for which a Motif-to-MS/Windows port was considered. Chapter Three discusses the Microsoft Windows Application Programmers Interface (API) and compares it to the X/Motif interface. Following this, Chapter Four discusses in detail the lessons learned in attempting to port the Ecosystem Information System to the version 4.0 of the

Microsoft Windows operating system, commonly known as Windows 95 (Win95). Chapter Five discusses an extension to the Microsoft Windows API called OWL, comparing it overall to X/Motif programs. Chapter Six presents the lessons learned in porting GIS2DX to Microsoft Windows using Owl. Finally, Chapter Seven summarizes the two porting experiences, and outlines a general process which can be used to port an X/Motif GUI to Microsoft Windows using Owl.

The primary contribution of this paper is provide insight into the process of porting an application from X/Motif to OWL. This paper will attempt to give the reader a better understanding of the similarities and differences of the two systems. The paper will also take the reader through the process of doing such a port, detailing the problems encountered along the way. So in summary the contribution of this paper is an aid to programmers wishing to port programs from X/Motif to OWL, with the hopes that their jobs will be made easier through this information.

Chapter 2

BACKGROUND

This section gives some background information and other information pertinent to porting. It discusses two specific locally developed application systems, The Ecosystem Information System and GIS2DX. Both systems are medium sized programs written in C/C++ with an X/Motif GUI.

The Ecosystem Information System (EIS).

The Ecosystem Information System(EIS) is a system that allows users to create object oriented hierarchies that describe classes of ecosystem information. These hierarchies are displayed as a tree in a window. The nodes of this tree consist of classes, instances and methods all of which contain ecosystem information. This tree allows users to browse through the hierarchy and display more specific information about a particular class, instance or method. This and the rest of the GUI are fairly straightforward and appeared to be a good candidate for porting to Microsoft Windows.

In the beginning when considering EIS for porting there was too much concentration on the portability of the GUI and too little attention paid to the rest of the code. The initial assumption concerning the rest of the code was that given the portability of the C/C++ language, it would not be difficult to port. However, it turned out that the non-GUI part of the code was not easily portable. The major factor in this was the client/server nature of EIS. EIS is designed as a client/server application. The idea is to have the ecosystem information stored on a server or servers. The information is then retrieved and viewed using the client

part of the application. In order to have the necessary communication between the client application and the server application it is necessary to use Remote Procedure Calls (RPC). RPC is a library of functions that allow a two or more programs to communicate by passing data over a network connection. RPC is standard on Unix systems, making it very portable. However, the Microsoft implementation of RPC is slightly different thereby causing some serious portability problems. These problems as well as a few others, as I will discuss shortly, led to the decision to consider a different application.

The Data Explorer(DX) Converter System, "GIS2DX".

The second application to be considered is the Data Explorer (DX) Converter format system known as "GIS2DX". GIS2DX is a collection of programs operated by a common graphical user interface. Each program (nine total) takes a data file in a particular GIS data format and converts it to a data file in the Data Explorer format. The programs making up GIS2DX are described in the following list:

- DEM Converts USGS Digital Elevation Models.
- DLG Converts Digital Line Graphs.
- MODFLOW Converts a few specific types of Modflow files.
- ERDAS Converts ERDAS LAN and GIS data files.
- ARC/INFO Converts Arc/Info coverages and grids. (Requires Arc/Info software.)
- SHAPEFILE Converts Arcview Shapefiles.
- ORACLE Converts Oracle databases. (Requires Oracle software.)

- DXF Converts DXF format files. (Requires DX Libraries)
- ERDAS/IMG Converts Erdas Imagine format files.

The GIS2DX GUI is made of of several dialog boxes. The main window for the GUI is a dialog box consisting of eleven buttons as well as some descriptive text. Two of these are Quit and Help. The remaining buttons each start up one of the converters described above. All of the dialogs for the converters are very similar in appearance and operation, except for the Oracle converter. These dialogs consist of three main parts: the open file box, the options section, and the save file box. The open and save file boxes do just what their names suggest and are virtually identical in every dialog. The options section is unique to each converter. This section contains a number of controls that allow the user control over the conversion process. Briefly stated, the Oracle converter dialog has a format that is much different and more appropriate for interaction with an Oracle database.

Initially, both EIS and GIS2DX were evaluated for feasibility of porting. GIS2DX was eventually chosen as a candidate for a Windows 95 port for several reasons, including the portability and structure of the non-GUI code. One aspect of the code design is the distinct nature of the nine programs making up GIS2DX. At one time the programs in GIS2DX were all separate programs. Later the separate programs were combined, and more programs were added to create the current GIS2DX. As a result, the parts of GIS2DX are still very modular. In addition, most of the converter modules are self contained, requiring no interaction with other modules or libraries specific to Unix. Six of the converters are distinct units of code, which can be easily removed without affecting any of the other code.

The three other parts involve links to proprietary third party Unix software (Arc/Info, DXF, and Oracle, respectively). These parts are simply omitted in the Win95 version, because the required third party libraries aren't available for Win95.

In more detail, the reasons for the omission of the three parts are as follows:

(1) The Arc/Info converter is designed to be used with a licensed copy of Arc/Info on the same machine. Arc/Info software is now available for Windows NT. However, it was not available for use when this porting was done, and the runtime libraries required to build the GIS2DX-to-Arc/Info link are still not available under Windows NT. It was also not really necessary to have such a program in the Windows 95 version of GIS2DX since the Shape converter can be used instead of the Arc/Info converter.

(2) The DXF converter uses DX library functions. Since DX is only available on Unix machines, it is not possible to port this program without extensive modification.

(3) The Oracle converter requires Oracle database software and runtime libraries to be installed on the same machine. As is the case with Arc/Info, the required libraries are not available on any PC to permit porting the converter code.

Summary.

The two applications described above represent good candidates for a port to Win95 for several reasons. Perhaps one of the main reasons for choosing these programs is that they were locally developed and as a consequence both code and developers were readily available. Yet another reason is the size of the code - -while not small, the size is small enough to be manageable by one person. Finally, the code is written in C/C++ and the Microsoft Windows

API is most easily callable from C/C++.

The primary reason for the selection of GIS2DX over EIS was the modular, portable structure of the GIS2DX code. It was possible to port six of the nine components of GIS2DX, providing real processing behind the new GUI. In contrast, porting the EIS code would have required a major effort to re-write the parsing and interprocess communication parts. Without those, the Windows GUI would have been merely a shell, without any real value.

Chapter 3.

X/Motif vs. Microsoft Windows.

X/Motif is the common designation for the standard windowing environment on Unix workstations, the X-Windows system using window construction techniques from the Motif library. Motif is an enhancement to the X-Windows system which provides a wide range of high level windows and window controls, as well as a rich set of libraries for window creation and control.

The windows and windows controls in X/Motif GUI's are typically called Widgets. The operational paradigm of X is that a user action, such as a button press, generates what is known as an *Event*, which records all the important characteristics of the action. Events are delivered to the X system and saved in a queue. A GUI must sort through these events and determine the appropriate program action. Usually the GUI programmer associates the execution of a particular function with an event, or what is known as a callback. If a callback X has been associated with an event E for a Widget W, then X will be executed when the event queue handler determines that event E has occurred at widget W. A programmer can control X-based GUI's by writing code to the lower level X libraries, i.e. describe events, actions, and callbacks based on X libraries. Alternatively, the programmer can construct an X/Motif GUI by writing simpler event, action, callback code based on the higher level Motif libraries.

Microsoft Windows is a PC based operating system developed by Microsoft Corporation in the mid 1980's. Version 4 of this operating system, commonly known as

Windows 95 (Win95), is the target platform for the X/Motif ports. Win95 comes with a basic Application Programmers Interface (API), consisting of a large library of functions for creation and management of the controls and windows making up user interfaces. The API and the Win95 system, in contrast to X/Motif, uses what are known as Messages. Win95 generates a message in response to a user event such a button press. The message in placed into a queue for the appropriate program. The Win95 system maintains a message queue for each of the programs currently being run. A typical program using the API has a piece of code called the message processing loop. This is a piece of code which loops forever until the program is exited. Within this loop there are two API functions executed. One of the functions retrieves a message from the program message queue. The other function dispatches the retrieved message to the appropriate windows procedure. There is typically a windows procedure associated with each of the windows or dialogs used in an API based program.

At this point it may appear that X/Motif and Win95 are very similar, considering that one uses an event queue and the other a message queue. However the differences arise when you look at what happens next. As stated previously the programmer handles X events using callbacks. The trouble is there is no such equivalent programming tool in the API. This forces the programmer to write very large switch statements to handle the messages that are dispatched to the windows procedure. As one can imagine this leads to some serious difficulties when trying to port a GUI that uses events and callbacks. I discuss these difficulties later, along with my decision to switch to OWL as the Win95 programming interface.

Creating Dialog boxes.

Typically a X/Motif dialog is created using a set of library calls that create controls that make up a dialog box. This set of function calls is generally generated using a GUI builder such as Teleuse, but can be written by hand.

The Win95 approach is slightly different. In Win95 and other versions of Windows, resource script files are used in the building of a GUI. A resource script file contains descriptions of a dialog box, menu or other GUI component. This is written in a very small language which is sufficient enough to describe GUI components. These script files are typically generated by a GUI builder such as Borland's Resource Workshop, but can be written by hand. Each of the GUI components, such as a dialog box, has a unique identifier. This identifier is used as an argument to API functions which create windows or dialog boxes. The difference between X/Motif and Win95 in creating dialog boxes is that X/Motif uses one set of library calls, while Win95 uses a few different library calls in conjunction with the resource script files. What this "small" difference means to the programmer wishing to port from X/Motif to Win95 is that the programmer will not be able to use any of the GUI building code of the X/Motif application. The programmer will instead have to use a GUI builder to Win95 and build the GUI from scratch using the visual appearance of the X/Motif version as a guide.

The Object Windows Library(OWL).

OWL is a set of classes developed by Borland International for use with its C++ compiler. These classes allow the programmer to write Windows programs without using the API functions. The classes also have a number of advantages that make it more attractive in

terms of porting from X/Motif to Windows.

One of the best features of OWL is its use of events. OWL uses events in response to user actions instead of messages like the API. This makes it very attractive since X/Motif also uses events.

Another attractive feature of OWL is its object-oriented design. OWL is very well organized into a hierarchical set of classes which provide virtually all of the functionality needed to implement Windows programs. This object-oriented design is attractive for a couple of reasons. One of the reasons is that X/Motif, though not written using C++ classes, is organized in a manner similar to a hierarchy of classes. This is attractive since this makes it a lot easier to compare OWL and X/Motif than it is to compare the API and X/Motif. Another reason the object-oriented design of OWL is attractive is ease of understanding. The class structure of OWL is very well designed making it much easier to learn and understand than the API.

The following Chapter discusses the porting of EIS to Windows 95. It discusses both my experiences using the API and those using OWL. The chapter also details the problems which led to the decision to choose another program to port to Windows 95.

Chapter 4.

EIS PORTING EXPERIENCES

As I have already stated, the project began this the goal of porting the EIS application from its current X/Motif platform to the Microsoft Windows platform. I started by studying the code for the EIS GUI.

The first step in porting this software was to gain as much understanding of the interface as I could. This was a very difficult task for several reasons. One of these reasons was my knowledge of X/Motif programming was no where near as good as I had thought. Consequently I had to learn quite a bit more about X/Motif programs to understand the code better. Understanding was further hindered by two major factors: very poor and in some cases non existent documentation, and poorly organized code. Poor documentation combined with little or no organization made understanding the code much more difficult than it should have been.

The next step in porting the code was to learn as much as possible about writing Microsoft Windows programs using the API. In trying to write simple programs in the API I immediately discovered problems related to resource script files used for dialogs. Because the underlying X/Motif code for dialog creation has no relation to the Windows resource scripts the code can't be converted - - the dialogs boxes and controls have to be converted visually. Specifically the best way to convert the interface is to get a hard copy of the X/Motif dialog box, then use this "look" as a guide in creating the dialog using a tool such as Resource Workshop.

Porting EIS to OWL.

As discussed in the last chapter OWL has some major advantages over the API which led to the decision to use OWL instead of the API. After reaching this decision the first task was to obtain a hard copy documentation of the X/Motif dialog boxes, showing what "look" the Windows version had to match. This was done by running the application, then using the program "xv" to capture an image of the X/Motif displays. The next step was to run Borland's GUI builder Resource Workshop to try to duplicate the look of the X/Motif dialogs as closely as possible. Once this was complete, the next step is to implement the callbacks to try to duplicate the X/Motif behavior in the Windows environment. Upon attempting this I ran into some major problems, which are discussed next.

RPC and Security Code problems.

The EIS application is designed as a client/server application which uses Remote Procedure Calls (RPC) to allow the client and server to communicate with each other. The fact that EIS uses RPC is a problem in porting to Microsoft Windows. Microsoft Windows has an RPC library, but this library is a little bit different from the Unix versions. Getting a server running on Unix to communicate correctly with a client running on Microsoft Windows is often difficult. Sorting out RPC inconsistencies is outside the goals of my project, and would increase its complexity dramatically.

The security code to authenticate users was an even bigger problem than the RPC code. The security code is scattered throughout all the EIS code, and is based on a Unix concept of users and groups which has no analog in Windows. Removing this code would require restructuring and rewriting a lot of code, which is not good since the goal was to port

the software with minimal changes to the code.

These problems led me to decide to terminate the EIS porting project, with some immediate conclusions about the X/Motif to Windows porting process.

- Porting the GUI involves mimicking the look/feel of the GUI in the Windows environment.
- Code with major behaviors based on libraries unique to Unix can't be easily ported.
- Code with major behaviors based on RPC and Unix users/groups can't be ported.

However, an application that doesn't have these characteristics should be portable, using a system such as OWL. Details on OWL and an example of such a port are given in the following chapters.

Chapter 5.

Owl Programming.

The Object Windows Library, or OWL, is a class framework developed by Borland International and packaged with their C++ compiler. The OWL framework is organized into a hierarchy shown in detail in **Appendix A**. Several classes make up the framework, which together provides most all of the functionality needed to develop good user interfaces. In this chapter I discuss some of the major classes. I will compare and contrast these classes with the equivalent X/Motif structures.

The TApplication Class.

As described in the Borland documentation, "Derived from TModule, TApplication acts as an object-oriented stand-in for an application module. TApplication and TModule supply the basic behavior required of an application. TApplication member functions create instances of a class, create main windows, and process messages."[9] "An instance of the TModule class serves as the object-oriented interface for an ObjectWindows DLL. TModule member functions provide support for window and memory management, and process errors."[9]

One important thing the TApplication class does is process messages. It processes the Windows messages and generates events which effectively makes the OWL programming event driven. The other important purpose of this class is to provide the InitMainWindow() function.

InitMainWindow is used to create a main window. The main window is a

TFrameWindow object and can have either a regular window or a dialog placed in its client area. If used as a regular window, created menus, status bars, and the like can be added, assuming a dialog is not used as the client. The frame window is set as the main window by calling the SetMainWindow() member function of the TApplication class.

By default the InitMainWindow() function creates a generic main window. A program usually creates a subclass of TApplication, then implements its own InitMainWindow() function. This allows the subclass to define a main window with the desired behavior.

The TFrameWindow Class.

An instance of the TFrameWindow class is generally used as the main window in an OWL program. Instances of this class allow menus, status bars, toolbars, etc. to be added to the window. The frame window also includes a client area maintained as a separate entity to insure that changes to the frame do not affect the client area. The client area can be a dialog box, among other things.

The rough X/Motif equivalent of the TFrameWindow class is the Form widget. Generally speaking an X/Motif main window starts with a form widget, with a number of other widgets created as children of this form to add features such as menus, toolbars, and client areas.

Although the TFrameWindow class allows the addition of things like toolbars, it is also very strict in what types of windows adorments one can and cannot add. It is easy to use, but does not provide as much flexibility as the X/Motif Form. Adding things like a status bar or a message bar is easy in OWL, but attempting to add something not built into the

classes can be extremely difficult.

The TWindow Class.

"TWindow is the base for all window classes, including TFrameWindow, TControl, and TDialog."[9] The TWindow class creates a very generic window primary consisting of just a caption, a window, and nothing else. This class is not generally used to create a window, but primarily serves as a base class for several other classes. Perhaps the most important feature of this class is that it provides several dozen functions and utilities for controlling the look and behavior of windows. The rough X/Motif equivalent would perhaps be a form widget with no children.

The TDialog Class.

The TDialog class is the primary base class used to create dialogs. It provides several functions for manipulating those dialogs. Each dialog in an OWL program typically has its own class with TDialog as the parent. TDialog also has several subclasses that are typically known as common dialogs, to implement things such as file selection, search and replace, printing files, etc. The closest equivalent of this class in X/Motif is an XmDialogShell with a Form widget as its child, or an XmFormDialog.

Given this basic overview of the key OWL classes and their X/Motif counterparts, I provide more detail in the next chapter by discussing an application that has been successfully ported from X/Motif. This discussion also includes several examples of actual code that will serve better to illustrate the key points in matching X/Motif and OWL/Windows structure.

Chapter 6.

Porting the Data Explorer Converters.

What I learned.

The collection of Data Explorer(DX) data format converters in the program GIS2DX combines a set of what were originally separate command line programs. Later a X/Motif GUI was added to collect the programs under a common interface. The programs were modified slightly to interact correctly with the GUI dialogs. The individual converters and later the user interfaces were created through the work of several different programmers over the past several years. I personally had worked on the GIS2DX for the previous year and a half, creating two new converters and extensively modifying the remaining seven converters that make up the DX Converters software to improve the GUI, fix bugs, etc. Thus although I was not the author of much of the system, I was familiar with both the GUI and the interaction between the GUI and the conversion components.

GIS2DX was a perfect choice for an application to port from X/Motif to Windows for a couple reasons. One major reason was that there is very little non-portable code that make up the converters, apart from the GUI. This eliminates the portability problems involved in porting the GUI "behavior" that doomed the attempted to port the EIS software. The second reason was my familiarity with the code. The third reason was the actual demand for a version of GIS2DX to run on Win95 — both users and the GIS2DX sponsor had interest in having this system be as portable as possible.

This chapter discusses in detail the process of converting GIS2DX to Win95 using

OWL. I have carefully compared the X/Motif code with the OWL code and have organized the information into several topic areas. In general most topics will have three pieces of information: problems encountered, differences discovered, and code examples. I begin my discussion with the main window and initial program code (i.e., main()).

The Main Window and Initial Program Code.

This section of the paper will discuss the very first things done in a typical Windows program whether it is in OWL or X/Motif. As any X/Motif programmer knows there are a variety of different ways of writing the initial windows setup code. Consequently the X/Motif part of this discussion will be very general and I will concentrate primarily on the OWL discussion.

In the last chapter I discussed the similarities between X/Motif and OWL concerning main windows and initial program code. Consequently this section has two main parts: a discussion of the typical initial OWL code, and a discussion of the typical X/Motif main() function.

OWL programs.

The code involved in getting a main window displayed in an OWL program generally involves two classes: TApplication and TFrameWindow. The first task is to create a class that derives from TApplication. The second is to create a class that derives from either TFrameWindow or TDecoratedFrame. This class will be needed if one plans to have any callback functions associated with menu items. However, if a dialog is used as the main window, this class will not be necessary because the class associated with the dialog box will handle any events. The reason to have a class derived from TFrameWindow in this case

would be if one needed to do anything special with the frame. Thus, the important class in setup is the class derived from TApplication. This class needs to override the base classes' InitMainWindow() function with one of its own.

It is in the InitMainWindow() function that the important actions occur. The first thing that is generally done in this function is creation of an instance of TFrameWindow, TDecoratedFrame, or a derived class. It is important that the reference to the instance point to an OWL Frame Window class. If it points to a user defined class instead, an error will occur when one tries to use it as the main window.

Following initial setup, two different routes can be taken. One option is to add Menus, status Lines, and the like to the frame. There are member functions in the OWL base classes that allow this to be accomplished. The next step would be to create instances of the classes for each of the main dialogs in the application, avoiding the creation of a new instance of a class every time an action is performed.

The second option is to use a dialog box as the main window. In this case things are done a bit differently. The first step is creation of an instance of the class representing the main window dialog. This is typically a class derived from TDialog, however one can use an instance of TDialog if a derived class is not needed. As with the other option, it is next appropriate to create an instance of each class for the dialogs that will make up the applications GUI. This is generally easiest to do in the main window dialog's constructor. The next step is to pass the pointer to this dialog class instance as the child window parameter to the constructor of the TFrameWindow class.

At this point whether or not a dialog has been used as the main window, a pointer to

a frame window has been initialized. The only other step necessary in InitMainWindow() is to call the SetMainWindow(TFrameWindow*) member function of the TApplication class.

There is only one other function in the initial code that needs to be discussed. The function is call OwlMain and is not a member of any class. This function is identical to main() as far as return type and parameters are concerned. Like main this is where execution in the user code will start. There is typically only one statement in this function: *return <class>.Run()*, where *<class>* is the name of the class derived from TApplication. One can, of course, create an instance of the class first and then call the Run() function. However, this is not necessary. The compiler will create a temporary instance and then call the Run() function. The Run() function is a member function of TApplication and is roughly similar to starting the main loop in an X/Motif program, in that it starts the Windows' event/message loop. For an example of using a dialog as the main window please refer to **Appendix D** and the files **gismain.cpp** and **gismain.h** in the OWL code.

X/Motif programs.

In my experience, most initialization code in an X/Motif program is done in the main() function. In general there are three tasks: initialize the toolkit, create an application context, and open a display. After doing all that, a series of functions can be called to create the dialog boxes that make up the GUI, including building the main window. As an example, refer to **Appendix E** and the file **main.C** in the X/Motif code.

The next section compares the OWL class framework to the X/Motif widget tree, using a combination of text and diagrams.

X/Motif Procedural Programming vs. OWL's class structure.

This section discusses the OWL class framework and its relation to X/Motif widget tree. Several diagrams in **Appendix A** show most of the class framework that makes up OWL. There are also several diagrams depicting the class like structure of X/Motif, as well as the equivalent OWL classes, if any.

The set of widgets making up the X/Motif library are organized in a hierarchical manner. The library was created using the C language and as a result does not use the *class* data type of C++. However, in spite of this the X/Motif widget tree is very much like a class hierarchy. As a result, the X/Motif widget tree is very comparable to OWL's class structure. In fact several of the X/Motif widgets have comparable classes in the OWL hierarchy. In most cases the OWL classes also have member function which correspond to X/Motif library functions which are designed to be used with a particular widget. These similarities are a significant factor in making the porting of X/Motif GUI's to OWL realistic. The similarities are illustrated in a few of the X/Motif widget tree diagrams in **Appendix A** where the equivalent OWL class, if any, is shown in parenthesis.

Event Mapping.

X/Motif.

This section discusses event mapping in both X/Motif and OWL. Event mapping refers to the process of connecting an event, such as a button press, to a callback function. The discussion starts with an explanation of how this in done in an X/Motif program.

X/Motif is event driven, meaning that an event is generated when the user performs

an action, such as using the mouse to press a button. X/Motif events are mapped to callback functions by using the *XtAddCallback(widget, event, callback, parameter)* function. The last parameter to this function is typically an array of widgets, which allow access to the widgets that make up a dialog box. Here is an example of this function being used:

```
XtAddCallback(w[WI_DEMPB], XmNactivateCallback,
        (XtCallbackProc) PopupDialogDEM, w);
```

The first parameter identifies the widget from which the event will be generated. The second parameter is the name of the event to be mapped. The third parameter is the name of the callback function. Finally, the last parameter is a widget array passed into the callback function as a parameter. For a more detailed look at event mapping look at the code in **Appendix E**.

OWL.

Microsoft Windows uses messages instead of the events which drive X/Motif. A certain type of message is sent whenever a user performs an action such as pressing a button with the mouse. Fortunately, OWL's base class hides the message system and simulates an event driven system, making it much easier to port X/Motif code to OWL.

OWL events are mapped using a pair of macros called DECLARE_RESPONSE_TABLE and DEFINE_RESPONSE_TABLE. The DECLARE macro does just what the name implies; it declares the response table. It is just a simple one line statement put in the class definition for the dialog box. The DEFINE macro is used to define event mappings. One important fact to note about OWL is that the callback functions

cannot have parameters. However, parameters really aren't necessary, since the callback functions can access the private data members of the dialog class. The callback functions are class members. Here is an example of a response table definition:

```
DEFINE_RESPONSE_TABLE1(TGis2dxDialog,TDialog)
  EV_CHILD_NOTIFY( IDC_DEMPB, BN_CLICKED, dem2dx),
END_RESPONSE_TABLE;
```

The first parameter here is the name of the dialog class in which this response table is declared. The second is its parent class. The first parameter to EV_CHILD_NOTIFY is the identifier for the control that will generate the event. The second parameter is the event that will be generated, in this case a button press. The last parameter is the name of the callback function to be called in response to the event. There can be as many of these statements in the definition as is necessary. For a more detailed look at these tables refer to the program listings in **Appendix D**.

Dialog Creation and Display.

This section, discusses the process of creation and displaying dialog boxes. The creation of dialogs is another area where X/Motif and OWL differ greatly. These differences, as well as the details of how dialogs are created in each system, are discussed next.

X/Motif

Typically an X/Motif application has a large function for creating dialog boxes. In this function there are several function calls to the X and Motif libraries to create the widgets that make up the dialog. Usually the widgets are stored in an array that is passed into the function. This array is used later by callback functions to access the various controls making

up the dialog.

This function is often created by using a GUI builder such as *Teleuse*. *Teleuse* has a graphical tool called *VIP* that provides an easy way to build a dialog box. Once the dialog box is built, the *VIP* program automatically creates the shell of the code to create the dialog box. This function can also certainly be coded by hand, but it is much more work and can be very frustrating at times.

Displaying a X/Motif dialog is quite simple. All that needs to be done is to call the XtManage() function with the dialogs form widget as a parameter. Generally this is done in some sort of callback function.

Microsoft Windows and OWL.

Microsoft Windows uses a much different method of doing dialog creation. Instead of using a series of library calls to build a dialog, Microsoft Windows uses something called resource script files to define dialog boxes, menus, cursors, and other things. These scripts are written in a fairly simple, special language used to describe dialog boxes and other objects. A Windows programmer should try to learn the syntax, and be able to understand and write these scripts if necessary. However, it is much easier use a GUI builder to create these scripts. Most of the commercial C++ compilers for PCs provide some kind of tool for creating these scripts. One such tool,which comes with the Borland C++ compiler, is called Resource Workshop. Once a resource script is created, the rest of the process of creating and displaying a dialog is done in the program code.

The TDialog class is an OWL class designed for building and manipulating dialog

boxes. Generally one creates a class that inherits from TDialog so that callback functions can be added. TDialog only has a few built-in callbacks, so anything other than very simple dialogs require creation of a new class.

The creation of the dialog box is handled by TDialog's constructor. One of the constructors takes a resource id parameter. This id is the identifier assigned to the dialog box when it was created in Resource Workshop. Once the constructor has been called the dialog exists and is ready to use.

Like an X/Motif dialog, the display of the OWL dialog itself is quite simple. All that is necessary to display the dialog is to either call the *Execute()* or *Create()* members of TDialog. The *Execute()* function is used for modal dialogs and does not return until the dialog is closed. Modeless dialogs, on the other hand, are created by calling the *Create()* function, which returns immediately.

At first upon reading the above one might think that the differences between X/Motif and OWL would make porting very difficult, However, this turns out not to be the case. The porting process for dialogs is really quite simple. The best method is to use an application such as *"XV"* to capture an image of the X/Motif dialog, then generate a copy. With this done, one can simply use a GUI builder on the PC, using the screen captures as a reference, to duplicate the look. There are of course a variety of other ways this "porting process" could be done but this probably the easiest. For an example of X/Motif dialog creation and display, see the source code in **Appendix E**. For an example of creating and displaying dialogs in OWL, refer to the .rc files and the various dialog classes in **Appendix D**.

Common Dialogs.

This section discusses common dialog boxes presented in porting the GUI to OWL. Common dialog boxes are dialog boxes built into the libraries of the windows' system. These dialogs provide interfaces for common tasks, such as a file selection dialog for opening or saving files.

In X/Motif, common dialog boxes are made up of two parts: a dialog shell and a box. Put together, the dialog shell and the box make up the common dialog. For instance, the file selection dialog is made up of a dialog shell and a file selection box. This two-piece nature of a common dialog makes it possible to take the same box and make it a part of several dialogs. This is was done within GIS2DX to simplify the dialog code. That is, two file selection boxes are put together with several other controls to make up a custom dialog, which is then used in several conversion options. To see an illustration of such a dialog see **page 69** in **Appendix C**.

OWLs' implementation of common dialogs is a bit different, presenting a major porting problem. OWL has common dialog boxes, but they are not formed in two parts as X/Motif. There is no concept of a common box that can be made part of another dialog. The only thing the programmer has to work with are standalone dialog boxes. This design is a perfect example of how, in at least some ways, X/Motif can be much more powerful than OWL.

Given this fact, is wasn't possible to use any built-in OWL common dialogs for file selection, because they do not match the existing GIS2DX "look". Consequently, I was forced to create dialog boxes that look the same as X/Motif dialogs, but did not have the

built-in file selection control. This solution creates a class that handles the functionality of the group of controls comprising the file selection box. Essentially, the code implements a file selection box with a list of control identifiers as input to the constructor. For an illustration of one of these dialogs, see page 53 in Appendix B.

Dialog Initialization and Display.

This section discusses dialog initialization and display in both the X/Motif and OWL systems. Initialization refers to registering callbacks, initializing variables, and resetting controls. The discussion starts with two functions used in most X/Motif code to perform these tasks, then discusses the equivalent code in OWL.

X/Motif

The dialog initialization and display in GIS2DX is done in two functions: AddCallbacks() and PopupDialog(). The AddCallbacks function has as input the array of widgets making up the dialog. The function contains a series of calls to the XtAddCallback() function, which basically associates events with callback functions.

The PopupDialog function does as its name implies, popping up the dialog box. This function typically performs three actions. The first is to initialize any variables, if necessary. The second action is to reset any dialog control that needs to be reset. An example of this would be resetting a radio button in a dialog, as required if this is the second time the dialog has been opened and the button changed during its first use. Finally, the third action is to display the dialog box by managing the form widget associated with the dialog.

OWL

In OWL programs, there are two important functions for dialog initialization and display: the dialog class constructor and the SetupWindow() member function. The class constructor is roughly analogous to the combination of the dialog creation function and the AddCallbacks function in X/Motif programs. Basically there are two actions performed in the constructor. The first is to create the dialog itself by calling the parent constructor with the appropriate arguments. The second action is to initialize pointers to controls, using the *new* function to create an instance of the correct class and passing it the identifier of the control. For example a pointer to a button control is created by making an instance of the TButton class, passing it the resource identifier of the button. As discussed earlier, the callbacks are registered using the DEFINE_RESPONSE_TABLE macro.

The SetupWindow function is a user-defined member of the dialog class. This function is roughly analogous to the PopupDialog function in an X/Motif program. Typically many of the same actions are performed as in the PopupDialog function. One important difference to note is that control initialization, such as filling in a ListBox, can only be done in this function. Controls can only be modified when the dialog has been displayed or in this function. Another important thing to note is that OWL dialogs do not retain their control state. This means that some extra initialization may be required in this function. For example, if one wishes the user to be able to close the dialog and open it again with nothing changed, it is necessary to save the dialog state upon close and then restore it in this function. Finally note that execution of this function is a result of the Execute() or Create() members of TDialog being called.

As in X/Motif, an OWL dialog is displayed by a callback function getting called in

response to an event such as a button press. In an X/Motif program this callback is PopupDialog(). In OWL there is also a function that is called. This function contains only one line, which calls the Execute() or Create() members using the appropriate dialog class instance. This call results in the SetupWindow() function getting called, which is similar in nature to PopupDialog.

Typically the display process starts by calling the Execute() or Create() members of TDialog. These functions eventually lead to the SetupWindow() function being called if one is defined in the class. Once the SetupWindow call is complete, the dialog is then displayed on the screen.

ListBoxes.

This section discusses list boxes. From their look it would appear that X/Motif and OWL List Boxes are identical. However there are minor differences that are important to note.

One of the differences is more of an irritation than a portability problem. This problem is in how the two windows' systems decided to index list items. In X/Motif list item indexes start counting at one and the end of the list referenced with an index of zero. However, in Microsoft Windows and OWL list item indexing starts counting at zero, and the end of the list is referenced by an index of negative one. These differences can obviously create a few problems, but they are generally easy to solve.

The other difference is in the data type of list box elements. X/Motif uses a special data type for list boxes elements called Compound Strings, or XmStrings. XmString's are

used in list boxes as well as a variety of other widgets. OWL, on the other hand, simply uses the char* data type for list box items. This makes it very nice for adding items to a list since the char* data can simply be passed as parameter to the function that adds an item to the list. In X/Motif, however, it is necessary to first convert a char* variable to an XmString before it can be added as an Item to a list. The examples below illustrate the X/Motif and OWL differences.

Owl source:

char* illustration:

```
// Place the list of layers in the list box.
for(i = 0; i < num_layers; i++)
{
   layerbox->AddString(img->layer_name[i]);
}
```

Index illustration:

```
item_pos = layerbox->GetSelIndex();
```

X/Motif code:

char* illustration:

```
// Place the list of layers in the list box.
for(i = 0; i < num_layers; i++)
{
  xms = (XmString) NULL;
  xms = XmStringCreateLocalized(img->layer_name[i]);
  XmListAddItemUnselected(wa[WI_LAYERSSL_IMG],xms,0);
  if (xms) XmStringFree(xms);
}
```

Index illustration:

```
item_pos = list->item_position - 1;
```

Closing Dialogs and Applications.

This section discusses differences in closing dialogs and applications in X/Motif and

OWL. In both windows' systems closing a dialog or application is fairly trivial, but the saving the dialog state in OWL is not automatic.

Typically in an X/Motif function each dialog has a function called something like PopdownDialog, with the sole purpose being simply to close the dialog box. This is done by simply calling the XtUnmanage() function with the dialog form widget as an argument. In comparing this to OWL dialogs, it is perhaps more appropriate to think of closing an X/Motif dialog as making it invisible. Typically an X/Motif application is closed by a function called Quit(). This function usually does only two things: close the display opened in the main() function, and call the exit() function.

The functionality to close OWL dialogs is built into the base class TDialog. This TDialog class has a response table entry that ties any button with an identifier of IDCANCEL to the member function CmCancel(). So, in OWL all that is necessary to setup close behavior is to make sure that the button used to close the dialog has the IDCANCEL identifier. When this button is pressed, the CmCancel() function is called and the dialog is closed. It is of course possible to define a special CmCancel() function in the derived class. This is especially important if the dialog needs to save the dialog state when it is closed. It is important to remember to call the parent class' function in order to get the dialog box to close properly.

Quitting a Windows 95 application is very much the same as quitting a dialog. In fact, if the button pressed with the IDCANCEL identifier happens to be a button in the main window it will effectively result in the application being terminated. It is also important to note that the above properties of OWL would also apply to menu items with the identifier IDCANCEL.

Up to this point it appears that the differences between X/Motif and OWL are not very significant. However, the state of controls in an OWL dialog is not saved upon close. This is where imagining an X/Motif dialog being made invisible makes things clearer. An OWL dialog when closed is truly closed and information in the dialog is lost. This is not the case with the X/Motif dialog. The information in the dialog will still be there when the dialog is opened again. To duplicate this behavior in Windows 95, the programmer may want to override the standard CmCancel() function, because this is where the state of the dialog must be saved. The information saved in this function can be restored to the dialog in the SetupWindow() function. It should be mentioned that OWL is supposed to support a special way of performing state saves and restores; however several attempts to use the functionality resulted in failure.

Messages, Cursors, and other Miscellaneous Information.

This section discusses message boxes and mouse cursors. Both OWL and Microsoft Windows have a much easier and convenient way of displaying messages and changing the cursor shape than is available in X/Motif. Microsoft Windows has a function called MessageBox(), which is also available in OWL in a slightly more convenient form. This function's purpose is to display a variety of different message boxes and wait for the user to respond. It can display Error boxes, Help boxes, Message Boxes, Information boxes, and a few others. These boxes can potentially have up to three buttons: Ok, Cancel, and Help. The buttons displayed depend on the parameters given in the MessageBox() function. As an example, here is a call to this function that displays an Error message box with an Ok button:

```
MessageBox("NO OUTPUT FILE SELECTED", 0, MB_OK | MB_ICONERROR);
```

The constant MB_OK says to only include an Ok button in the box, and the constant MB_ICONERROR says to display an icon in the box showing that this is an error message.

Changing the cursor shape in OWL is also quite simple. All that is necessary is to call the SetCursor(Window pointer, resource id) with the appropriate parameters. The first parameter is a pointer to a window. If calling this function within a dialog class this would simply be the *'this'* pointer. The second parameter is a resource identifier for the cursor shape to use. This would typically be created using a tool such as Resource Workshop. A special case of this function is when the first parameter given is zero. In this case, the second parameter used can be selected from several built in cursor identifiers of the form IDC_xxxx. This effectively changes the cursor for all visible windows of the application.

Unfortunately Displaying Message Boxes in X/Motif is not quite as simple. There is no built in function for displaying a variety of different message boxes. Instead it is typically best to create a class that will encapsulate all that is necessary to create the various different message boxes. Another important item to note is that in order to have the application effectively halt and wait for the user to reply, it is necessary to simulate an event loop. Like message boxes, there is no simple function to change the cursor shape in X/Motif. It is best to encapsulate all of the necessary functionality in a class. This way all that is necessary in the application code is to create an instance of the class and call the appropriate member functions.

Finally, coming under the heading of miscellaneous information is opening binary files on a PC. The Unix operating system does not distinguish between binary and ASCII files.

MS-DOS, on the other hand, does distinguish between binary and ASCII. This makes it necessary to be sure to use the *ios::binary* flag anytime a binary file is opened for reading or writing.

Chapter 7.

A Step-by-Step Guide to Porting GUI's from X/Motif to OWL.

This chapter attempts to guide the reader through the process of porting a GUI written for X/Motif to OWL. Each step in this process is described, but some steps are covered briefly if they are covered in previous parts of this paper.

Dialog Appearance.

The first task in porting to OWL is to port the windows themselves, that is to create windows and dialogs on the PC that look identical to their X/Motif counterparts. As noted earlier, one of the best ways to do this is to simply capture an image of the original dialogs, and use this as a guide in the PC development. Alternatively one could set up the Unix workstation and the development PC next to each other. This would allow one to run the program on the Unix workstation while creating the dialogs on the PC. It is important to have a list of the identifier names for the dialog controls handy to make porting the source code easier. After gathering the appropriate information, then the next step is to create the dialogs.

This dialog creation is best done using a GUI building tool such as Borland's Resource Workshop. The image captures can now be used as reference in creating the dialogs. In some cases it might be help in creating the new dialog to know exactly how some of the dialog's controls operate. It is in cases such as these where having a Unix workstation next to the PC is very helpful, particularly if the software has a large number of windows, which makes image capturing unrealistic.

Another initial task is to port the main window. There are two ways that this can be done. The simplest is to make the main window a dialog and create it as described above. Alternatively, a more difficult route is to build the window within the InitMainWindow() function. Be aware however that menus still have to be done in a resource script file. Resource Workshop has tools that will allow one to create menus easily.

Dialog Functionality.

At this point one should have all of the dialogs that are needed in the program defined in a resource script file. The next most logical task is to port the functionality of the dialogs. It of course would be very helpful to know exactly how the dialogs function. One can always look at the source code, but the source code is not always clear.

The functionality of the dialogs is controlled exclusively by a set of functions known as callbacks. The best way to start is to determine exactly what callback functions are associated with the dialog currently being ported. With this information in hand the dialog's class definition can be written.

Dialog Class Definition.

The dialog's class typically inherits from the TDialog class. Most classes will have five components: a constructor, a SetupWindow function, callback functions, other functions, and data members. The constructor calls the parent's constructor, passing it the dialog's resource identifier. The constructor is also where pointers to dialog controls are created. The SetupWindow function, which is optional, is called after the dialog is popped up but before it is visible to the user. This allows various controls to be initialized before they are visible. The member functions that will be callback functions are most likely needed in the OWL class

as well. In some cases the function may not be needed, but it is best to assume it is needed. Any other functions that are needed should be declared now. These include functions that are called by the callbacks associated with the dialog. Finally, the class may have private data members. In most cases global variables or variables passed as callback parameters become private data members in the OWL class. An example is the widget array that is usually passed as a parameter to an X/Motif callback. In the case of the OWL code this translates into several private data members in the class, all of which are pointers to dialog controls. The last thing that needs to be in the class is the declaration of the response table, done using the DECLARE_RESPONSE_TABLE macro. For an example of a class definition for a dialog, refer to **Appendix D**.

Response Table Definition.

The Response table is defined using the DEFINE_RESPONSE_TABLE macro to associate control events with callback functions. The best place to place this table is at the top of the dialog class implementation file. Writing this table goes much more quickly by making sure to have three items available when it is written. First is the code containing the XtAddCallback() function calls, which is invaluable in determining what to include in the table. The second item is a list of the identifiers that go with each control, which are needed when the table is written. If it happens the identifiers are the same as the X/Motif version, then simply refer to the XtAddCallback calls. The final item is a reference guide showing a list of events that can occur to various controls, which makes it much easier to find the appropriate event to use.

The Dialog Class Implementation File.

This file contains implementations of all of the member function of the dialog class as well as the response table definition. The best way to approach this is to collect all of the X/Motif callbacks associated with this dialog into one file. Next surround each and every function with the #if 0 and #endif compiler directives. These directives effectively prevent the enclosed piece of code from being compiled. This allows the code to be compiled and tested without having all of the functions finished. Be sure to remember to modify the declaration of unimplemented functions to make them empty (e.g., Openfile(); becomes Openfile() {}). Tackle each of the callback function one at a time and port them to OWL. An OWL reference guide is an invaluable tool in determining the appropriate class and class member functions to use in place of a X/Motif function or set of functions. However, be aware that there is not always a function or class available in OWL that exactly matches the functionality of a X/Motif function. Consequently some modification of the code may be necessary.

The Main Window functionality.

As with dialogs the main window also has a variety of callback functions. Regardless of whether or not the main window is a dialog box, it can be ported in much the same manner as described above. In the case of a window that is not a dialog box the main windows' class would probably inherit from TDecoratedFrame instead of TDialog. Also try to remember that the process of popping up a dialog window is a callback function that belongs in this main window class. Any initialization of dialog controls and the like is done in the dialog classes SetupWindow() function.

Chapter 8.

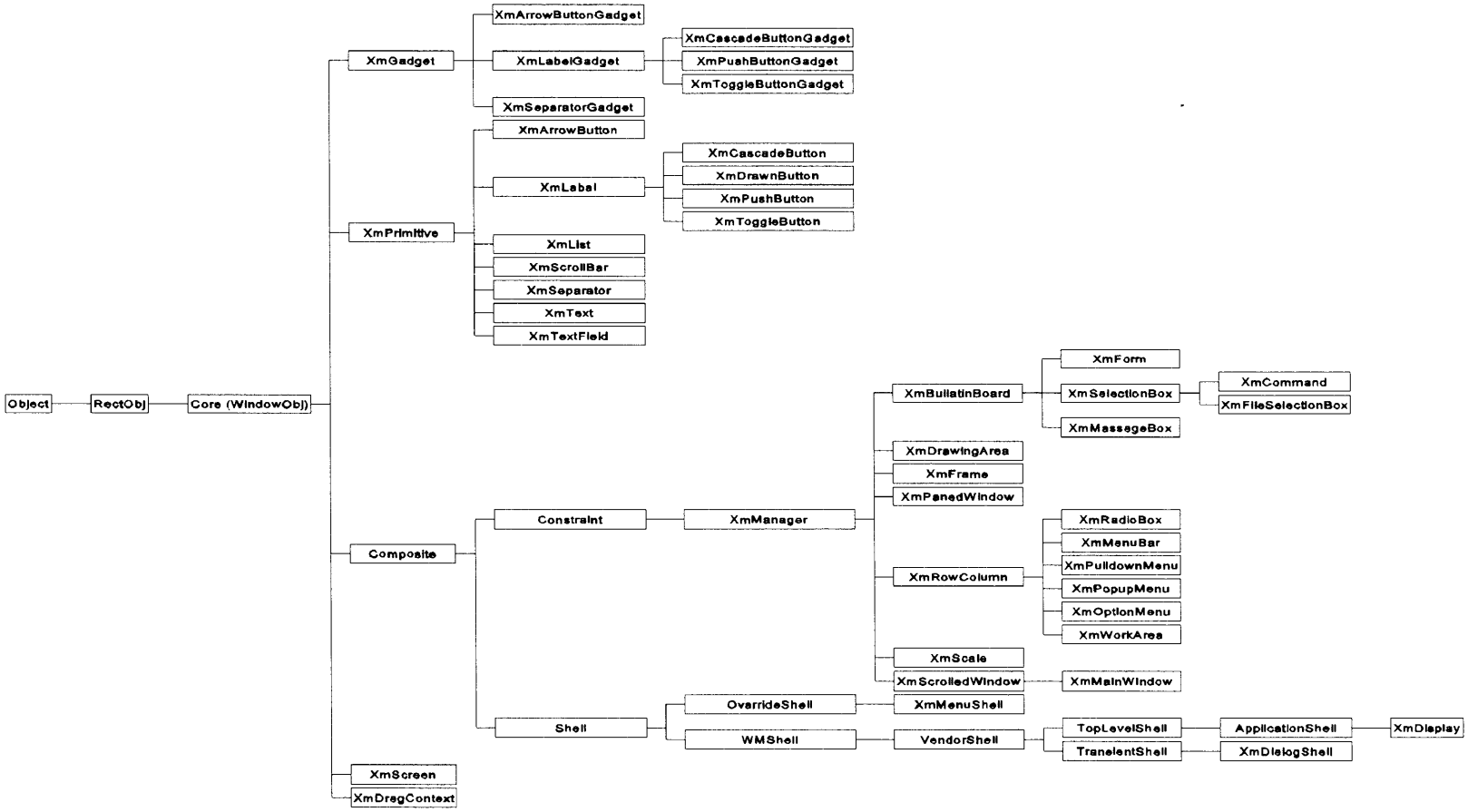
Conclusion.

This concludes my paper on porting X/Motif based GUI's to the Microsoft Windows platform. I have discussed in detail this process with the target being Borland's class framework known as the Object Windows Library. I have of course not covered every aspect of the two windows' systems. However, I believe I have covered them in sufficient detail to be of great benefit to anyone trying to perform such a port. The ideal reader wishing to use this information needs a good working knowledge of programming in X/Motif and OWL. All of the source code relating to the GUI, both the X/Motif and OWL code is in the following appendixes.

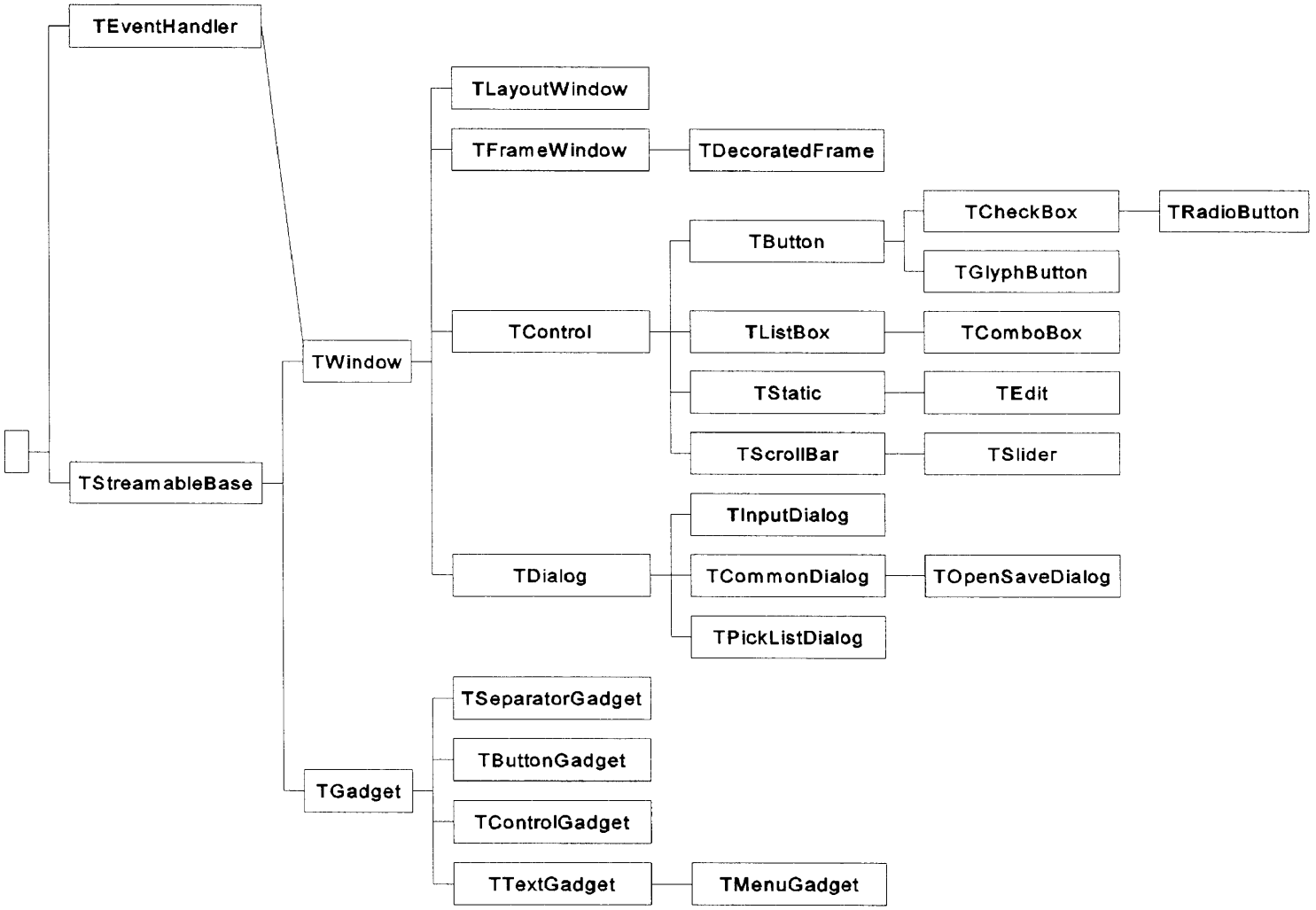
APPENDIX A

OWL AND X/MOTIF CLASS DIAGRAMS

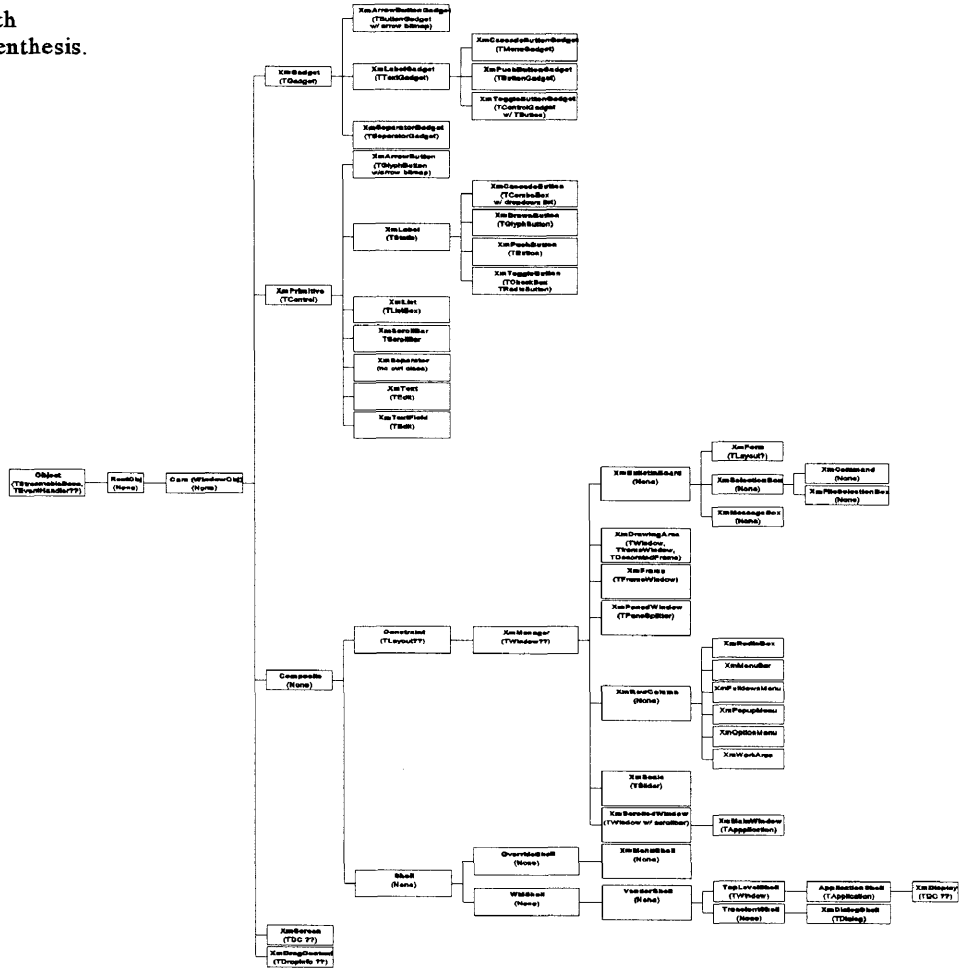
The X/Motif Widget Tree.



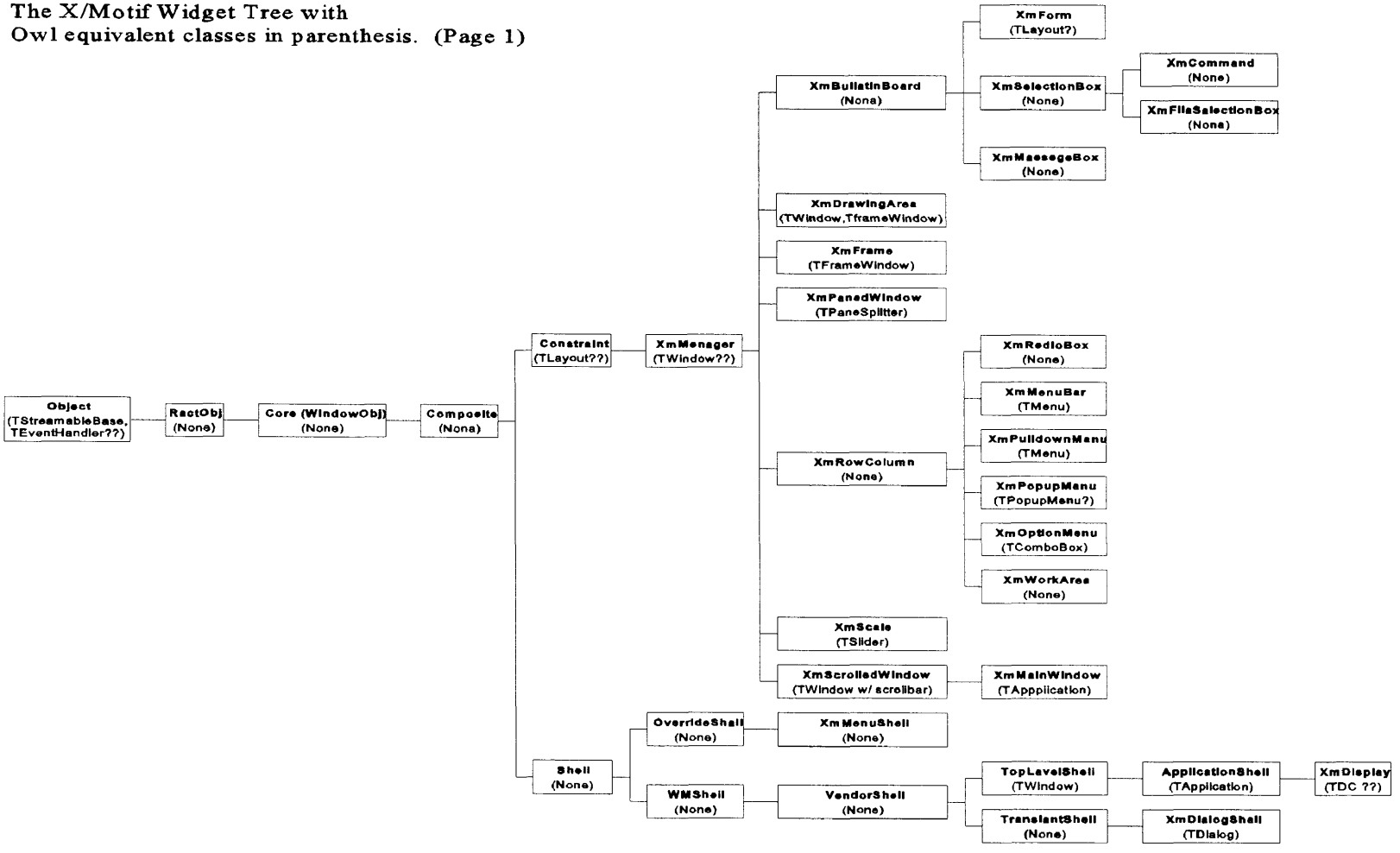
Abbreviated Tree of Owl Classes.



The X/Motif Widget Tree with
Owl equivalent classes in parenthesis.



The X/Motif Widget Tree with
Owl equivalent classes in parenthesis. (Page 1)



The X/Motif Widget Tree with
Owl equivalent classes in parenthesis. (Page 2)

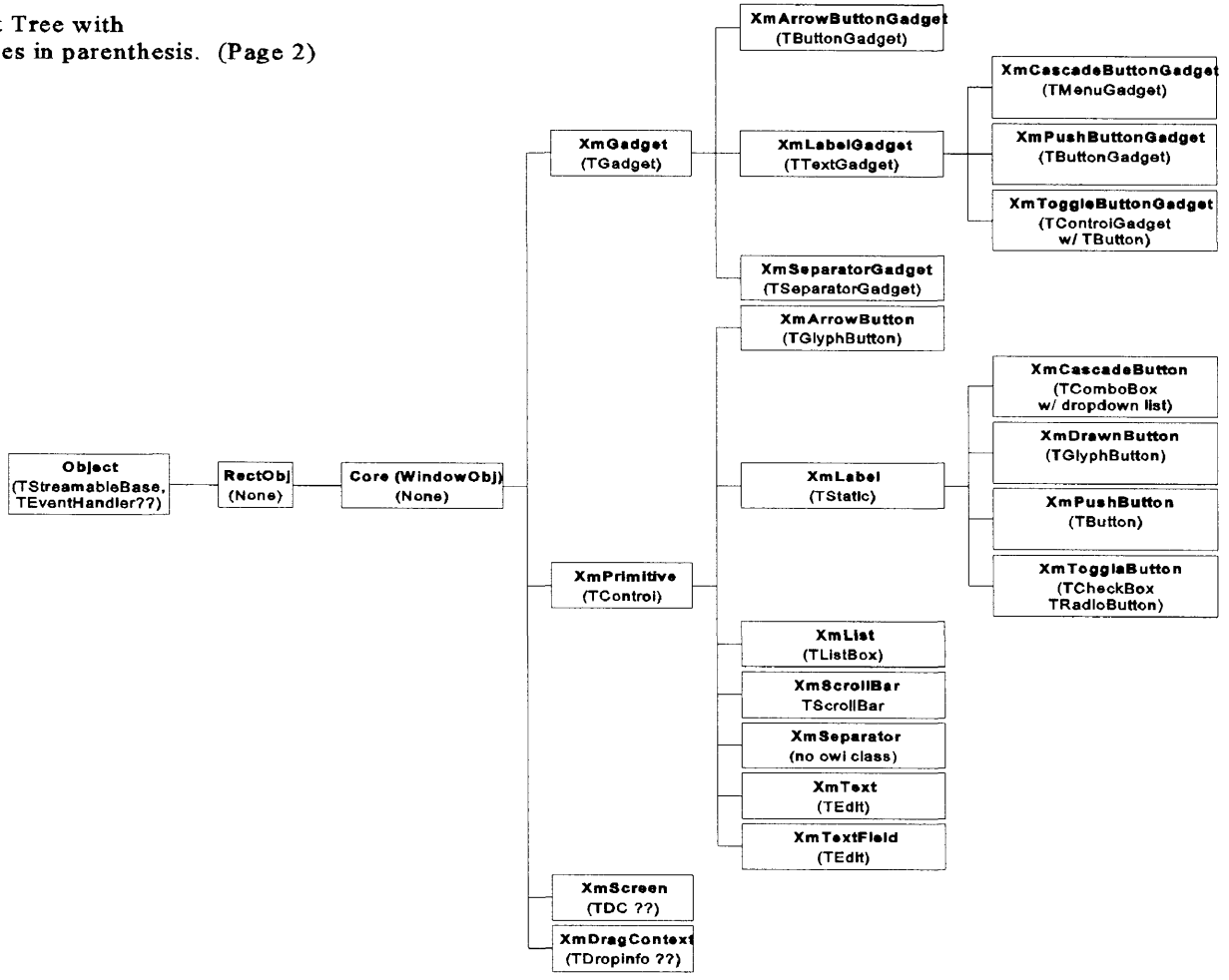
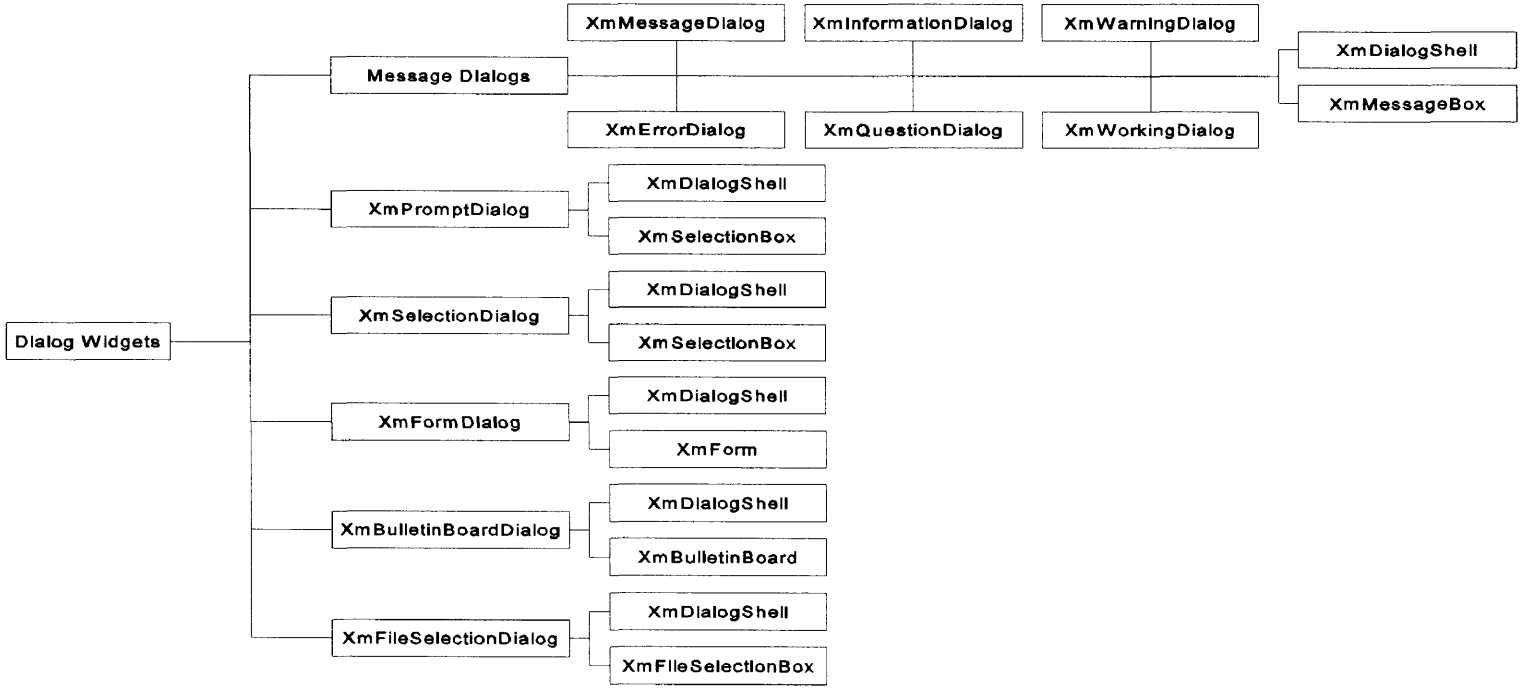
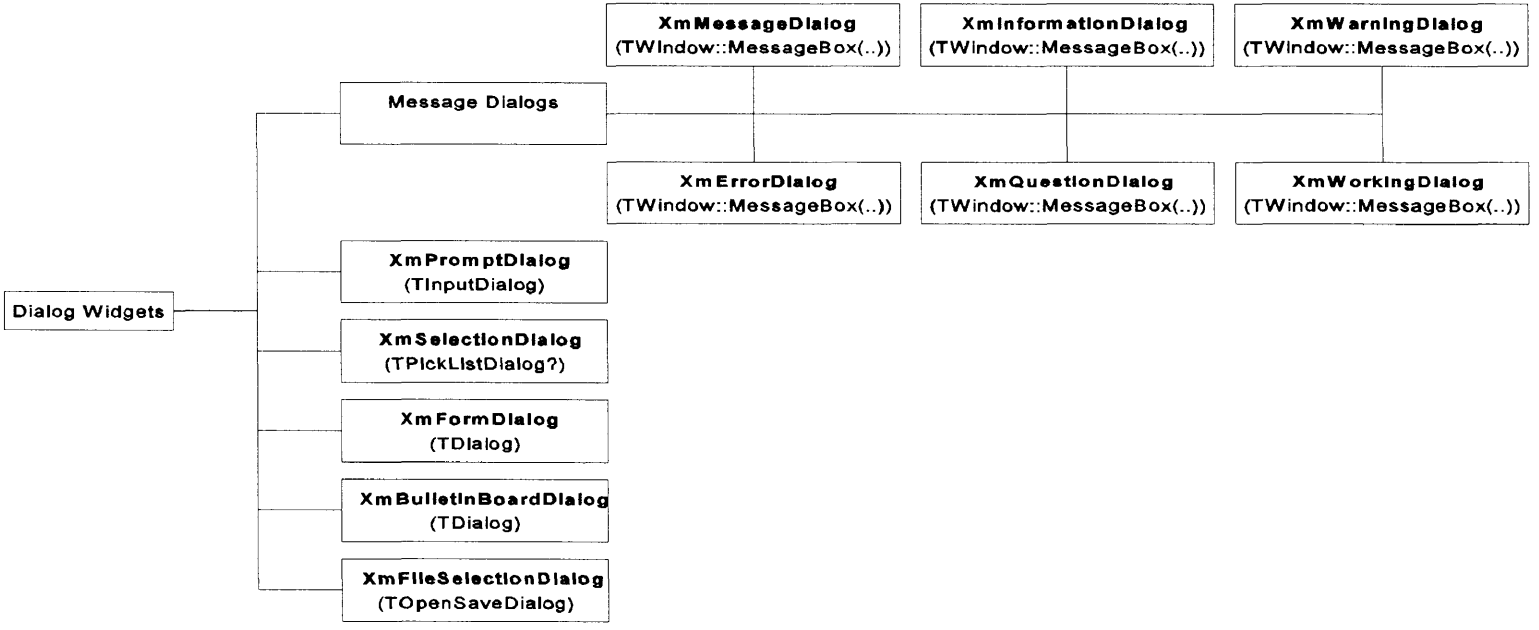


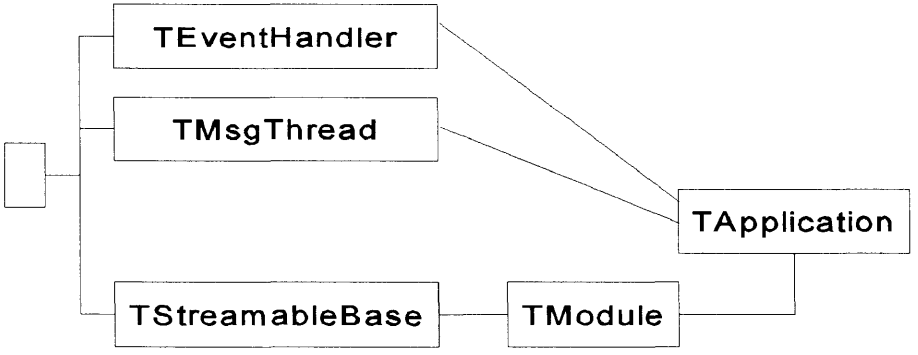
Diagram illustrating the make up of X/Motif's built in dialog boxes.



X/Motif's built in dialog boxes and the Owl equivalents in parenthesis.



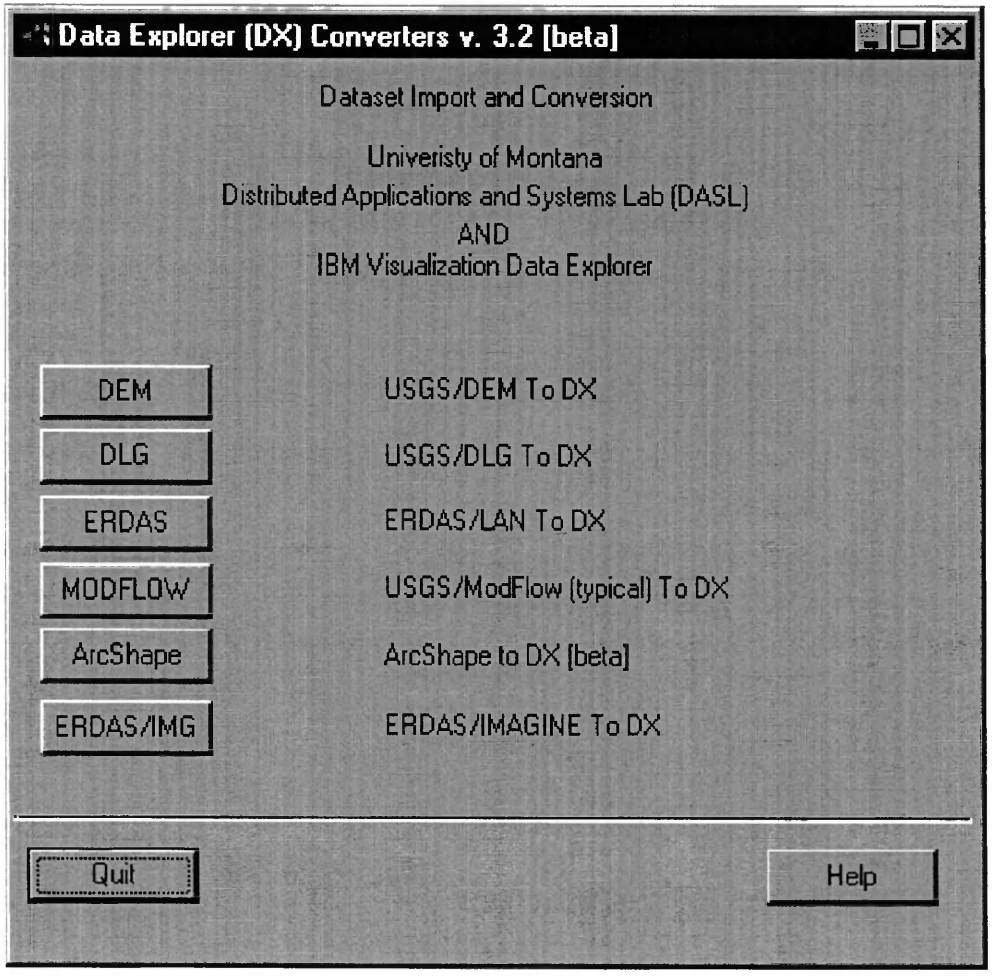
TApplication class (similiar to X/Motif's MainWindow and ApplicationShell).



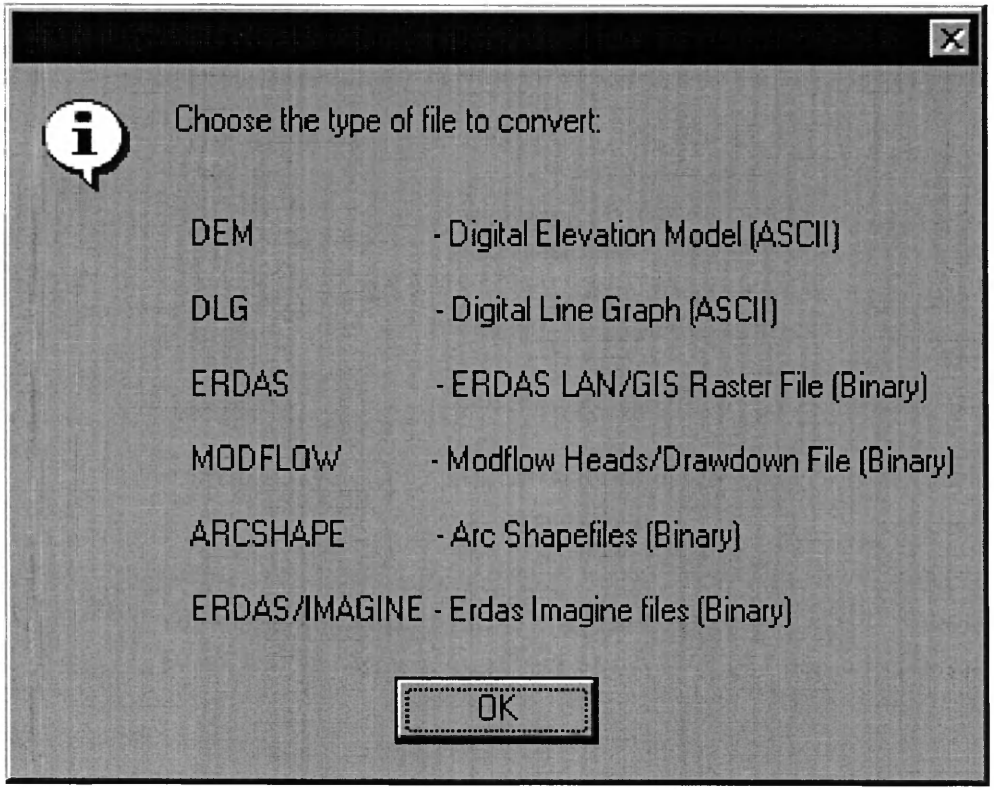
APPENDIX B

GIS2DX SCREEN SHOTS

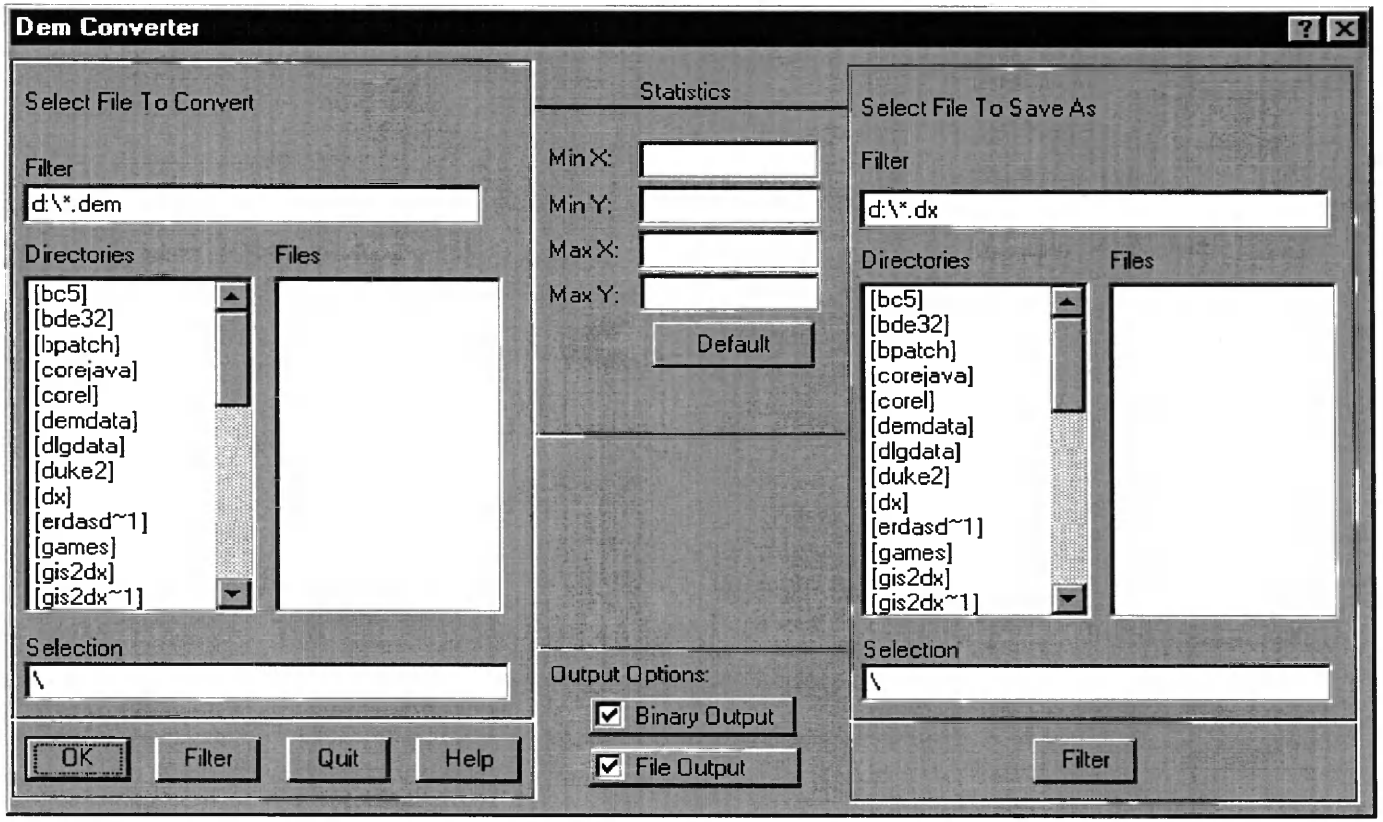
WIN95 VERSION



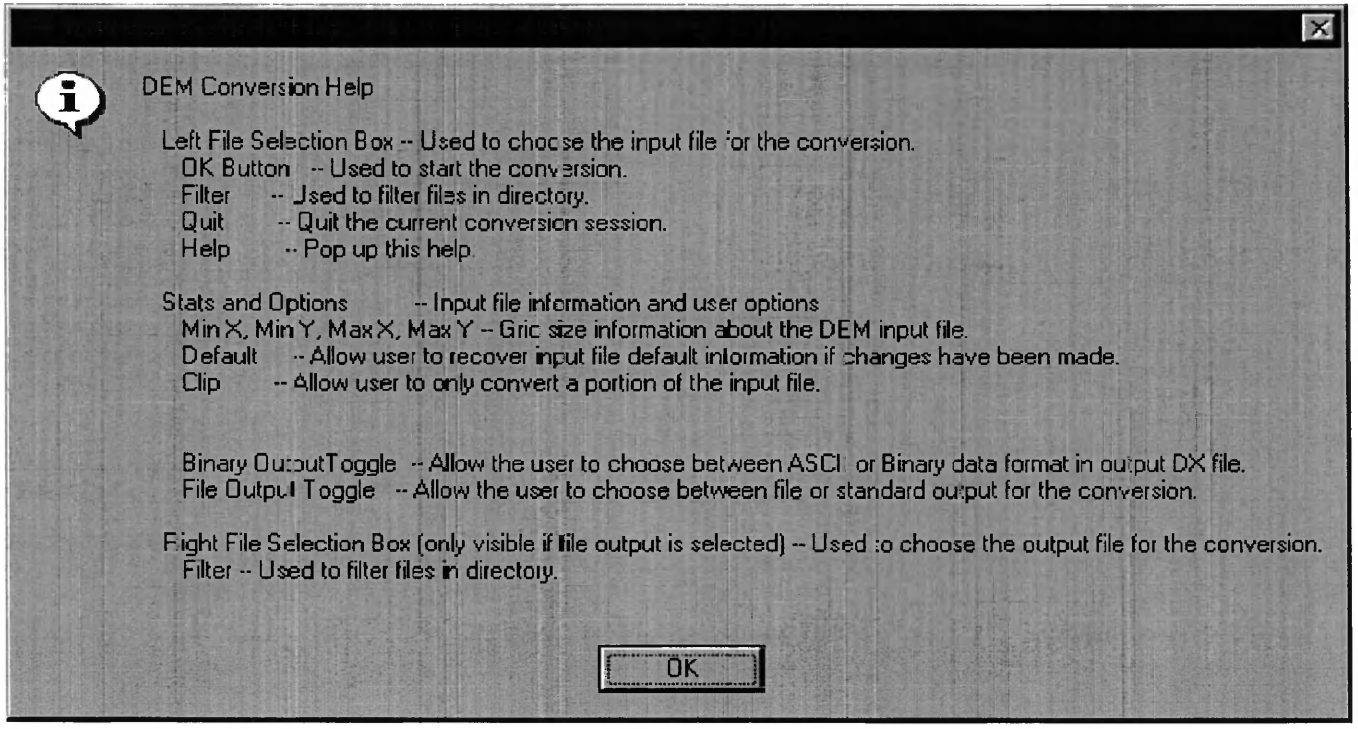
Main dialog of the Win95 version of GIS2DX.



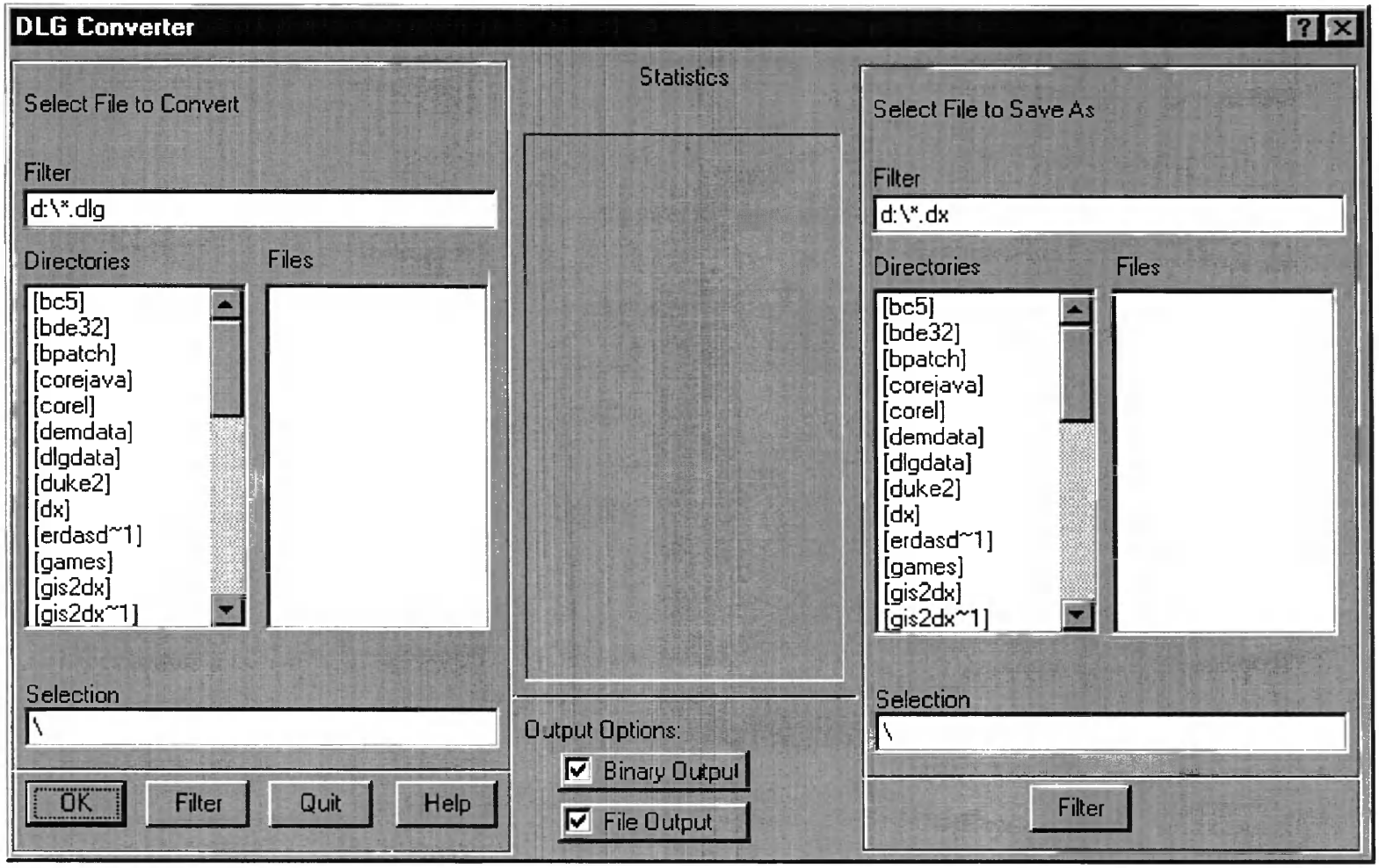
Main dialog help box.



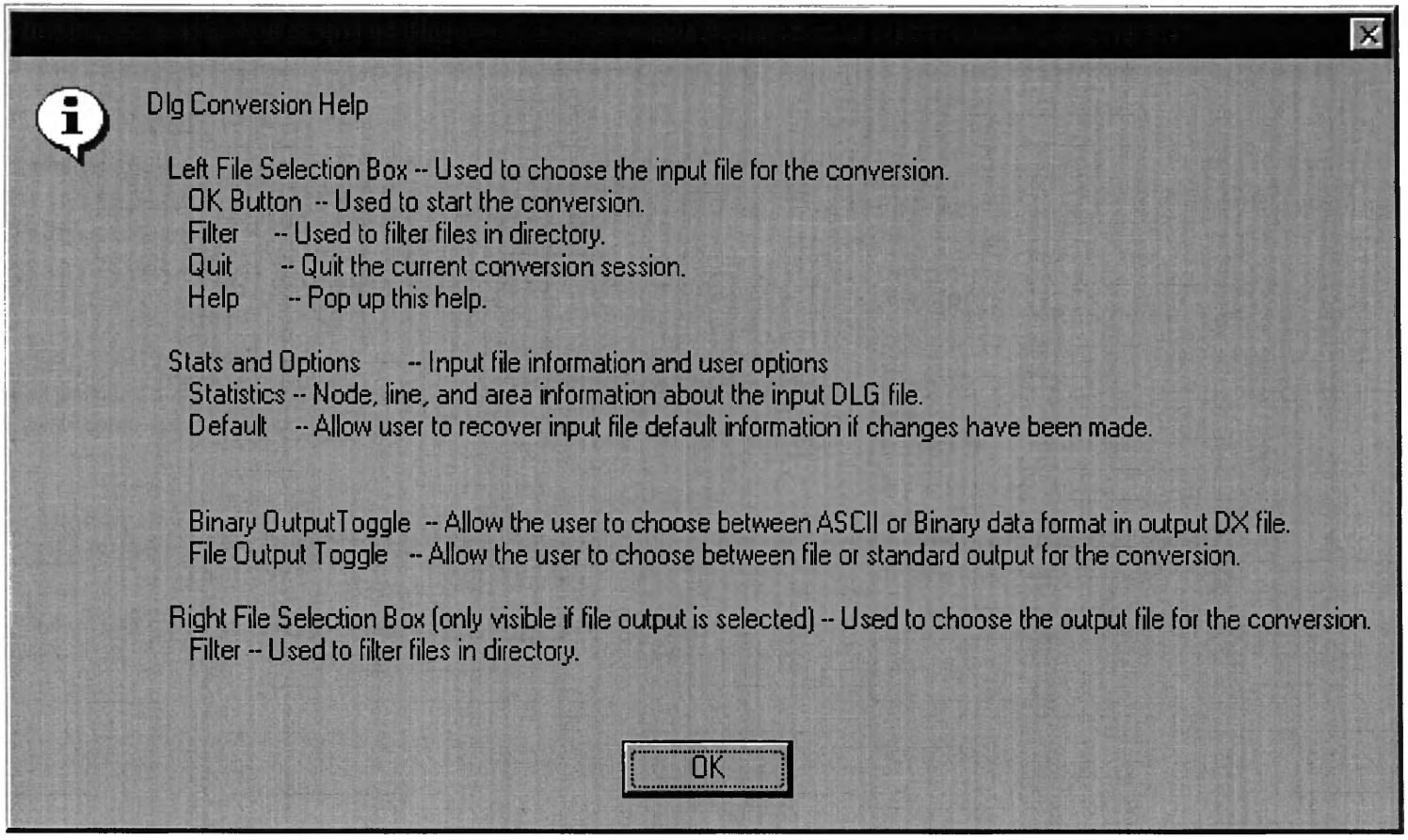
DEM Converter dialog box.



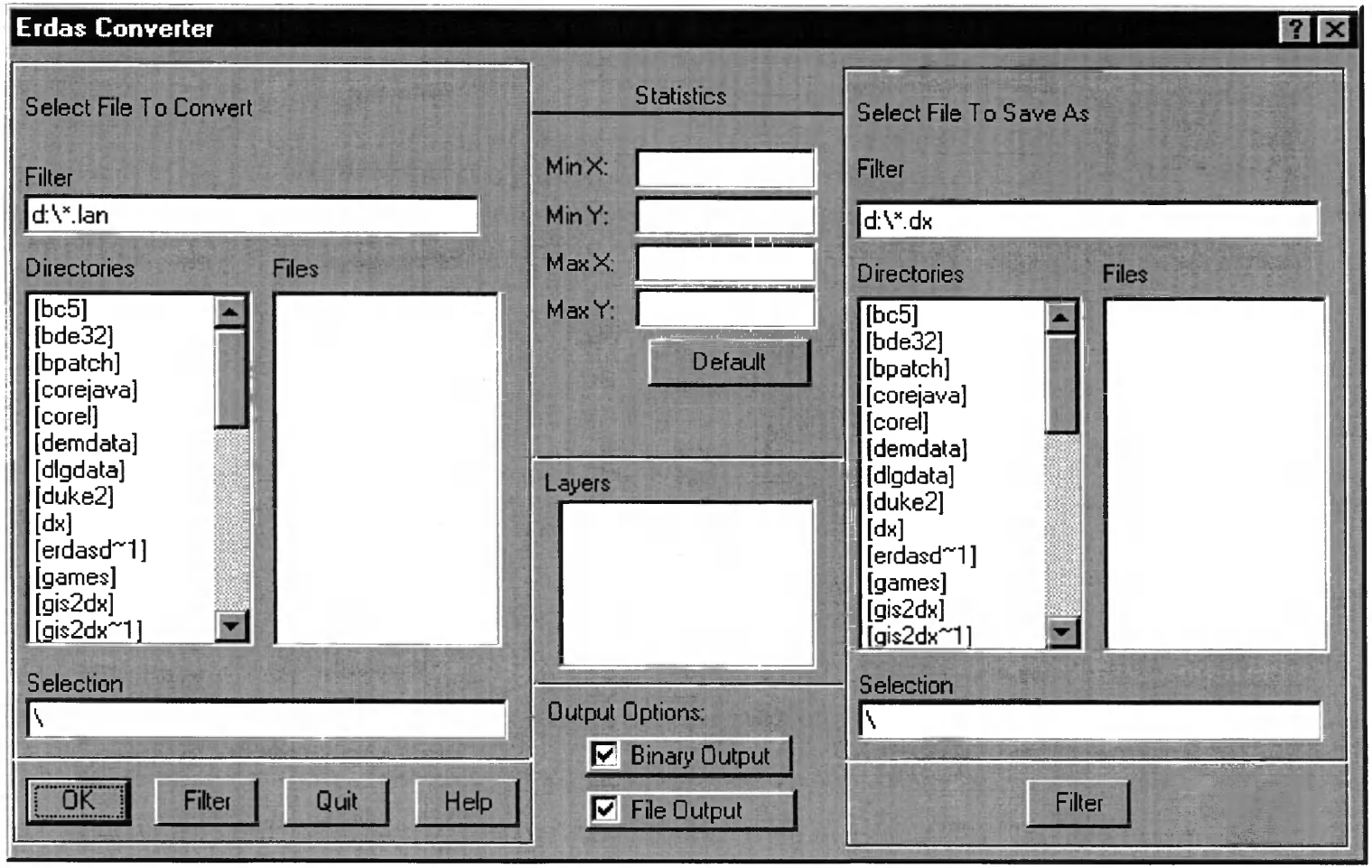
DEM converter help dialog box.



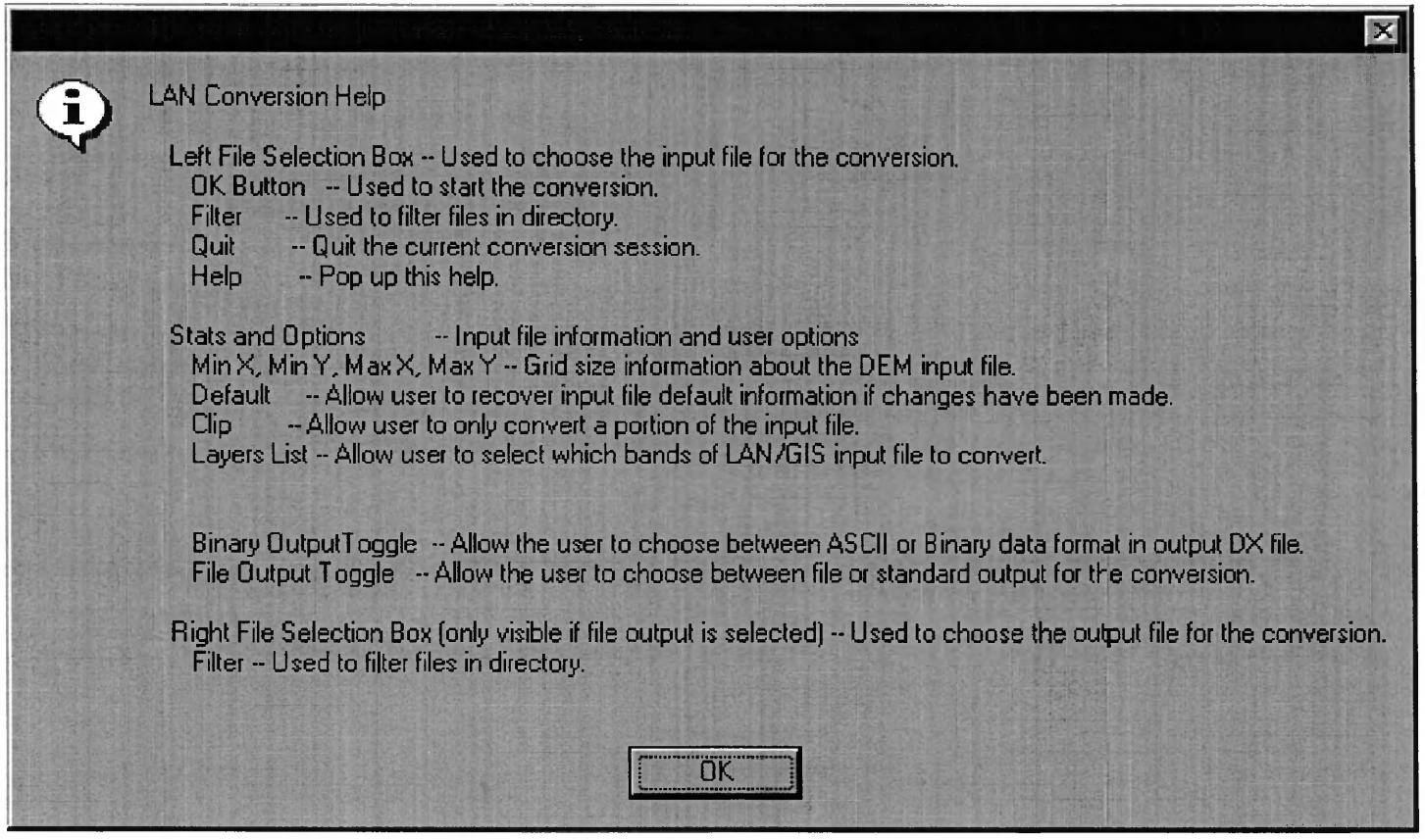
DLG converter dialog box.



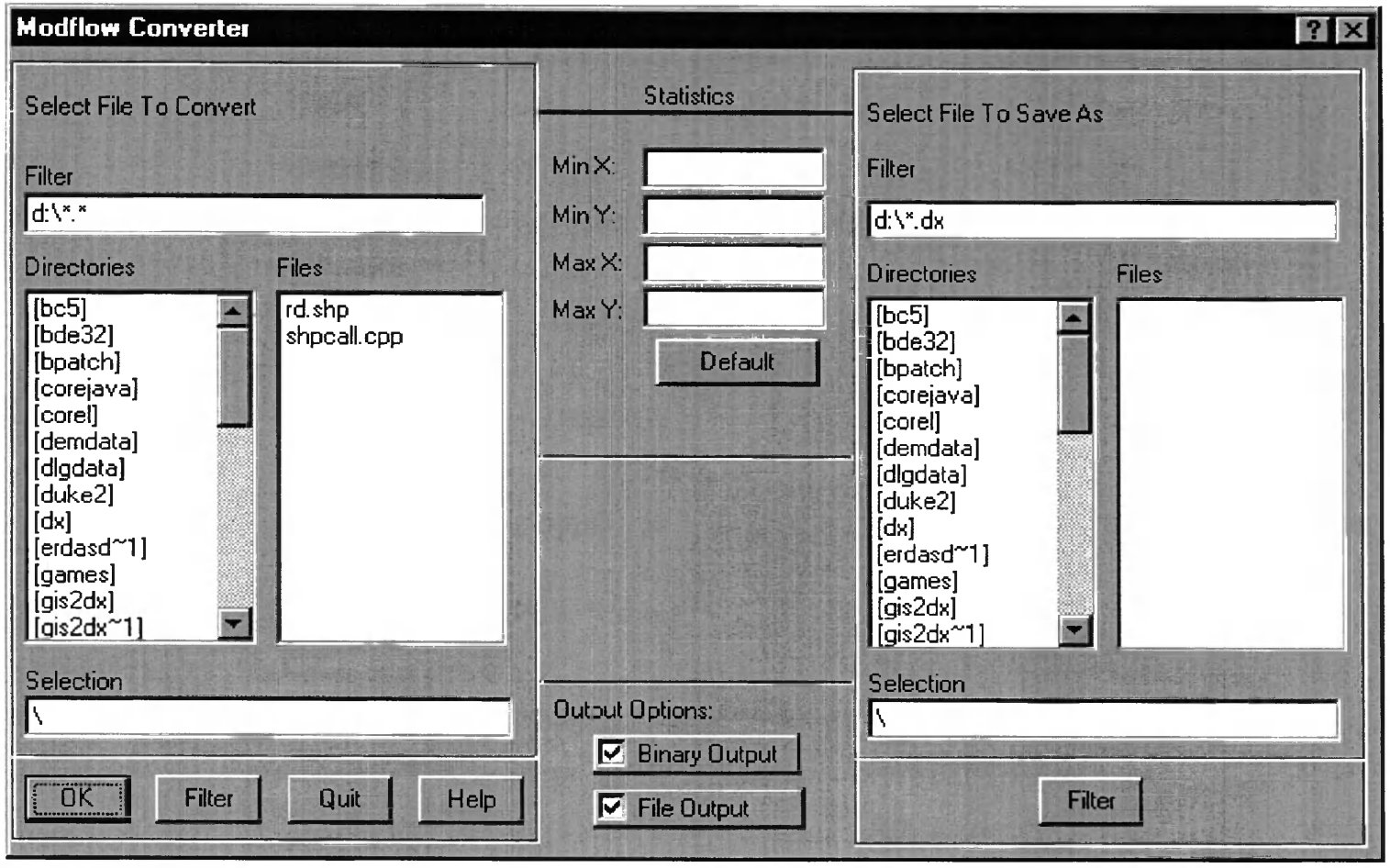
DLG converter help dialog box.



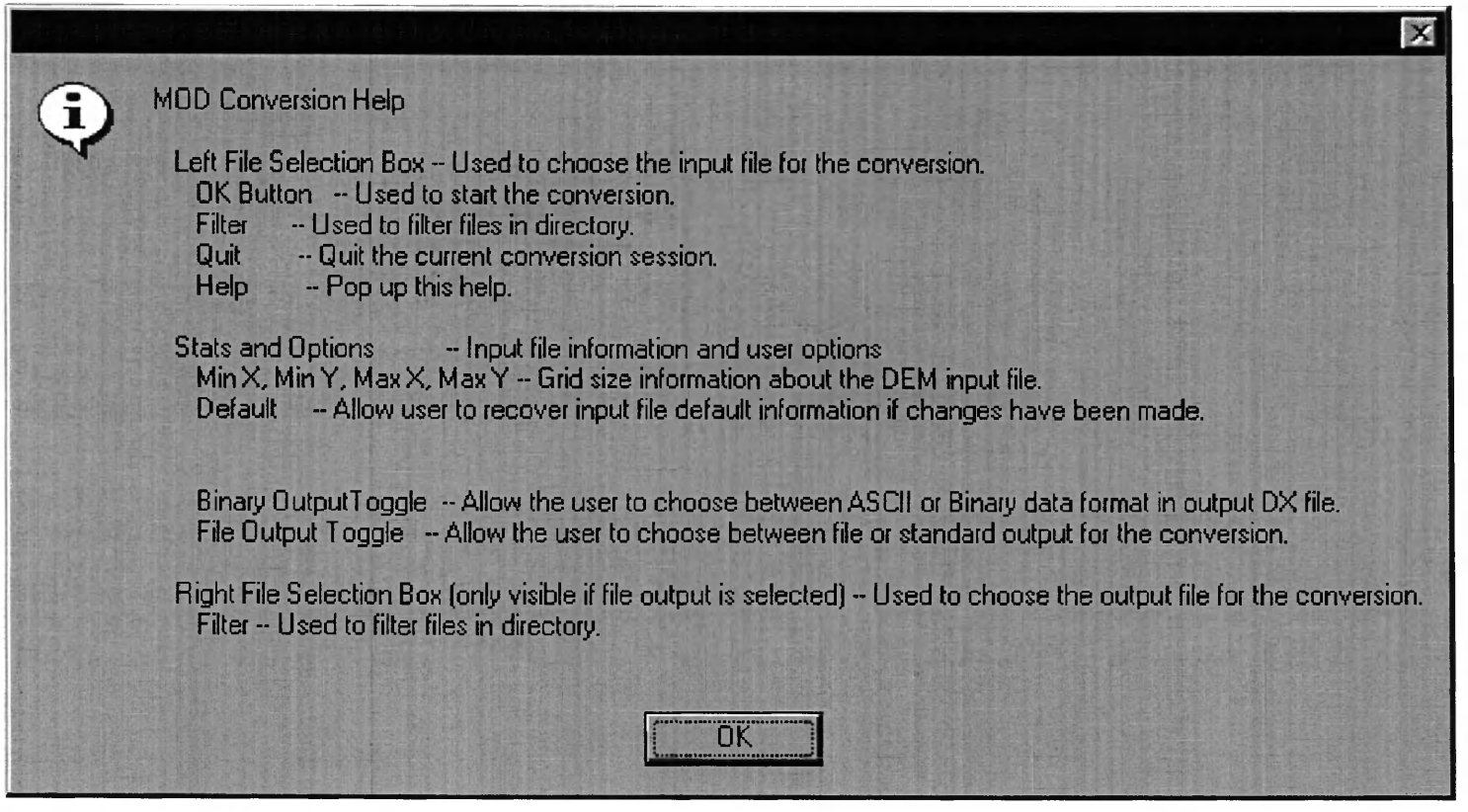
Erdas(.gis, .lan) converter dialog box.



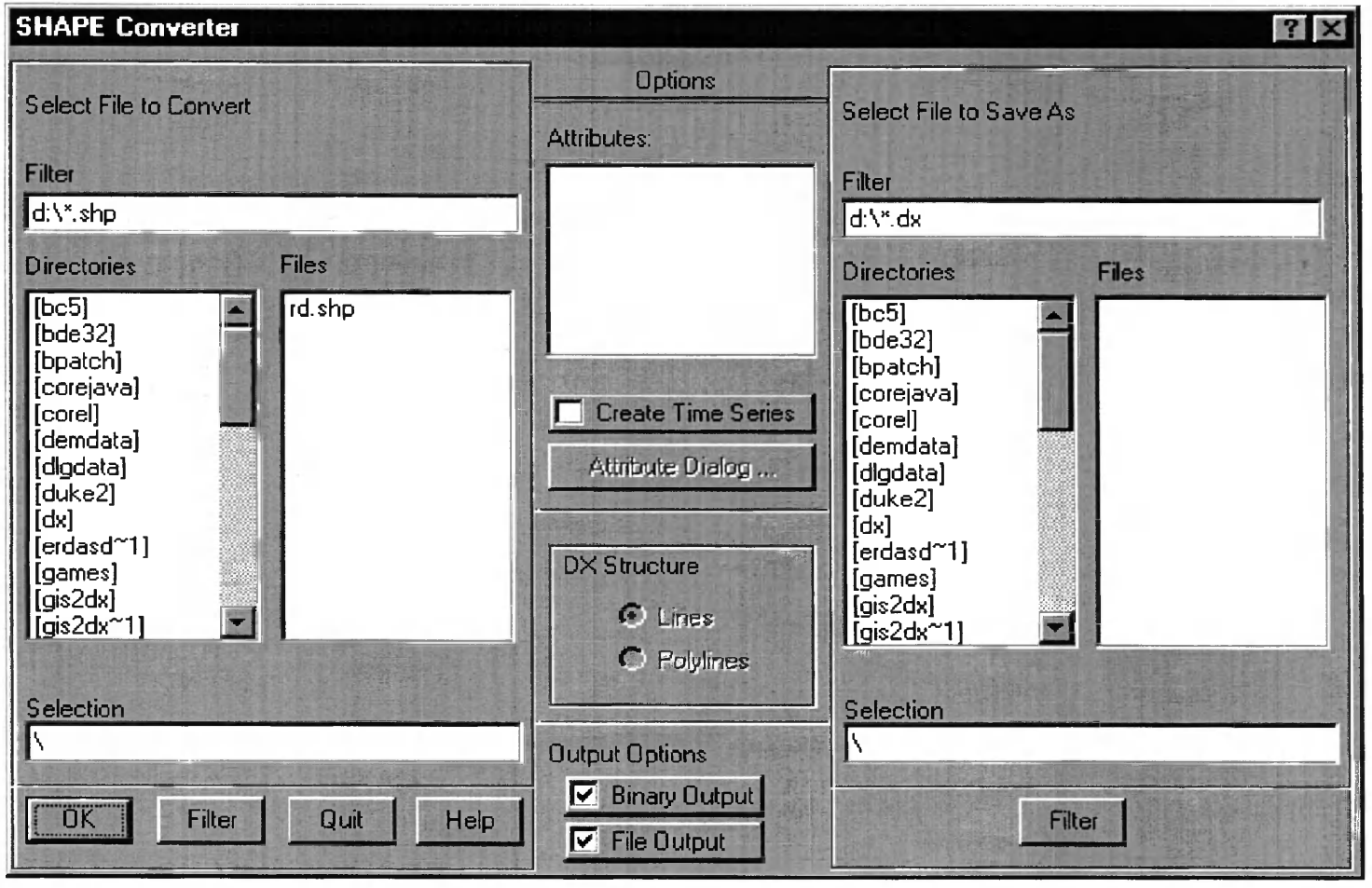
Erdas converter help dialog box.



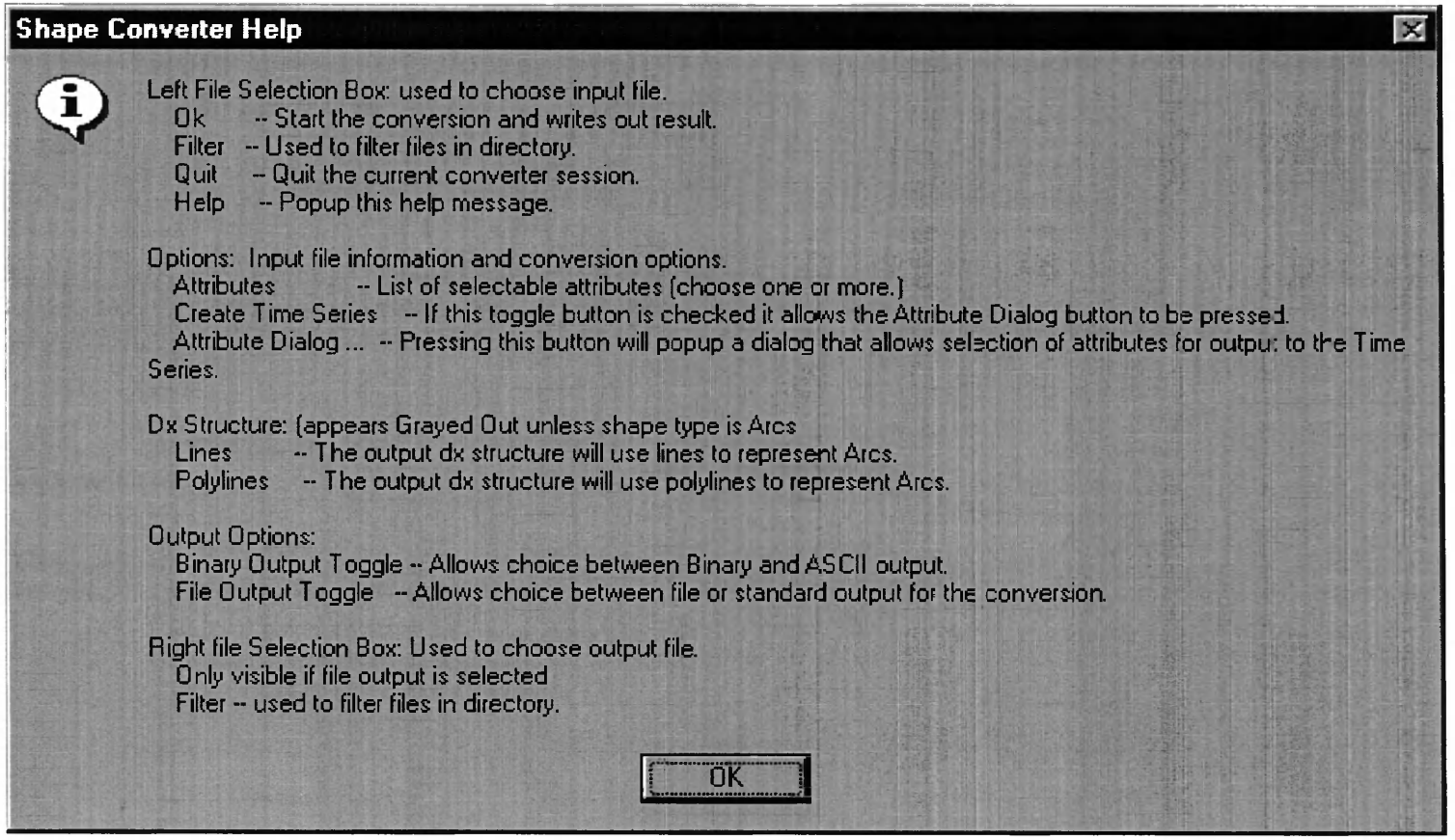
Modflow converter dialog box.



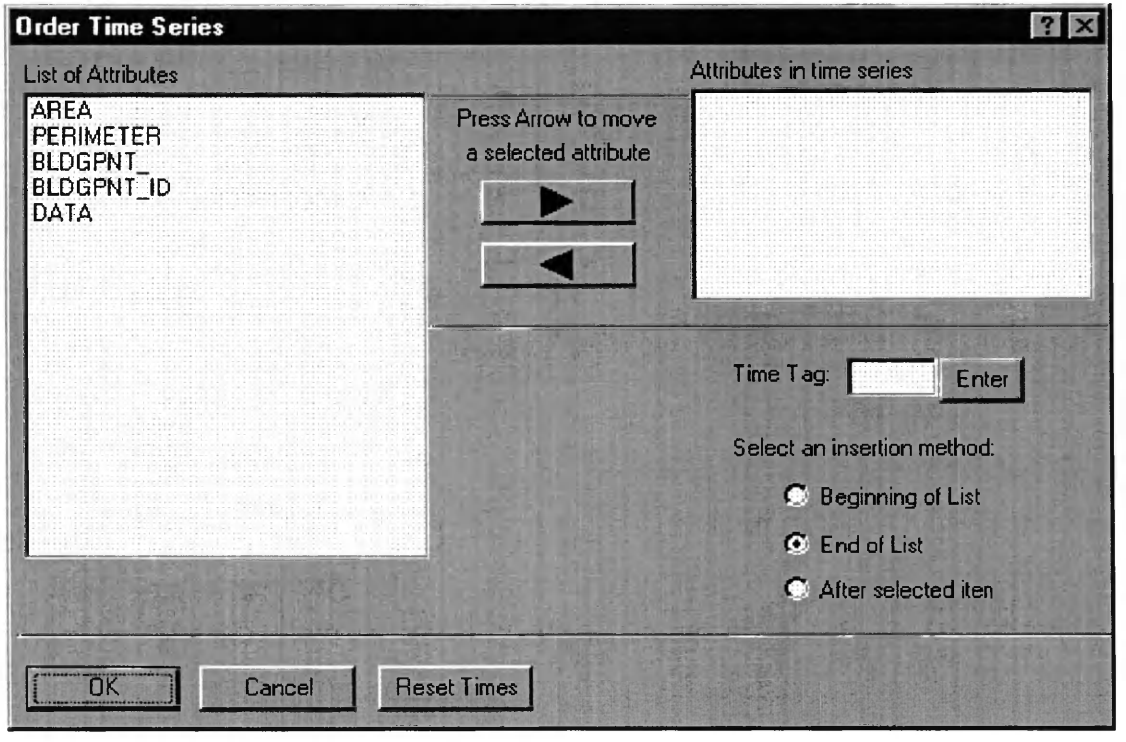
Modflow converter help dialog box.



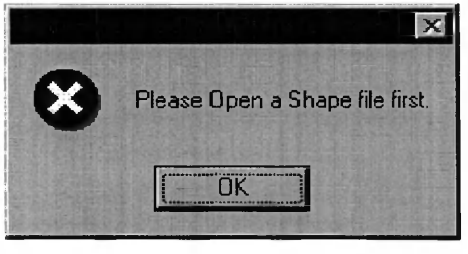
ArcInfo Shapefile converter dialog box.



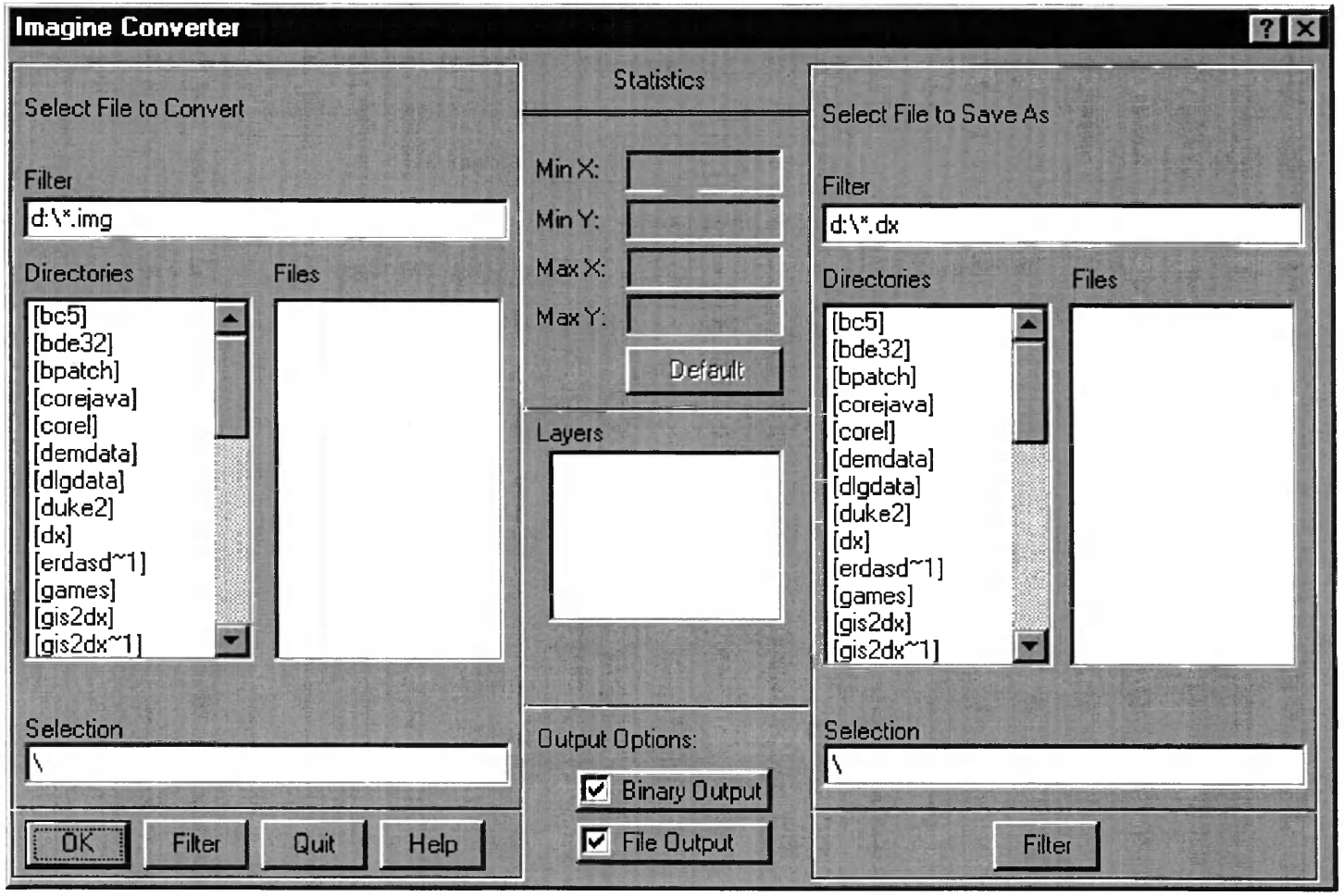
ArcInfo Shapefile converter help dialog box.



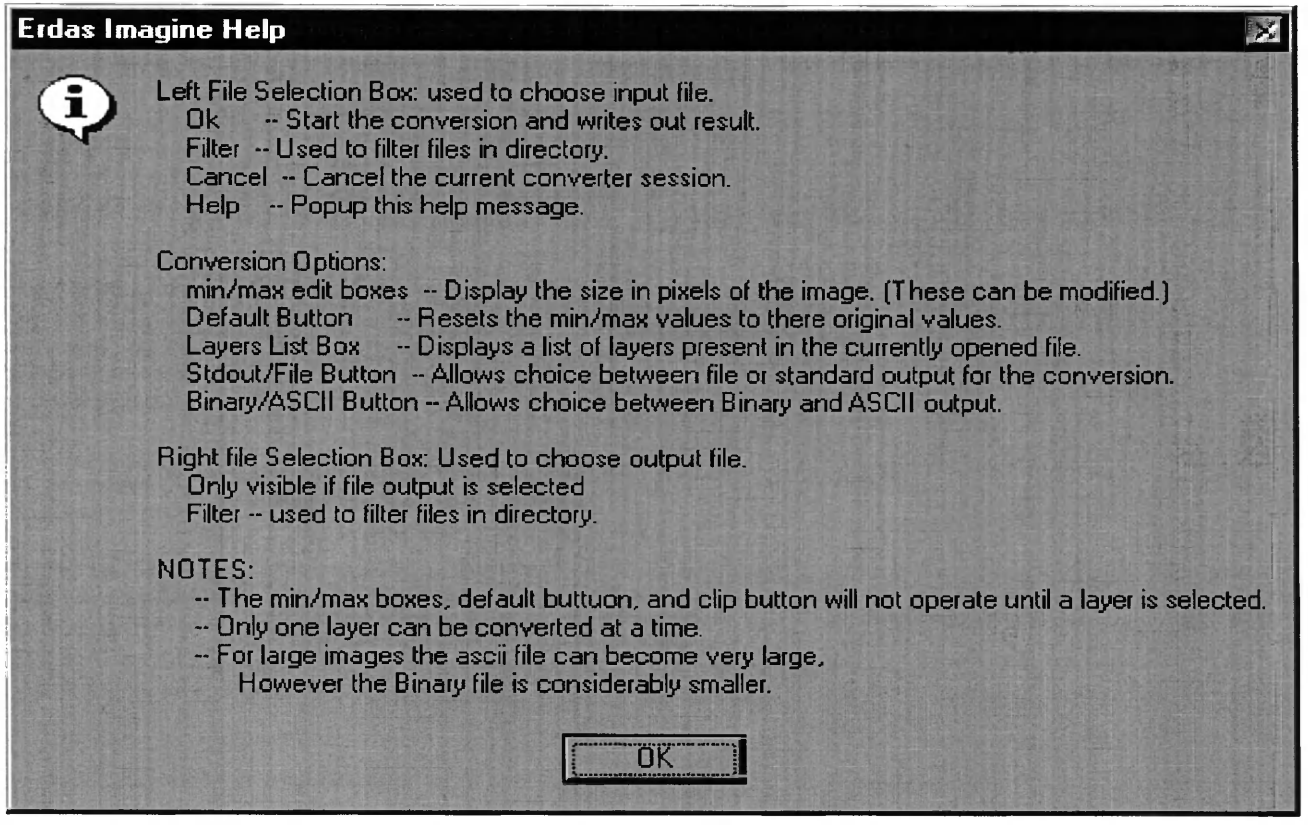
Shape converter time series dialog box.



Example of an error dialog. This occurs when trying to open the above dialog without a shape file loaded.



Erdas Imagine converter dialog box.

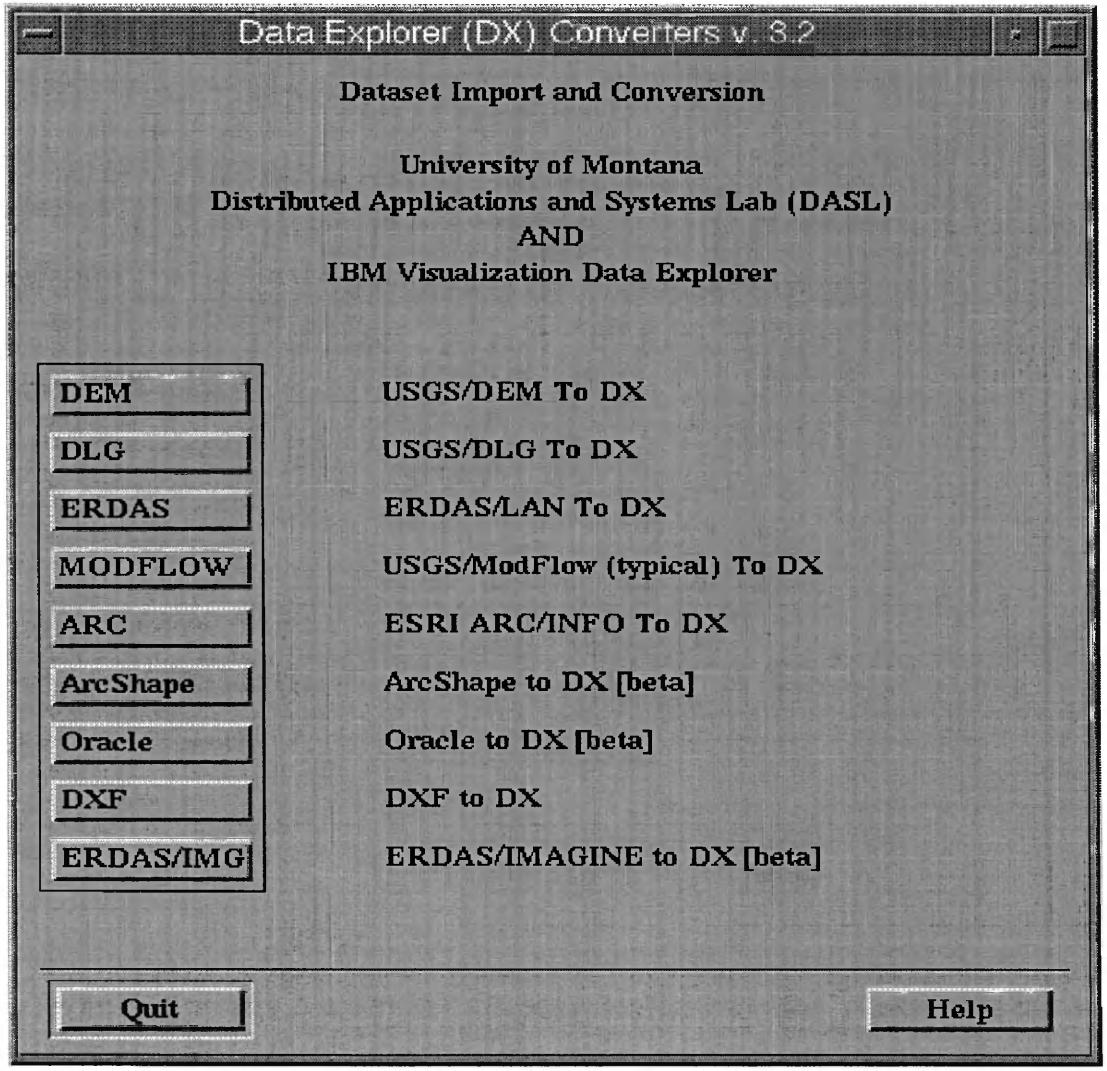


Erdas Imagine help dialog box.

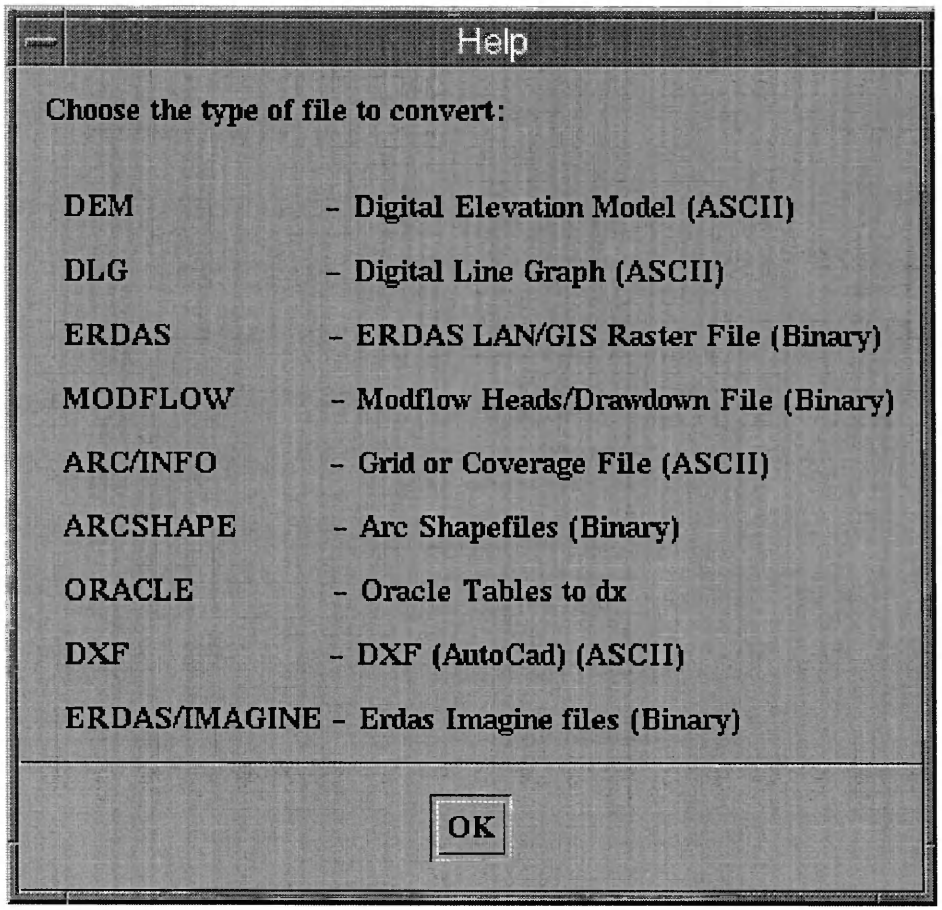
APPENDIX C

GIS2DX SCREEN SHOTS

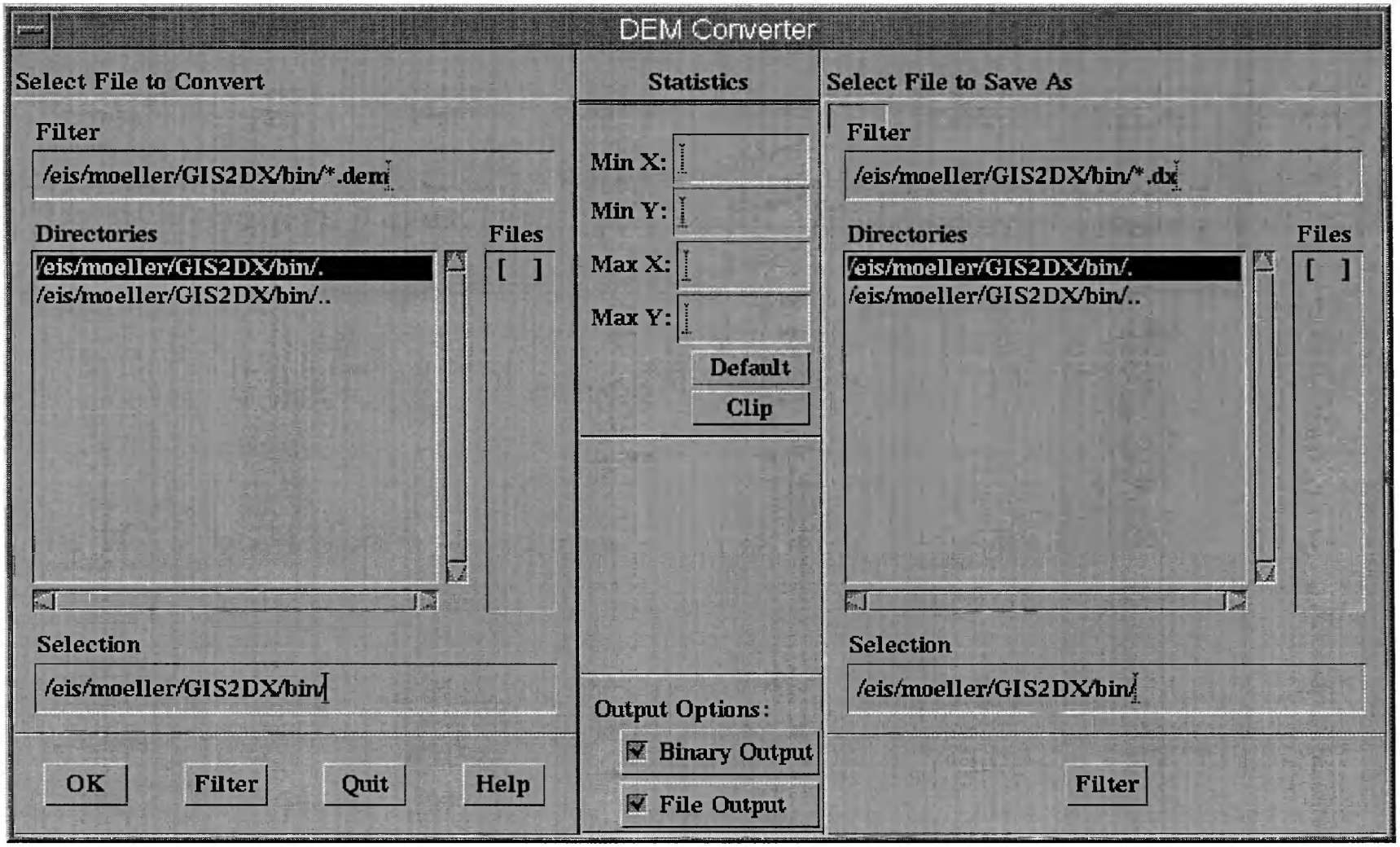
X/Motif Version



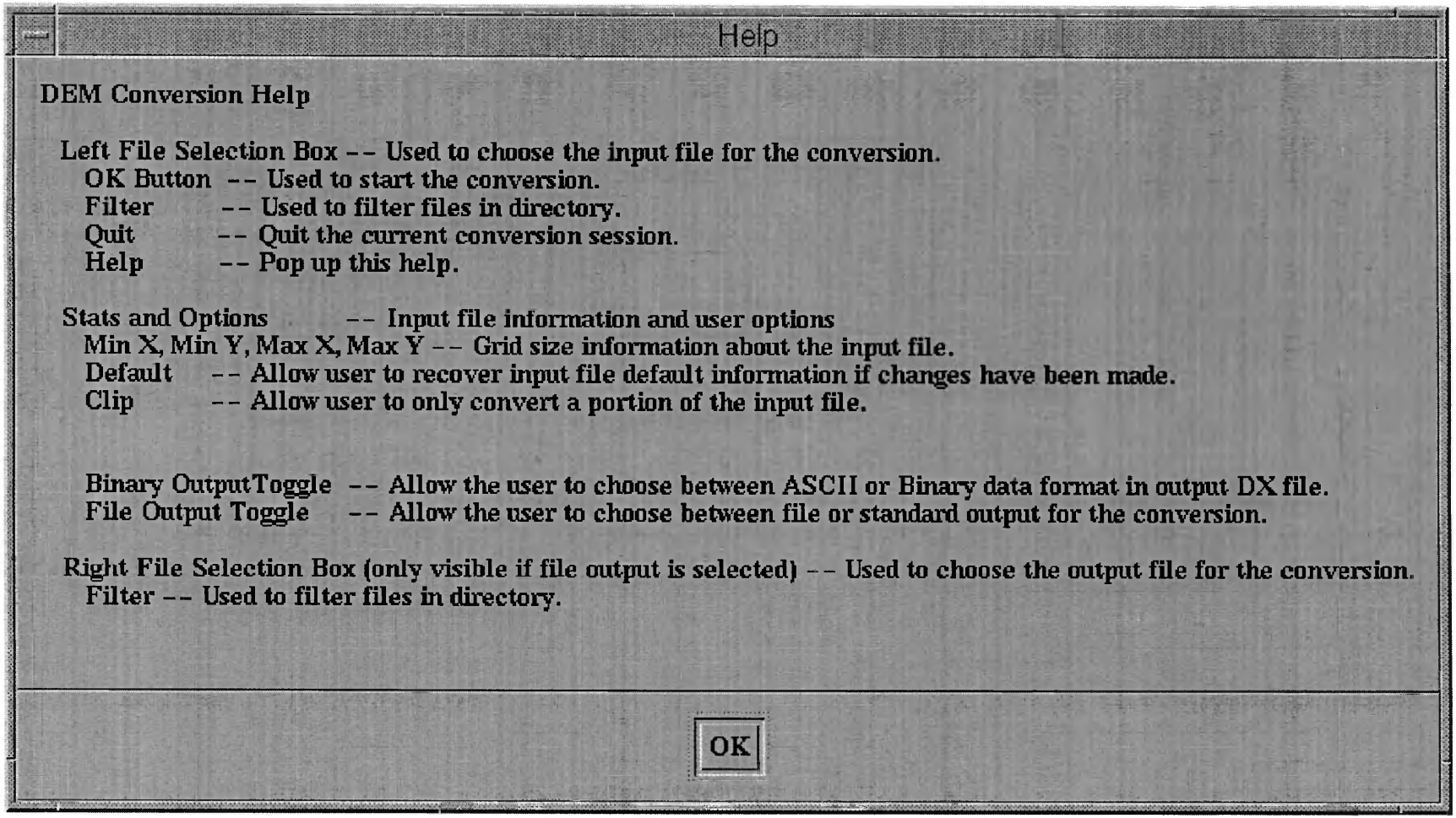
GIS2DX Main dialog box.



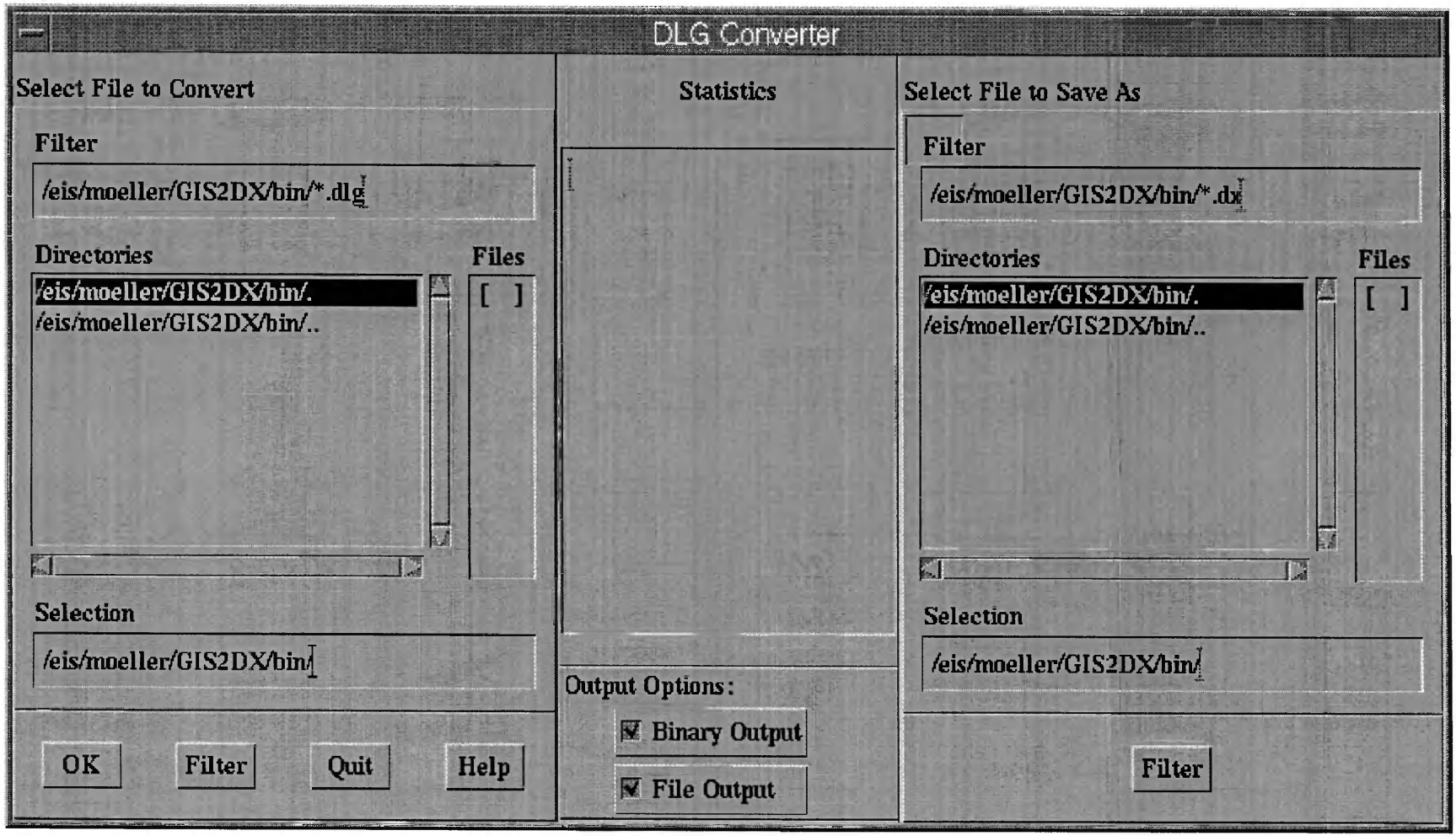
GIS2DX Main help dialog box.



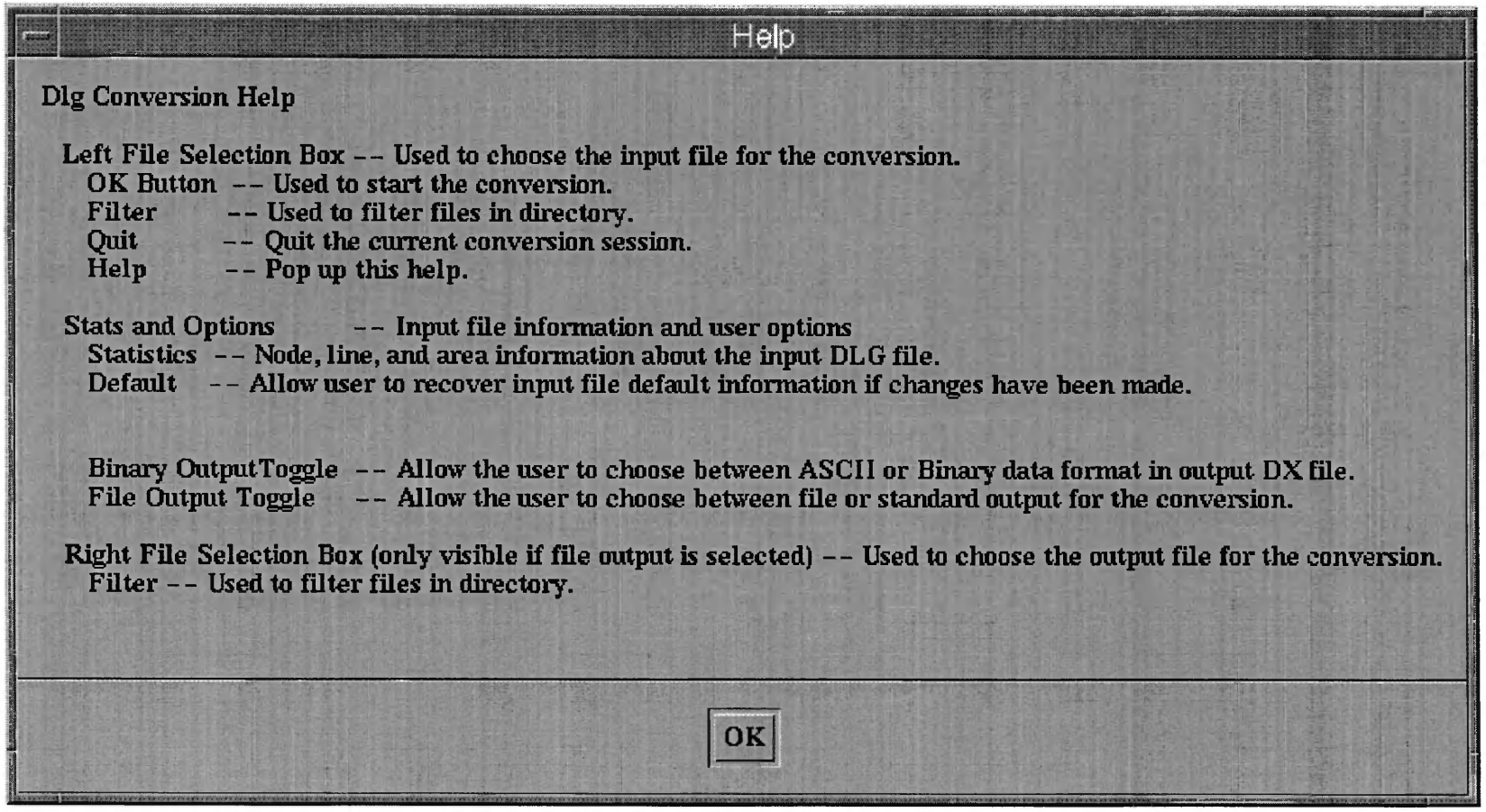
Dem converter dialog box.



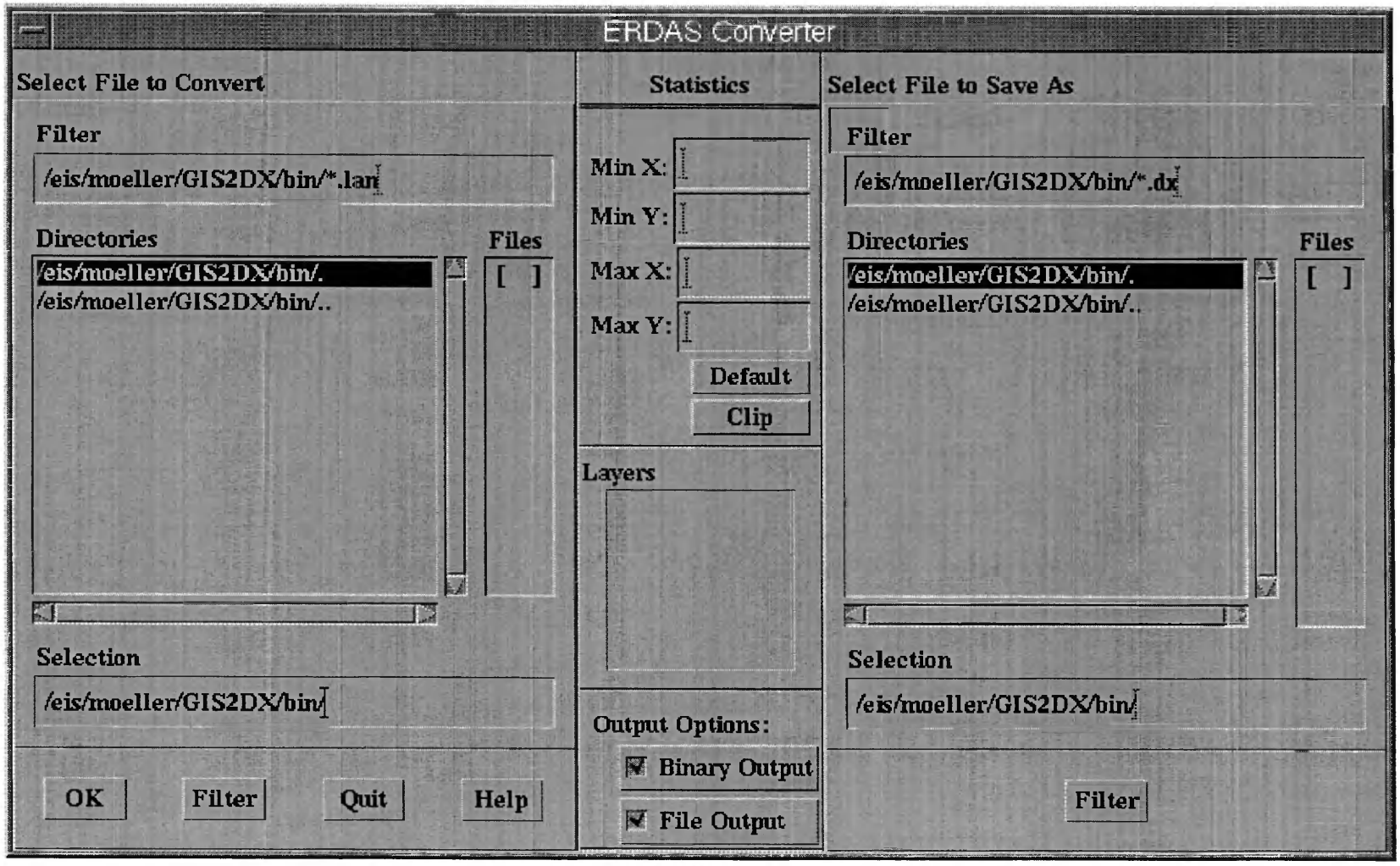
Dem converter help dialog box.



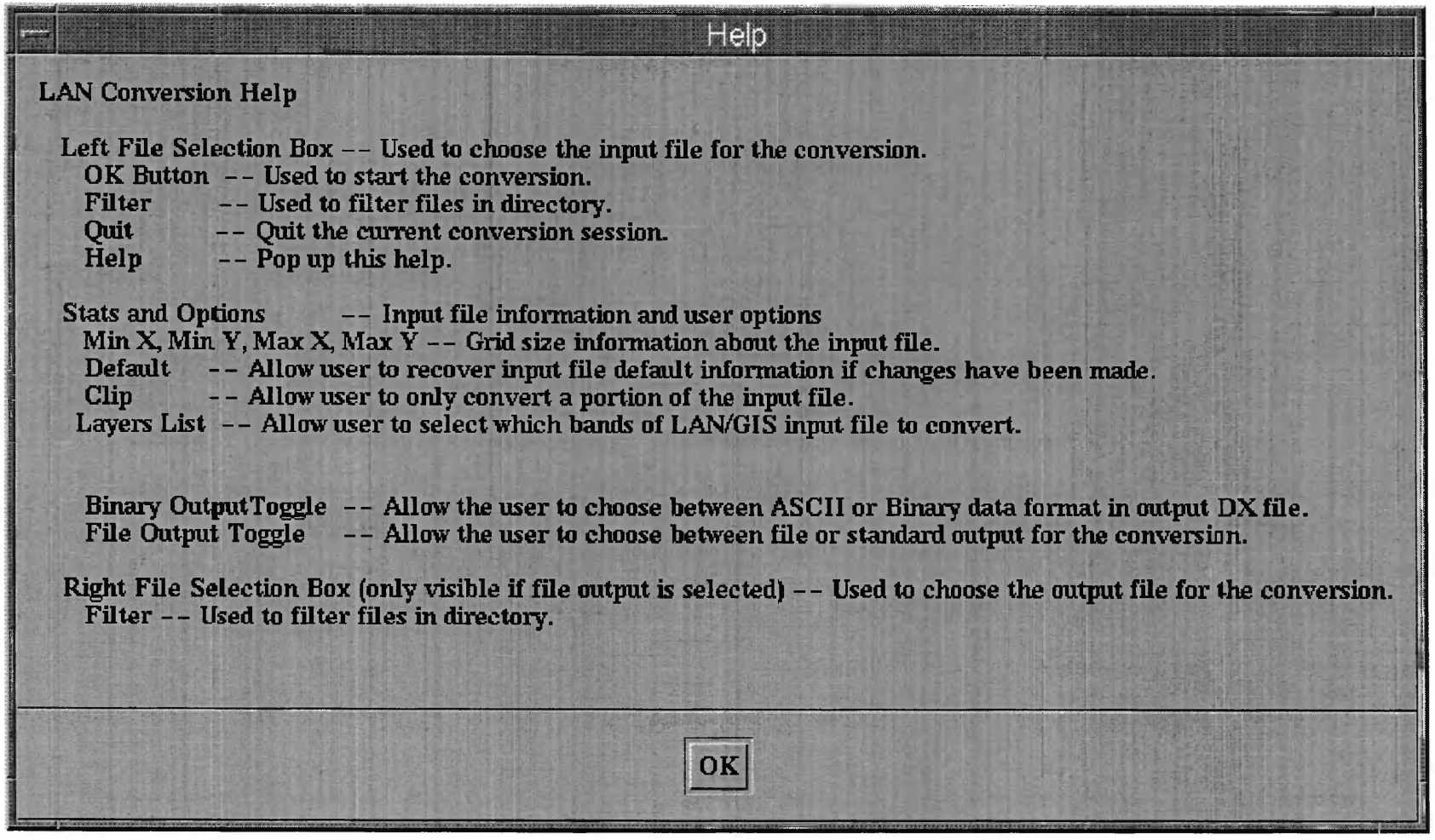
Dlg converter dialog box.



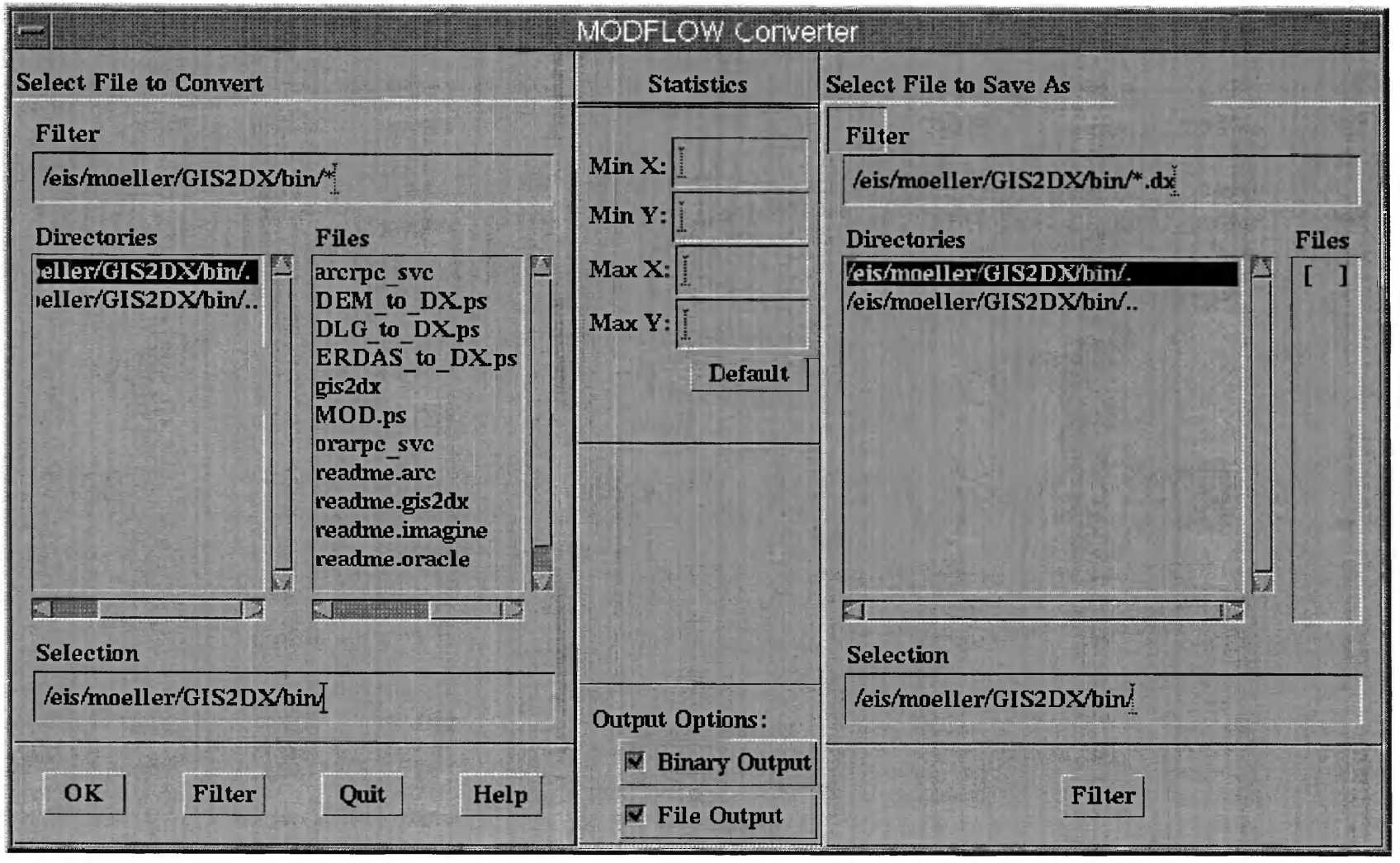
Dlg converter help dialog box.



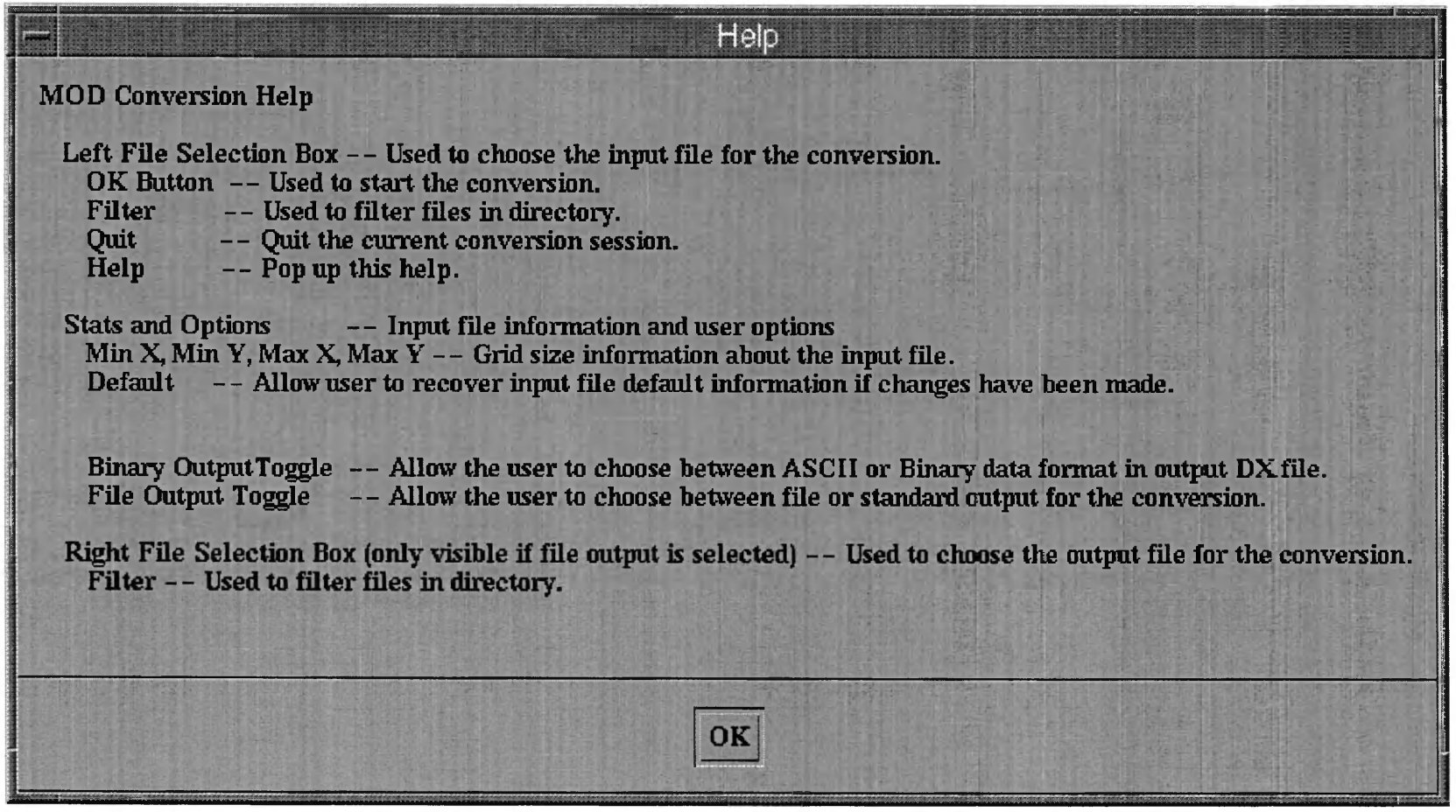
Erdas converter dialog box.



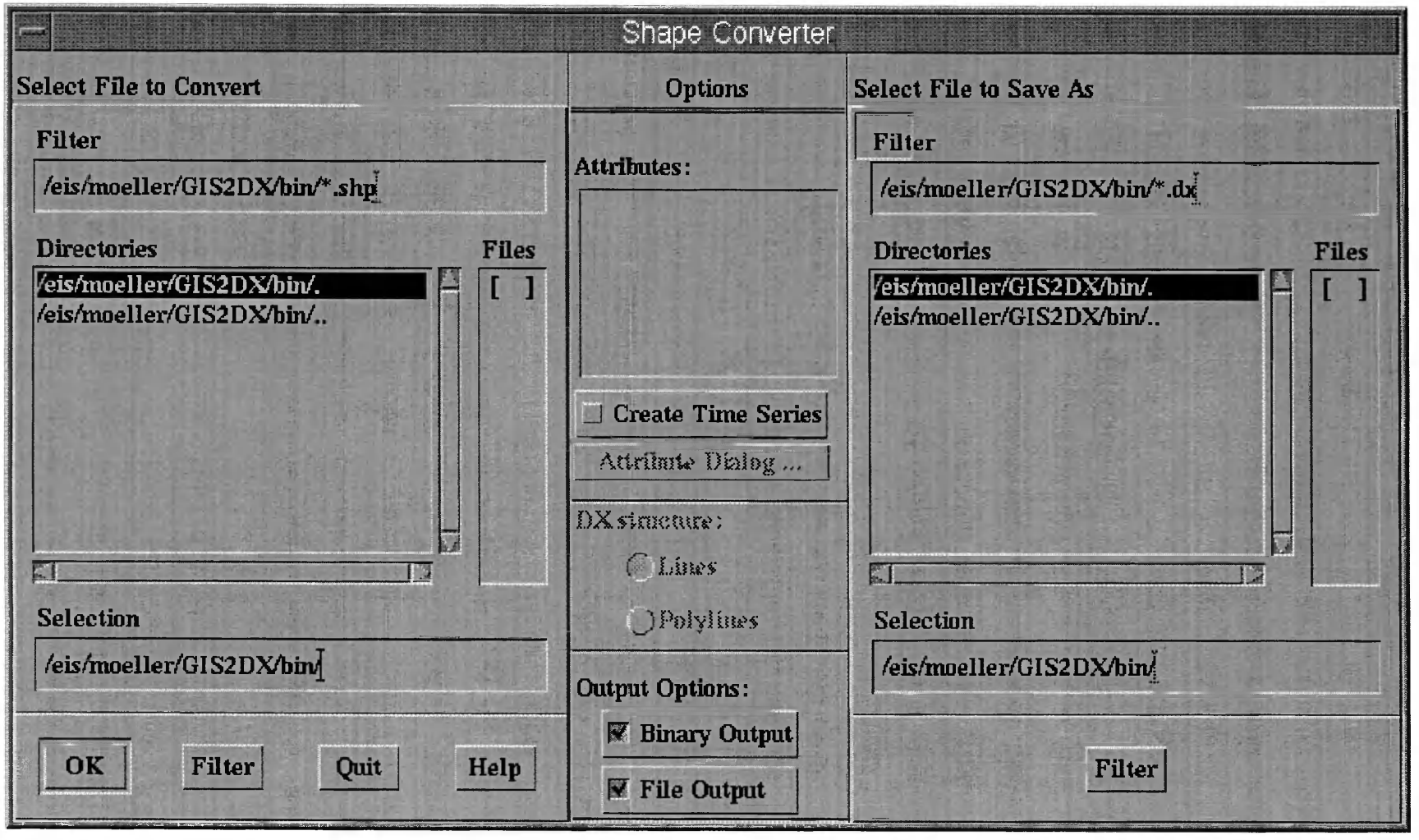
Erdas converter help dialog box.



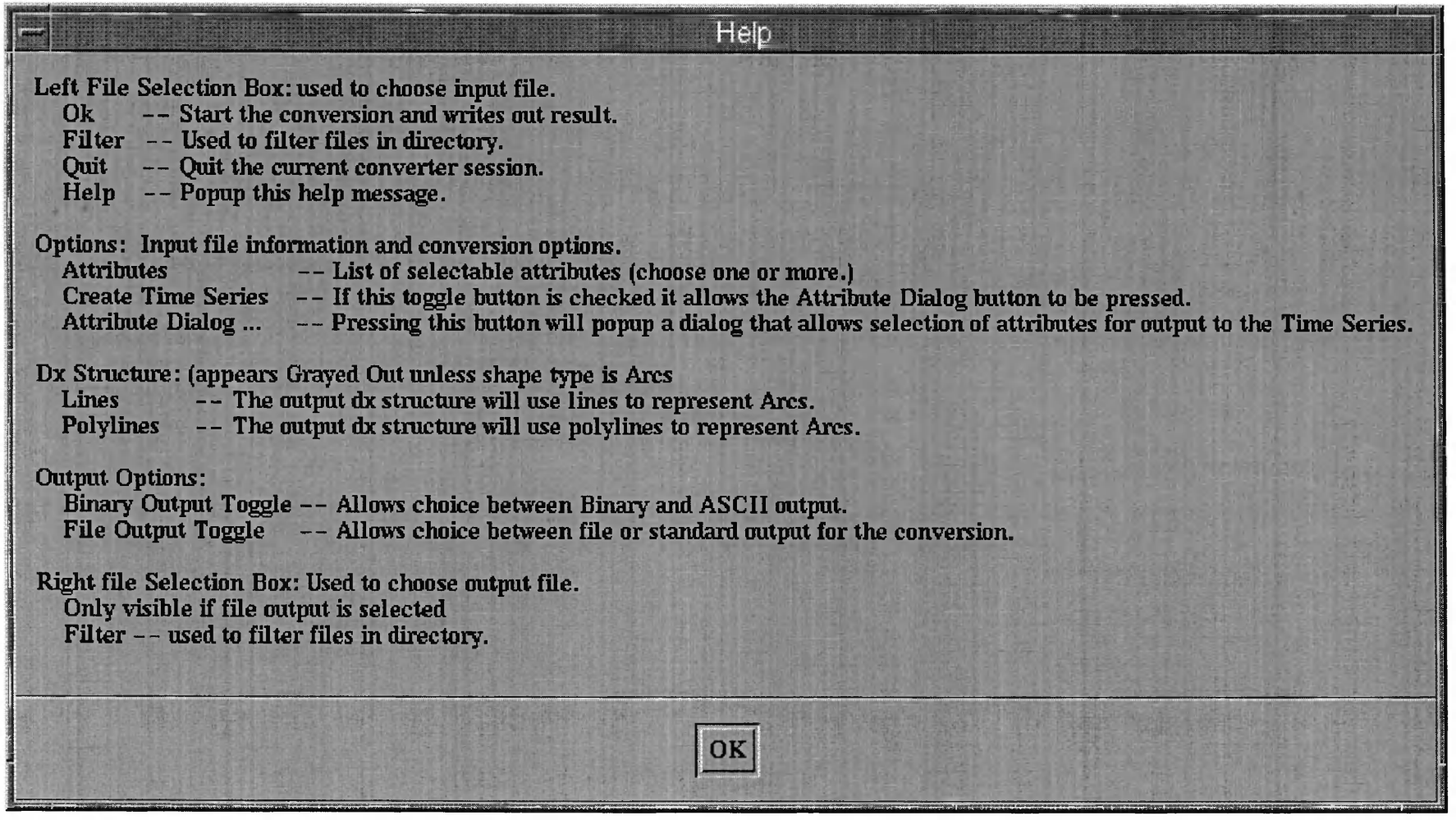
Modflow converter dialog box.



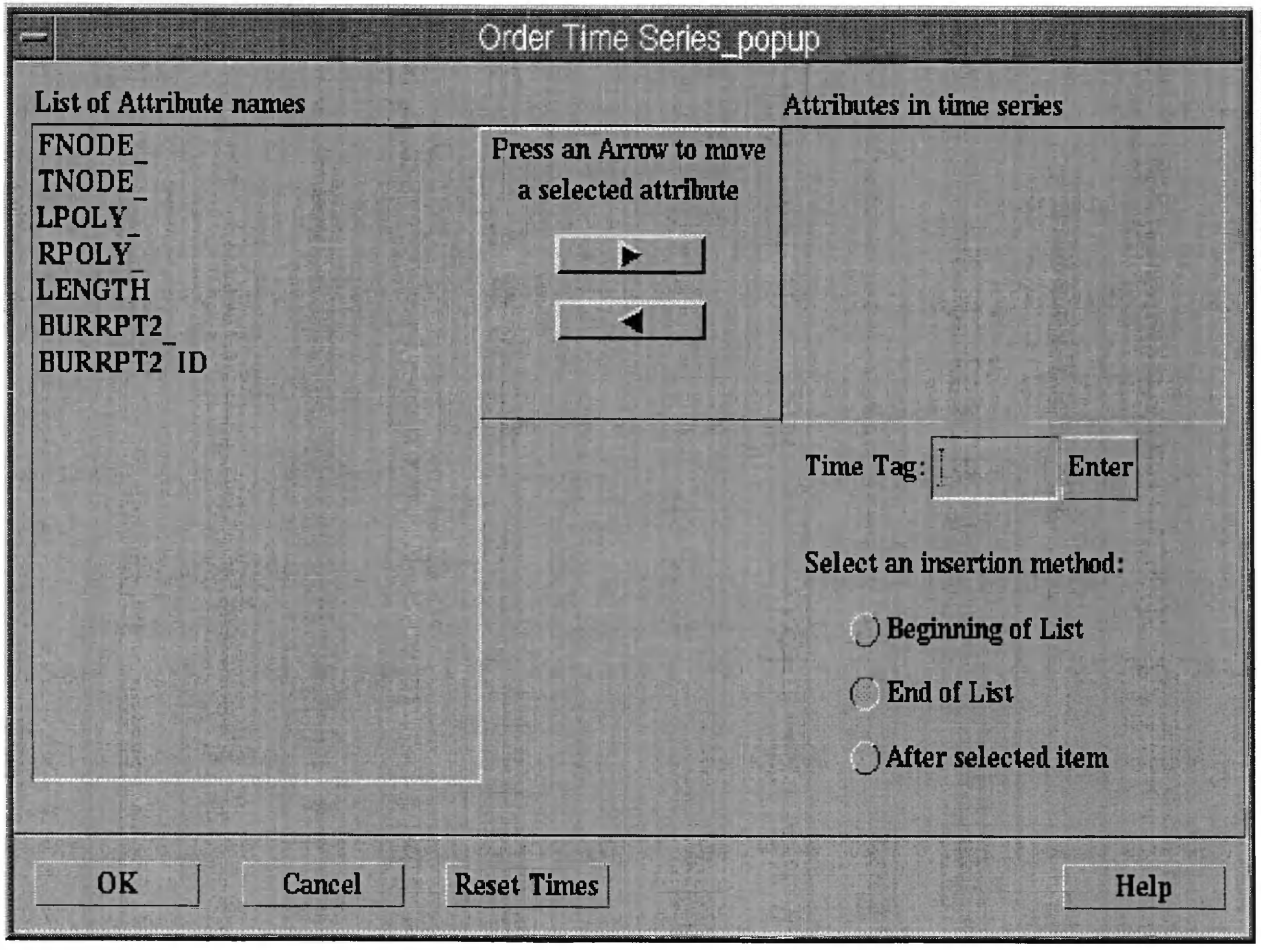
Modflow converter help dialog box.



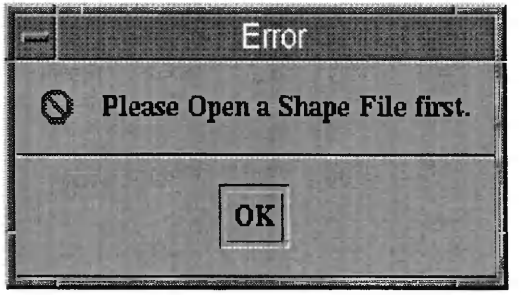
ArcInfo Shapefile converter dialog box.



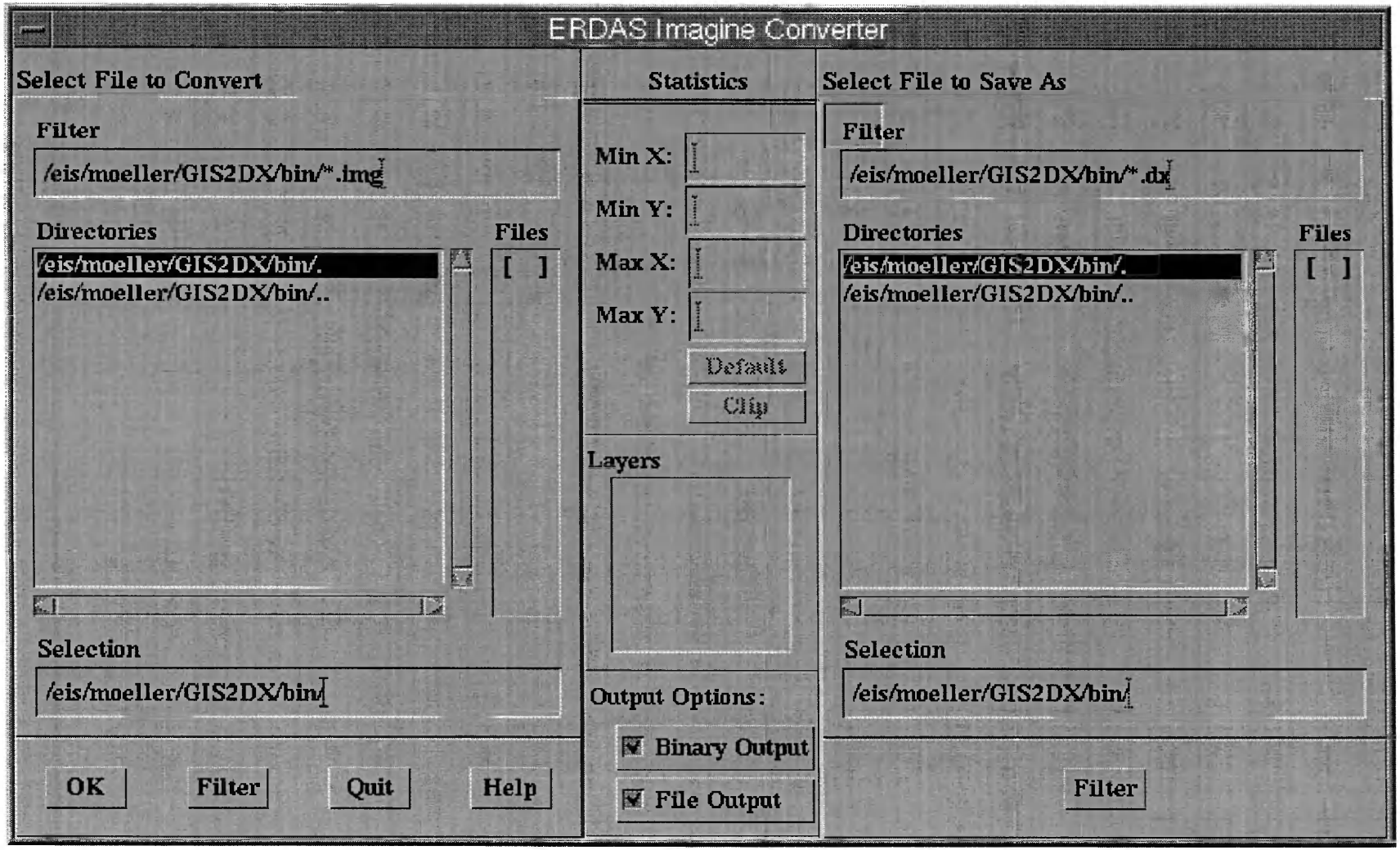
ArcInfo Shapefile converter dialog box.



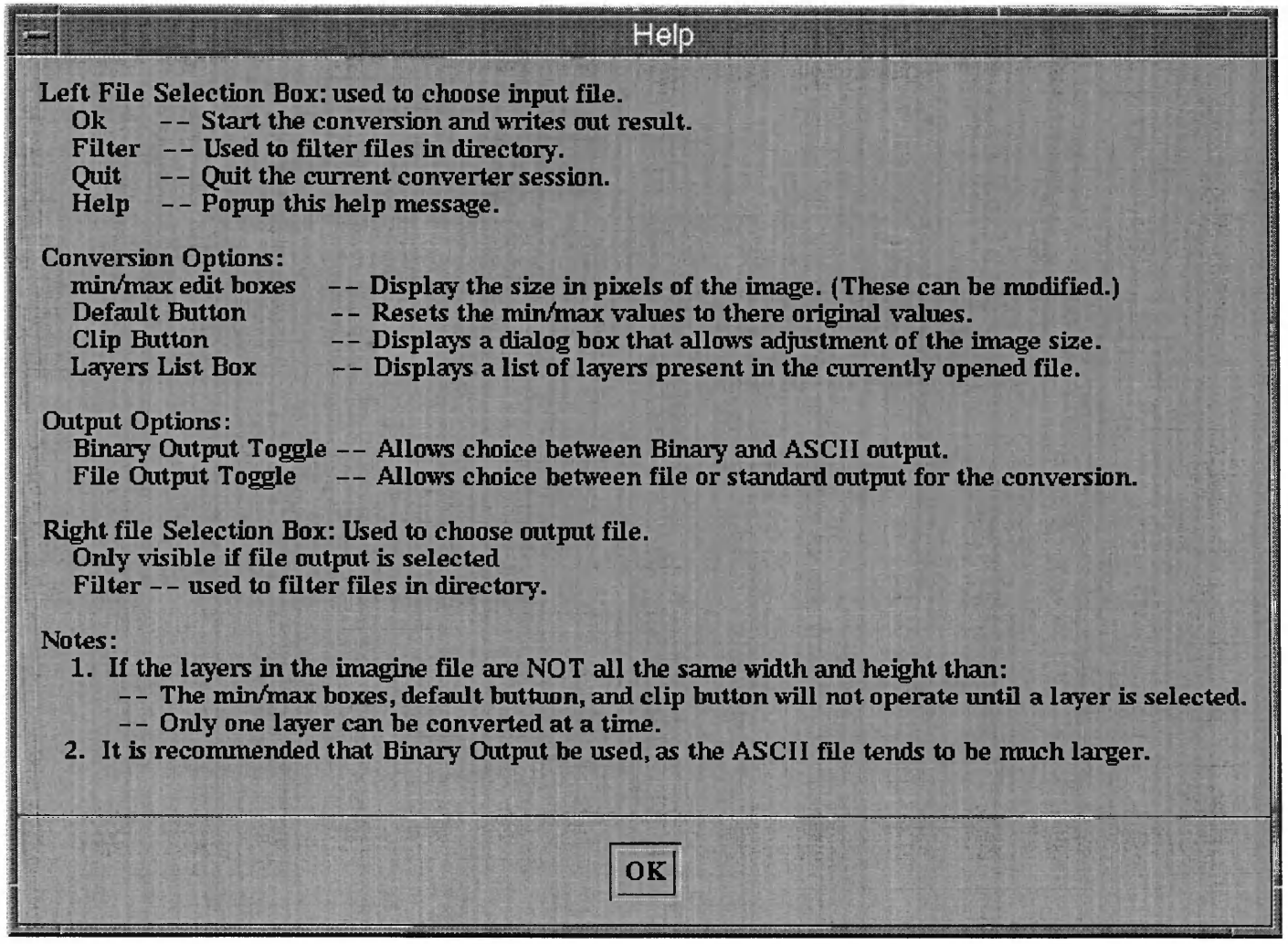
Time Series selection dialog box.



Example of an Error dialog box. This dialog is displayed if the user tries to open the time series dialog with no shapefile open.



Erdas Imagine converter dialog box.



Erdas Imagine converter help dialog box.

Appendix D

Source code for Win95 version of GIS2DX. Some non-GUI code has been omitted.

```
#ifndef GISOWL_H
#define GISOWL_H

#include <owl\applicat.h>
#include <owl\decframe.h>
#include <owl\dialog.h>
#include <owl\edit.h>
#include <owl\listbox.h>
#include <owl\button.h>
#include <owl\static.h>
#include <owl\checkbox.h>
#include <owl\static.h>
#include <owl\radiobut.h>
#include <owl\glyphbtn.h>

#endif
```

```
#ifndef GISMAIN_H
#define GISMAIN_H

#include "gisowl.h"
#include "convW.rh"
#include "shp2dxW.rh"
#include "img2dxW.rh"
#include "shp_callbacks.h"
#include "callbacksIMG.h"
#include "callbacks.h"

class TGis2dxFrame: public TDecoratedFrame
{
 public:
            TGis2dxFrame(TWindow* parent, const char* name,
           TWindow* clientWnd, char *fontsize);
};

class TGis2dxApplication: public TApplication
{
 public:
            TGis2dxApplication(char *s): TApplication() {strcpy(fontsize,s);}
            void InitMainWindow();
 private:
   char fontsize[20];
};

class TGis2dxDialog: public TDialog
{
 public:
            TGis2dxDialog(TWindow* parent,TResId resId);
            void SetFrame(TDecoratedFrame* frm) {frame = frm;}
            void Terminate();
   void dem2dx();
   void dlg2dx();
   void mod2dx();
   void lan2dx();
            void img2dx();
            void shp2dx();
   void HelpTop();
            TStatic* text1;
            TStatic* text2;
 private:
                  TDecoratedFrame* frame;
    TDemDialog     *demdlg;
    TDlgDialog     *DLGdlg;
    TDemDialog     *moddlg;
    TDemDialog     *landlg;
                  TImgDialog     *imgdlg;
                  TShpDialog     *shpdlg;

 DECLARE_RESPONSE_TABLE(TGis2dxDialog);
};

#endif
```

```
// ****************************************
// File:      gismain.cpp
// Author: Kirk A. Moeller
// ****************************************
#include "gismain.h"
#include "stdio.h"

TGis2dxFrame::TGis2dxFrame(TWindow* parent, const char* name,TWindow* clientWnd,
        char *fontsize):TDecoratedFrame(parent,name,clientWnd)
{
// H = Resource Workshop value + 13
// W = Resource Workshop value + 20

 if (strcmp(fontsize,"large") != 0)
 {
  // ** small fonts set in windows 95 **
  Attr.W = 430;
  Attr.H = 400;
 }
 else
 {
  // ** large fonts set in windows 95 **
  Attr.W = 570;
  Attr.H = 485;
 }
}

void TGis2dxApplication::InitMainWindow()
{
 TGis2dxDialog* dlg = new TGis2dxDialog(0,IDD_MAINDLG);
 dlg->SetBkgndColor(TColor::LtGray);
 TDecoratedFrame* frame =
    new TGis2dxFrame(0,"Data Explorer (DX) Converters v. 3.2 [beta]",dlg,fontsize);
 dlg->SetFrame(frame);

 SetMainWindow(frame);
}

DEFINE_RESPONSE_TABLE1(TGis2dxDialog,TDialog)
 EV_CHILD_NOTIFY(IDCANCEL, BN_CLICKED, Terminate),
 EV_CHILD_NOTIFY(IDHELP, BN_CLICKED, HelpTop),
 EV_CHILD_NOTIFY(IDC_DEMPB,BN_CLICKED,dem2dx),
 EV_CHILD_NOTIFY(IDC_DLGPB,BN_CLICKED,dlg2dx),
 EV_CHILD_NOTIFY(IDC_MODFLOWPB,BN_CLICKED,mod2dx),
 EV_CHILD_NOTIFY(IDC_ERDASPB,BN_CLICKED,lan2dx),
 EV_CHILD_NOTIFY(IDC_SHAPEPB,BN_CLICKED,shp2dx),
 EV_CHILD_NOTIFY(IDC_IMAGINEPB,BN_CLICKED,img2dx),
END_RESPONSE_TABLE;

TGis2dxDialog::TGis2dxDialog(TWindow* parent,TResId resId):

            TDialog(parent,resId)
{
//          text1 = new TStatic(this,IDC_TEXT1);
//          text2 = new TStatic(this,IDC_TEXT2);
 demdlg = new TDemDialog(this,IDD_DEMDLG, Dem);
 DLGdlg = new TDlgDialog(this,IDD_DLGDLG);
 moddlg = new TDemDialog(this,IDD_DEMDLG, Modflow);
 landlg = new TDemDialog(this,IDD_DEMDLG, Erdas);
 imgdlg = new TImgDialog(this,IDD_IMAGINEDLG);
 shpdlg = new TShpDialog(this,IDD_SHAPEDLG);
}


void TGis2dxDialog::Terminate()
{
 CloseWindow();
 frame->CloseWindow();
}

void TGis2dxDialog::dem2dx()
{
 demdlg->Execute();
}

void TGis2dxDialog::dlg2dx()
{
 DLGdlg->Execute();
}

void TGis2dxDialog::mod2dx()
{
 moddlg->Execute();
}

void TGis2dxDialog::lan2dx()
{
 landlg->Execute();
}
```

```
void TGis2dxDialog::img2dx()
{
 imgdlg->Execute();
}

void TGis2dxDialog::shp2dx()
{
 shpdlg->Execute();
}

// This procedure displays help in response to the help button
// being pressed in the main application window.
void TGis2dxDialog::HelpTop()
{
   char buf[600];
   buf[0] = '\0';

   strcat(buf,"Choose the type of file to convert:\n\n\n");
   strcat(buf,"  DEM                - Digital Elevation Model (ASCII)\n\n");
   strcat(buf,"  DLG                - Digital Line Graph (ASCII)\n\n");
   strcat(buf,"  ERDAS              - ERDAS LAN/GIS Raster File (Binary)\n\n");
   strcat(buf,"  MODFLOW            - Modflow Heads/Drawdown File (Binary)\n\n");
   strcat(buf,"  ARCSHAPE           - Arc Shapefiles (Binary)\n\n");
   strcat(buf,"  ERDAS/IMAGINE - Erdas Imagine files (Binary)");
   MessageBox(buf,0,MB_OK | MB_ICONINFORMATION);
}

int OwlMain(int argc, char* argv[])
{
 if (argc == 2)
   return TGis2dxApplication(argv[1]).Run();
 else
   return TGis2dxApplication("").Run();
}
```

```
#ifndef     CALLBACKS_H
#define  CALLBACKS_H

#include <fstream.h>
#include "gisowl.h"
#include "gisfiles.h"
#include "callutils.h"
#include "convW.rh"

#define MIN(a,b)  ((a) < (b) ? (a) : (b))
#define MAX(a,b)  ((a) > (b) ? (a) : (b))

class TDemDialog: public TDialog
{
 public:
            TDemDialog(TWindow* parent,TResId resId,int);
   void SetupWindow();
   void newOutfile();
   void newInfile();
   void newOpenDir()  {openbox->NewDir();}
   void newSaveDir()  {savebox->NewDir();}
   void newOpenFilter() {openbox->NewFilter();}
   void newSaveFilter() {savebox->NewFilter();}
 /* misc callbacks functions */
            void ToggleMeFile();
   char* between(int, int, int);
   void MakeVisible();
   void MakeInvisible();
   void OpenFile();
   void OpenFileDem(ifstream&);
   void OpenFileMOD(ifstream&);
   void OpenFileLAN(ifstream& in);
   void ResetFileOpenBox();
   void PlaceDefault();
   void ClearMinMax();
   void OkCallBack();
   void OtherRun(char* file1, char* file2, bool isFile);
   void HelpAll();
private:
   GisFileBox *openbox;
   GisFileBox *savebox;
   TEdit     *o_filter, *s_filter;
   TListBox  *o_dirbox, *s_dirbox;
   TListBox  *o_filebox, *s_filebox;
   TEdit     *o_select, *s_select;
   TButton   *o_filterPB,*s_filterPB;

   TStatic   *layerlbl;
   TListBox  *layerbox;
   TEdit     *minxW, *minyW, *maxxW, *maxyW;
   TCheckBox *filetb;
   TCheckBox *bintb;
   int       convType;
   char      fileExt[10];

 DECLARE_RESPONSE_TABLE(TDemDialog);
};

class TDlgDialog: public TDialog
{
 public:
            TDlgDialog(TWindow* parent,TResId resId);
   void SetupWindow();
   void newOutfile();
   void newInfile();
   void newOpenDir()  {openbox->NewDir();}
   void newSaveDir()  {savebox->NewDir();}
   void newOpenFilter() {openbox->NewFilter();}
   void newSaveFilter() {savebox->NewFilter();}
   void OpenFileDLG();
   void OkCallBackDLG();
   void DLGRun(char* file1, char* file2, bool isFile);
   void ResetFileOpenBoxDLG();
   void HelpDLG();
            void fileToggle();
 private:
   GisFileBox *openbox;
   GisFileBox *savebox;
   TEdit     *o_filter, *s_filter;
   TListBox  *o_dirbox, *s_dirbox;
   TListBox  *o_filebox, *s_filebox;
   TEdit     *o_select, *s_select;
   TButton   *o_filterPB,*s_filterPB;

   TStatic   *stattxt;
   TCheckBox *filetb;
   TCheckBox *bintb;
   char      fileExt[10];
```

```
  DECLARE_RESPONSE_TABLE(TDlgDialog);
};

#endif
```

```
// DEM, ERDAS, MODFLOW Converters
//////////////////////////////////////////////////////////////////////
//   File:          callbacks.cpp
//   Author:        David Thompson, Dick Thompson, Petr Votava,
//                  Kirk Moeller, and others.
//   Description:
//                  Implementation of all of the callbacks
//                  functions necessary to implement all of the
//                  operations of the dialogs associated with the
//                  converters list above.
//////////////////////////////////////////////////////////////////////

#include <stdio.h>
#include "callbacks.h"

#include "dem2dx.h"
#include "mod2dx.h"
#include "e2dx.h"

#include "callutils.h"
/*#include "globals.h"*/

/* converter globals */
int BANDS = 0;

DEFINE_RESPONSE_TABLE1(TDemDialog,TDialog)
  EV_CHILD_NOTIFY(IDC_LFPB, BN_CLICKED, newOpenFilter),
  EV_CHILD_NOTIFY(IDC_LDIRBOX, LBN_DBLCLK, newOpenDir),
  EV_CHILD_NOTIFY(IDC_RFPB, BN_CLICKED, newSaveFilter),
  EV_CHILD_NOTIFY(IDC_RDIRBOX, LBN_DBLCLK, newSaveDir),
  EV_CHILD_NOTIFY(IDC_FILETB, BN_CLICKED, ToggleMeFile),
  /* misc callbacks */
  EV_CHILD_NOTIFY(IDC_LFILEBOX, LBN_SELCHANGE, OpenFile),
  EV_CHILD_NOTIFY(IDC_RFILEBOX, LBN_SELCHANGE, newOutfile),
  EV_CHILD_NOTIFY(IDOK, BN_CLICKED, OkCallBack),
  EV_CHILD_NOTIFY(IDHELP, BN_CLICKED, HelpAll),
  EV_CHILD_NOTIFY(IDC_DEFPB, BN_CLICKED, PlaceDefault),
END_RESPONSE_TABLE;

TDemDialog::TDemDialog(TWindow* parent,TResId resId,int type): TDialog(parent,resId)
{
  /* open box controls */
  o_filter      = new TEdit(this,IDC_LFILTER);
  o_dirbox      = new TListBox(this,IDC_LDIRBOX);
  o_filebox     = new TListBox(this,IDC_LFILEBOX);
  o_select      = new TEdit(this,IDC_LSELECT);
  o_filterPB    = new TButton(this,IDC_LFPB);

  /* save box controls */
  s_filter      = new TEdit(this,IDC_RFILTER);
  s_dirbox      = new TListBox(this,IDC_RDIRBOX);
  s_filebox     = new TListBox(this,IDC_RFILEBOX);
  s_select      = new TEdit(this,IDC_RSELECT);
  s_filterPB    = new TButton(this,IDC_RFPB);

  layerlbl   = new TStatic(this, IDC_LAYERSLBL);
  layerbox   = new TListBox(this, IDC_LAYERSSL);
  minxW      = new TEdit(this, IDC_MINXTXT);
  minyW      = new TEdit(this, IDC_MINYTXT);
  maxxW      = new TEdit(this, IDC_MAXXTXT);
  maxyW      = new TEdit(this, IDC_MAXYTXT);
  filetb     = new TCheckBox(this,IDC_FILETB);
  bintb      = new TCheckBox(this,IDC_BINTB);
  convType = type;

  switch (convType)
  {
   case Dem:    SetCaption("Dem Converter");
                strcpy(fileExt,"d:\\*.dem");
                break;
   case Modflow: SetCaption("Modflow Converter");
                strcpy(fileExt,"d:\\*.*");
                break;
   case Erdas:  SetCaption("Erdas Converter");
                strcpy(fileExt,"d:\\*.lan");
  }
}

void TDemDialog::SetupWindow()
{
  TDialog::SetupWindow();

  if (convType == Erdas)
    MakeVisible();
  else
    MakeInvisible();

  filetb->Check();
  bintb->Check();
```

```
 layerbox->ClearList();
 ClearMinMax();

 fileState(false);
 setGisType(convType);

 /* this stuff can't be done until after has been created */
 openbox = new GisFileBox(this,o_filter,o_dirbox,o_filebox,o_select,
                o_filterPB, IDC_LDIRBOX, IDC_LFILEBOX,
                fileExt);
 savebox = new GisFileBox(this,s_filter,s_dirbox,s_filebox,s_select,
                s_filterPB,IDC_RDIRBOX, IDC_RFILEBOX,
                "d:\\*.dx");
}

void TDemDialog::newOutfile()
{
 char path[256], file[256];

 savebox->GetPath(path);
 strcat(path,"\\");
 s_filebox->GetSelString(file,256);
 strcat(path,file);
 s_select->SetText(path);
}

void TDemDialog::newInfile()
{
 char path[256], file[256];

 openbox->GetPath(path);
 strcat(path,"\\");
 o_filebox->GetSelString(file,256);
 strcat(path,file);
 o_select->SetText(path);
}

char* TDemDialog::between(int i, int j, int ROW)
{
 static char buf[256];
 if(ROW)
    sprintf(buf, "%d - %d", MAX(0, MIN(i,j)), MIN(getWidth(), MAX(i,j)));
 else
    sprintf(buf, "%d - %d", MAX(0, MIN(i,j)), MIN(getHeight(), MAX(i,j)));
 return buf;
}

void TDemDialog::MakeVisible()
{
// layerlbl->EnableWindow(true);
// layerbox->EnableWindow(true);
 layerlbl->Show(true);
 layerbox->Show(true);
}

void TDemDialog::MakeInvisible()
{
// layerlbl->EnableWindow(false);
// layerbox->EnableWindow(false);
 layerlbl->Show(false);
 layerbox->Show(false);
}

#if 0
void PopupDialogMisc(Widget *wa)
{
 if(PERSISTENT || FILTER)
 {
  XtUnmanageChild(wa[WI_RFORM_DEM]);
  XmToggleButtonSetState(wa[WI_FILESAVETB],False,False);
 }
}
#endif


// This is the open file function.  It open files for all four converters.
// Those four converters are: DEM, DLG, ERDAS, MODFLOW.
void TDemDialog::OpenFile()
{
 char filename[256];
 ifstream in;

 newInfile();
 o_select->GetText(filename,256);

 if (gisType() == Erdas || gisType() == Modflow)
  in.open(filename,ios::binary);
 else
  in.open(filename);
```

```
 if(!in)
 {
    in.close();
    ResetFileOpenBox();

    MessageBox("Error: Can not read input file!",0,MB_OK | MB_ICONERROR);
    return;
 }

 fileState(true);
 if(gisType() == Dem)
    OpenFileDem(in);
 else if(gisType() == Modflow)
    OpenFileMOD(in);
 else if(gisType() == Erdas)
    OpenFileLAN(in);
}

void TDemDialog::OpenFileDem(ifstream& in)
{
 int WIDTH, HEIGHT;

 if(!get_DEM_info(in, WIDTH, HEIGHT))
 {
    in.close();
    setWidth(WIDTH);
    setHeight(HEIGHT);
    PlaceDefault();
 }
 else
 {
    in.close();
    ResetFileOpenBox();
    ClearMinMax();

    MessageBox("Error: Not a DEM file!",0,MB_OK | MB_ICONERROR);
 }
}

void TDemDialog::OpenFileMOD(ifstream& in)
{
 int WIDTH, HEIGHT;

 if(!get_MODFLOW_info(in, WIDTH, HEIGHT))
 {
    in.close();
    setWidth(WIDTH);
    setHeight(HEIGHT);
    PlaceDefault();
 }
 else
 {
    in.close();
    ResetFileOpenBox();
    ClearMinMax();

    MessageBox("Error: Not a MODFLOW file!",0, MB_OK | MB_ICONERROR);
 }
}

void TDemDialog::OpenFileLAN(ifstream& in)
{
 char buf[20];
 int WIDTH, HEIGHT;

 layerbox->ClearList();
 if(!get_ERDAS_info(in, WIDTH, HEIGHT, BANDS))
 {
    in.close();
    setWidth(WIDTH);
    setHeight(HEIGHT);
    PlaceDefault();

    int i=0;
    for(i = 0; i < BANDS; i++)
    {
     sprintf(buf, "Band %d", i+1);
     layerbox->AddString(buf);
     layerbox->SetSel(i,true);
    }
 }
 else
 {
    in.close();
    ResetFileOpenBox();
    ClearMinMax();

    MessageBox("Error: Not an ERDAS file!",0,MB_OK | MB_ICONERROR);
 }
}
```

```
void TDemDialog::ResetFileOpenBox()
{
 fileState(false);
 layerbox->SetSel(-1,false);
}

void TDemDialog::PlaceDefault()
{
  minxW->SetText("0");
  minyW->SetText("0");
  maxxW->SetText(itoa( getWidth() ));
  maxyW->SetText(itoa( getHeight() ));
}

void TDemDialog::ClearMinMax()
{
  minxW->SetText("");
  minyW->SetText("");
  maxxW->SetText("");
  maxyW->SetText("");
}

// This is used by all four converters (DEM, DLG, ERDAS, MODFLOW).
// This is where the dx info get generated.
void TDemDialog::OkCallBack()
{
 char file1[256], file2[256], tmp[256];
 bool isFile;
 char buf[500];

 o_select->GetText(file1,256);
 s_select->GetText(file2,256);

 isFile = filetb->GetCheck() == BF_CHECKED;
/** Check out the output file name (if using file output).
 ** if it exists?, do we overwrite.
 ** add a .dx extension if there is no extension present.
**/
 if (!fileOpen())
 {
  MessageBox("No input file selected!",0,MB_OK | MB_ICONERROR);
  return;
 }
 else if (!FileExists(file1))
 {
  MessageBox("Input file does not exist or is unreadable!",0,MB_OK | MB_ICONERROR);
  return;
 }

 if (isFile)
 {
   if (!validFile(file2,s_select))
   {
    strcpy(buf,"No output file selected or path is incorrect.\n");
    strcat(buf,"The full path must be displayed in the selection box!\n");
    strcat(buf,"  -- Pressing the filter button will display the current path.");
    MessageBox(buf,0,MB_OK | MB_ICONERROR);
    return;
   }
   else if (FileExists(file2) && (!overwrite(this)))
     return;
 }
 // call the conversion
 OtherRun(file1, file2, isFile);
}

void TDemDialog::OtherRun(char* file1, char* file2, bool isFile)
{
 char xy[4][10];
 int sx, sy, ex, ey, binary;
 bool isBinary;

 SetCursor(0,IDC_WAIT);

 isBinary = bintb->GetCheck() == BF_CHECKED;

 ifstream in;

 if (gisType() == Erdas || gisType() == Modflow)
  in.open(file1,ios::binary);
 else
  in.open(file1);

 if(!in)
 {
   MessageBox("Error: Can not read input file!",0,MB_OK | MB_ICONERROR);
   SetCursor(0, IDC_ARROW);
   return;
 }
 ofstream out;
```

```
if(!isFile)
   out.attach(1);
else if (isBinary)
   out.open(file2,ios::binary);
else
   out.open(file2);

if(!out)
{
   in.close();
   MessageBox("Error: Can not write output file!",0,MB_OK | MB_ICONERROR);
   SetCursor(0, IDC_ARROW);
   return;
}
binary = (isBinary) ? 1 : 0;

minxW->GetText(xy[0],10);
minyW->GetText(xy[1],10);
maxxW->GetText(xy[2],10);
maxyW->GetText(xy[3],10);

sx = atoi(xy[0]);
ex = atoi(xy[2]);
sy = atoi(xy[1]);
ey = atoi(xy[3]);
if(gisType() == Dem)
{
   DEM test(in, out, sx, ex, sy, ey, binary);
}
else if(gisType() == Erdas)
{
   int bands[10], numbds;

   numbds = layerbox->GetSelIndexes(bands,10);
   for(int k=0;k<numbds;bands[k++]++);

   if(numbds)
   {
    ERDAS test(in, out, sx, ex, sy, ey, binary, numbds, bands);
   }
   else
   {
    MessageBox("Error: No Bands Selected to Convert!",0,MB_OK | MB_ICONERROR);
    SetCursor(0, IDC_ARROW);
    return;
   }
 }
 else if(gisType() == Modflow)
    MODFLOW(in, out, binary);

 out.flush();

 in.close();
 if (!isFile)
  /*out.detach()*/ ;
 else
  out.close();

/* if(PERSISTENT) Quit(WA[0], NULL, NULL);*/
 SetCursor(0,IDC_ARROW);
}

// Hide or Show the file save box as is appropriate.
// Note: the button state has already changed when this
//  function is reached.
void TDemDialog::ToggleMeFile()
{
 bool checked;

 checked = (filetb->GetCheck() == BF_CHECKED);
 savebox->EnableBox(checked);
}

// Displays help for the following converter dialogs:
// DEM, ERDAS, MODFLOW, ARC.
void TDemDialog::HelpAll()
{
 char buf[2000];
 int ConvType;

 ConvType = gisType();

 buf[0] = '\0';
 switch(ConvType)
 {
  case Dem:  strcpy(buf,"DEM ");  break;

  case Erdas:  strcpy(buf,"LAN ");  break;

  case Modflow:  strcpy(buf,"MOD ");  break;
```

```
}

  strcat(buf,"Conversion Help\n\n");
  strcat(buf,"  Left File Selection Box -- Used to choose the input file for the conversion.\n");
  strcat(buf,"     OK Button  -- Used to start the conversion.\n");
  strcat(buf,"     Filter     -- Used to filter files in directory.\n");
  strcat(buf,"     Quit       -- Quit the current conversion session.\n");
  strcat(buf,"     Help       -- Pop up this help.\n\n");

  strcat(buf,"  Stats and Options         -- Input file information and user options\n");
  strcat(buf,"     Min X, Min Y, Max X, Max Y -- Grid size information about the DEM input
file.\n");
  strcat(buf,"     Default    -- Allow user to recover input file default information if changes have
been made.\n");


 if (ConvType != Modflow)
  strcat(buf,"     Clip       -- Allow user to only convert a portion of the input file.\n");

 if(ConvType == Erdas)
  strcat(buf,"     Layers List -- Allow user to select which bands of LAN/GIS input file to
convert.\n");

  strcat(buf,"\n\n");
  strcat(buf,"     Binary OutputToggle  -- Allow the user to choose between ASCII or Binary data
format in output DX file.\n");
  strcat(buf,"     File Output Toggle   -- Allow the user to choose between file or standard output
for the conversion.\n\n");

  strcat(buf,"  Right File Selection Box (only visible if file output is selected) -- Used to choose
the output file for the conversion.\n");
  strcat(buf,"     Filter -- Used to filter files in directory.\n\n");


  MessageBox(buf,0,MB_OK | MB_ICONINFORMATION);
}
```

```
// DLG Converters
//////////////////////////////////////////////////////////////////////
//   File:          callbacksDlg.cpp
//   Author:        David Thompson, Dick Thompson, Petr Votava,
//                  Kirk Moeller, and others.
//   Description:
//                  Implementation of all of the callbacks
//                  functions necessary to implement all of the
//                  operations of the dialogs associated with the
//                  converters list above.
//////////////////////////////////////////////////////////////////////

#include <iostream.h>

#include "callbacks.h"
#include "dlg2dx.h"
#include "callutils.h"
/*#include "globals.h"*/

DEFINE_RESPONSE_TABLE1(TDlgDialog,TDialog)
 EV_CHILD_NOTIFY(IDCANCEL, BN_CLICKED, CmCancel),
 EV_CHILD_NOTIFY(IDC_DLG_LFPB, BN_CLICKED, newOpenFilter),
 EV_CHILD_NOTIFY(IDC_DLG_LDIRBOX, LBN_DBLCLK, newOpenDir),
 EV_CHILD_NOTIFY(IDC_DLG_RFPB, BN_CLICKED, newSaveFilter),
 EV_CHILD_NOTIFY(IDC_DLG_RDIRBOX, LBN_DBLCLK, newSaveDir),
 EV_CHILD_NOTIFY(IDC_DLG_LFILEBOX, LBN_SELCHANGE, OpenFileDLG),
 EV_CHILD_NOTIFY(IDC_DLG_RFILEBOX, LBN_SELCHANGE, newOutfile),
 EV_CHILD_NOTIFY(IDOK, BN_CLICKED, OkCallBackDLG),
 EV_CHILD_NOTIFY(IDHELP, BN_CLICKED, HelpDLG),
 EV_CHILD_NOTIFY(IDC_DLG_FILETB, BN_CLICKED, fileToggle),
END_RESPONSE_TABLE;

TDlgDialog::TDlgDialog(TWindow* parent,TResId resId): TDialog(parent,resId)
{
 /* open box controls */
 o_filter     = new TEdit(this,IDC_DLG_LFILTER);
 o_dirbox     = new TListBox(this,IDC_DLG_LDIRBOX);
 o_filebox    = new TListBox(this,IDC_DLG_LFILEBOX);
 o_select     = new TEdit(this,IDC_DLG_LSELECT);
 o_filterPB   = new TButton(this,IDC_DLG_LFPB);

 /* save box controls */
 s_filter     = new TEdit(this,IDC_DLG_RFILTER);
 s_dirbox     = new TListBox(this,IDC_DLG_RDIRBOX);
 s_filebox    = new TListBox(this,IDC_DLG_RFILEBOX);
 s_select     = new TEdit(this,IDC_DLG_RSELECT);
 s_filterPB   = new TButton(this,IDC_DLG_RFPB);

 stattxt   = new TStatic(this, IDC_DLG_STATTXT);
 filetb    = new TCheckBox(this,IDC_DLG_FILETB);
 bintb     = new TCheckBox(this,IDC_DLG_BINTB);
}

void TDlgDialog::SetupWindow()
{
 TDialog::SetupWindow();
 filetb->Check();
 bintb->Check();

  /* this stuff can't be done until after has been created */
 openbox = new GisFileBox(this,o_filter,o_dirbox,o_filebox,o_select,
                  o_filterPB, IDC_DLG_LDIRBOX, IDC_DLG_LFILEBOX,
                  "d:\\*.dlg");
 savebox = new GisFileBox(this,s_filter,s_dirbox,s_filebox,s_select,
                  s_filterPB,IDC_DLG_RDIRBOX, IDC_DLG_RFILEBOX,
                  "d:\\*.dx");
 fileState(false);
 setGisType(Dlg);

/*  if(PERSISTENT || FILTER)
 {
   XtUnmanageChild(wa[WI_RFORM_DLG]);
   XmToggleButtonSetState(wa[WI_FILESAVETB1],False,False);
 }
*/
 stattxt->SetText("");
}

void TDlgDialog::newOutfile()
{
 char path[256], file[256];

 savebox->GetPath(path);
 strcat(path,"\\");
 s_filebox->GetSelString(file,256);
 strcat(path,file);
 s_select->SetText(path);
}

void TDlgDialog::newInfile()
```

```
{
 char path[256], file[256];

 openbox->GetPath(path);
 strcat(path,"\\");
 o_filebox->GetSelString(file,256);
 strcat(path,file);
 o_select->SetText(path);
}

void TDlgDialog::OpenFileDLG()
{
 int nodes, lines, areas, areainfo, lineinfo;
 char name[41], type[21], buf[150], temp[15];
 char filename[256];

 newInfile();
 o_select->GetText(filename,256);

 ifstream in(filename);
 if(!in)
 {
    in.close();
    ResetFileOpenBoxDLG();

    MessageBox("Error: Can not read input file!",0,MB_OK | MB_ICONERROR);
    return;
 }

 fileState(true);

 if(!get_DLG_info(in, name, type, nodes, lines, areas, areainfo, lineinfo))
 {
    in.close();
    (areainfo)? sprintf(temp, "present") : sprintf(temp, "not present");
    sprintf(buf, "Area name:\n%s\n\nMap type:\n%s\n\n Nodes: %d\n Lines: %d\n Areas: %d\n\n
Polygon info: %s",
          name, type, nodes, lines, areas, temp);

    stattxt->SetText(buf);
 }
 else
 {
    ResetFileOpenBoxDLG();
    in.close();
    buf[0] = 0;
    stattxt->SetText(buf);
    MessageBox("Error: Not a DLG file!",0,MB_OK | MB_ICONERROR);
 }
}

void TDlgDialog::OkCallBackDLG()
{
 char   file1[256], file2[256];
 bool  isFile;


 o_select->GetText(file1,256);
 s_select->GetText(file2,256);

 isFile = filetb->GetCheck() == BF_CHECKED;
/** Check out the output file name (if using file output).
 ** if it exists?, do we overwrite.
 ** add a .dx extension if there is no extension present.
**/
 if (!fileOpen())
 {
  MessageBox("No input file selected!",0,MB_OK | MB_ICONERROR);
  return;
 }
 else if (!FileExists(file1))
 {
  MessageBox("Input file does not exist or is unreadable!",0,MB_OK | MB_ICONERROR);
  return;
 }

 if (isFile)
 {
    if (!validFile(file2,s_select))
    {
     MessageBox("NO OUTPUT FILE SELECTED!",0,MB_OK | MB_ICONERROR);
     return;
    }
    else if (FileExists(file2) && (!overwrite(this)))
      return;
 }
 // call the conversion
 DLGRun(file1, file2, isFile);
}
```

```
void TDlgDialog::DLGRun(char* file1, char* file2, bool isFile)
{
 int binary;
 bool isBinary;

 SetCursor(0,IDC_WAIT);

 isBinary = bintb->GetCheck() == BF_CHECKED;

 ifstream in(file1);
 if(!in)
 {
    MessageBox("Error: Can not read input file!",0,MB_OK | MB_ICONERROR);
    return;
 }

 ofstream out;
 if(!isFile)
    out.attach(1);
 else if (isBinary)
    out.open(file2,ios::binary);
 else
    out.open(file2);

 if(!out)
 {
    in.close();
    MessageBox("Error: Can not write output file!",0,MB_OK | MB_ICONERROR);
    return;
 }
 binary = (isBinary) ? 1 : 0;
 DLG test(in, out, binary);
 out.flush();

 in.close();
 if (!isFile)
  /* out.detach();*/
  ;
 else
  out.close();

 SetCursor(0,IDC_ARROW);
}

// Hide or Show the file save box as is appropriate.
// Note: the button state has already changed when this
//  function is reached.
void TDlgDialog::fileToggle()
{
 bool checked;

 checked = (filetb->GetCheck() == BF_CHECKED);
 savebox->EnableBox(checked);
}


void TDlgDialog::ResetFileOpenBoxDLG()
{
 int item;

 fileState(false);

 item = o_filebox->GetSelIndex();
 o_filebox->SetSel(item,false);
}


void TDlgDialog::HelpDLG()
{
 char buf[2000];

 buf[0] = '\0';
 strcat(buf,"Dlg Conversion Help\n\n");
 strcat(buf,"   Left File Selection Box -- Used to choose the input file for the conversion.\n");
 strcat(buf,"     OK Button  -- Used to start the conversion.\n");
 strcat(buf,"     Filter     -- Used to filter files in directory.\n");
 strcat(buf,"     Quit       -- Quit the current conversion session.\n");
 strcat(buf,"     Help       -- Pop up this help.\n\n");

 strcat(buf,"   Stats and Options      -- Input file information and user options\n");

 strcat(buf,"     Statistics -- Node, line, and area information about the input DLG file.\n");
 strcat(buf,"     Default    -- Allow user to recover input file default information if changes have
been made.\n");

 strcat(buf,"\n\n");
 strcat(buf,"     Binary OutputToggle  -- Allow the user to choose between ASCII or Binary data
format in output DX file.\n");
 strcat(buf,"     File Output Toggle   -- Allow the user to choose between file or standard output
for the conversion.\n\n");
```

```
  strcat(buf,"   Right File Selection Box (only visible if file output is selected) -- Used to choose the
output file for the conversion.\n");
  strcat(buf,"      Filter -- Used to filter files in directory.\n\n");

  MessageBox(buf,0,MB_OK | MB_ICONINFORMATION);
}
```

```
#ifndef SHP_CALLBACKS_H
#define SHP_CALLBACKS_H

#include <dos.h>
#include <dir.h>
#include "gisowl.h"
#include "shp2dxW.rh"
#include "seriesW.rh"
#include "seriesCall.h"
#include "Xshp2dx.h"
#include "gisfiles.h"


class TShpDialog: public TDialog
{
 public:
        TShpDialog(TWindow* parent,TResId resId);
  ~TShpDialog();
        void SetupWindow();
  void PopupDialogSeries();

 /******/
  void newOutfile();
  void newInfile();
        void NewOpenDir() {openbox->NewDir();}
        void NewSaveDir() {savebox->NewDir();}
  void NewOpenFilter()  {openbox->NewFilter();}
  void NewSaveFilter() {savebox->NewFilter();}
 /******/

        void LinesToggle();
        void PolylinesToggle();
        void OpenFileShape();
  void updateDialog();
  void ShapeOkCallBack();
  int  GetSelectedNames(FIELD selected[]);
  void fileToggle();
  void HelpShape();
  void SeriesToggle_Shape();
 private:
  GisFileBox *openbox;
  GisFileBox *savebox;
  TEdit     *o_filter, *s_filter;
  TListBox   *o_dirbox, *s_dirbox;
  TListBox   *o_filebox, *s_filebox;
  TEdit     *o_select, *s_select;
  TButton    *o_filterPB,*s_filterPB;

  TStatic     *attriblbl;
        TListBox    *attriblst;
        TRadioButton *linestb;
        TRadioButton *polylinestb;
  TCheckBox   *seriestb;
  TButton     *attribpb;
  TCheckBox   *filetb;
  TCheckBox   *bintb;

  TSeriesDialog *seriesdlg;

  bool NEWLIST;

  DECLARE_RESPONSE_TABLE(TShpDialog);
};

#endif
```

```
// SHAPE Converter
//////////////////////////////////////////////////////////////////////
//   File:          shp_callbacks.cpp
//   Author:        Kirk A. Moeller
//   Description:
//                  Implementation of the callbacks functions
//                  which respond to user actions in the
//                  shape converter dialog.
//////////////////////////////////////////////////////////////////////

#include <fstream.h>
#include <string.h>
#include "shp_callbacks.h"
/*#include "globals.h"*/
#include "seriesCall.h"
#include "callutils.h"

/* shape converter globals */
 char          *outfile;
 char          shpfile[200];
 char          dbffile[200];
 short             len[NumFields];
 FIELD             fields[NumFields];
 int                 NumOfRecords;
 short             DataOffset;
 int                 numfields;
 extern SHAPETYPE   shape;   // DEFINED IN Xshp2dx.C

// This procedure registers the calllbacks for the widget in
// the shape converter dialog box.


DEFINE_RESPONSE_TABLE1(TShpDialog,TDialog)
 EV_CHILD_NOTIFY(IDCANCEL, BN_CLICKED, CmCancel),
 EV_CHILD_NOTIFY(IDC_SHAPE_LFPB, BN_CLICKED, NewOpenFilter),
 EV_CHILD_NOTIFY(IDC_SHAPE_LDIRBOX, LBN_DBLCLK, NewOpenDir),
 EV_CHILD_NOTIFY(IDC_SHAPE_RFPB, BN_CLICKED, NewSaveFilter),
 EV_CHILD_NOTIFY(IDC_SHAPE_RDIRBOX, LBN_DBLCLK, NewSaveDir),
 EV_CHILD_NOTIFY(IDC_SHAPE_LFILEBOX,LBN_SELCHANGE,OpenFileShape),
 EV_CHILD_NOTIFY(IDC_SHAPE_RFILEBOX,LBN_SELCHANGE,newOutfile),
 EV_CHILD_NOTIFY(IDC_SHAPE_LINESRB,BN_CLICKED,LinesToggle),
 EV_CHILD_NOTIFY(IDC_SHAPE_POLYRB,BN_CLICKED,PolylinesToggle),
 EV_CHILD_NOTIFY(IDC_SHAPE_FILETB,BN_CLICKED,fileToggle),
 EV_CHILD_NOTIFY(IDOK,BN_CLICKED,ShapeOkCallBack),
 EV_CHILD_NOTIFY(IDHELP, BN_CLICKED, HelpShape),
 EV_CHILD_NOTIFY(IDC_SHAPE_ATTRIBPB, BN_CLICKED, PopupDialogSeries),
 EV_CHILD_NOTIFY(IDC_SHAPE_SERIESTB, BN_CLICKED, SeriesToggle_Shape),
END_RESPONSE_TABLE;

TShpDialog::TShpDialog(TWindow* parent,TResId resId): TDialog(parent,resId)
{
 /* open box controls */
 o_filter      = new TEdit(this,IDC_SHAPE_LFILTER);
 o_dirbox      = new TListBox(this,IDC_SHAPE_LDIRBOX);
 o_filebox     = new TListBox(this,IDC_SHAPE_LFILEBOX);
 o_select      = new TEdit(this,IDC_SHAPE_LSELECT);
 o_filterPB    = new TButton(this,IDC_SHAPE_LFPB);

 /* save box controls */
 s_filter      = new TEdit(this,IDC_SHAPE_RFILTER);
 s_dirbox      = new TListBox(this,IDC_SHAPE_RDIRBOX);
 s_filebox     = new TListBox(this,IDC_SHAPE_RFILEBOX);
 s_select      = new TEdit(this,IDC_SHAPE_RSELECT);
 s_filterPB    = new TButton(this,IDC_SHAPE_RFPB);

 attriblbl     = new TStatic(this,IDC_SHAPE_ATTLBL);
 attriblst     = new TListBox(this,IDC_SHAPE_ATTBOX);
 linestb       = new TRadioButton(this,IDC_SHAPE_LINESRB);
 polylinestb   = new TRadioButton(this,IDC_SHAPE_POLYRB);
 seriestb      = new TCheckBox(this, IDC_SHAPE_SERIESTB);
 attribpb      = new TButton(this, IDC_SHAPE_ATTRIBPB);
 filetb        = new TCheckBox(this,IDC_SHAPE_FILETB);
 bintb         = new TCheckBox(this,IDC_SHAPE_BINTB);

 /* series dialog box */
 seriesdlg = new TSeriesDialog(this,IDD_SERIESDLG);
 NEWLIST = true;

}

TShpDialog::~TShpDialog()
{
}

void TShpDialog::SetupWindow()
{
 TDialog::SetupWindow();
 linestb->Check();
 linestb->EnableWindow(false);
 polylinestb->EnableWindow(false);
```

```
 filetb->Check();
 bintb->Check();

  /* this stuff can't be done until after has been created */
 openbox = new GisFileBox(this,o_filter,o_dirbox,o_filebox,o_select,
                  o_filterPB, IDC_SHAPE_LDIRBOX, IDC_SHAPE_LFILEBOX,
                  "d:\\*.shp");
 savebox = new GisFileBox(this,s_filter,s_dirbox,s_filebox,s_select,
                  s_filterPB,IDC_SHAPE_RDIRBOX, IDC_SHAPE_RFILEBOX,
                  "d:\\*.dx");

// Change dialog to standard out mode if these conditions are true.
/*  if (PERSISTENT || FILTER)
  {
   XtUnmanageChild(wa[WI_RFORM_SHAPE]);
   XmToggleButtonSetState(wa[WI_FILETB_SHAPE],False,False);
  }
*/
 /** Time series Option is initially NO! **/
 seriestb->Uncheck();
 SeriesToggle_Shape();

 attriblst->ClearList();

 attriblbl->SetText("Attributes:");

 fileState(false);
}

void TShpDialog::newOutfile()
{
 char path[256], file[256];

 savebox->GetPath(path);
 strcat(path,"\\");
 s_filebox->GetSelString(file,256);
 strcat(path,file);
 s_select->SetText(path);
}

void TShpDialog::newInfile()
{
 char path[256], file[256];

 openbox->GetPath(path);
 strcat(path,"\\");
 o_filebox->GetSelString(file,256);
 strcat(path,file);
 o_select->SetText(path);
}

void TShpDialog::PopupDialogSeries()
{
 int nAttribs;
 char* Attribs[50];
 int i;

 for(i=0;i<50;Attribs[i++] = new char[SizeOfName]);

 if (!fileOpen())
 {
  MessageBox("Please Open a Shape file first.",0, MB_OK | MB_ICONERROR);
  return;
 }

 if (NEWLIST)
 {
  nAttribs = attriblst->GetCount();
  for(int i=0;i<nAttribs;i++)
   attriblst->GetString(Attribs[i],i);

  seriesdlg->setAttribList(Attribs,nAttribs);
  for(i=0;i<SizeOfName;delete [] Attribs[i++]);
 }
 seriesdlg->Execute();

 if (NEWLIST) NEWLIST = false;
}

// These two functions toggle wether lines or polyline info
// is generated.
void TShpDialog::LinesToggle()
{
 SetLinesValue(true);
}



void TShpDialog::PolylinesToggle()
{
```

```
 SetLinesValue(false);
}


// Open the input shape file.
// In this case that consists of reading the dbase file and
// getting the attribute names.
// These names are then displayed in the list box.
void TShpDialog::OpenFileShape()
{
 char filename[256];
 int i;
 char *pstr;
 ifstream fin, fdbf;

 // desensitize the polylines/lines radio box.
 linestb->EnableWindow(false);
 polylinestb->EnableWindow(false);

//*** INITIALIZE GLOBALS ***
 NumOfRecords =0;
 DataOffset=0;
 numfields=0;

// Make sure series dialog lists are empty!
// Set a flag that will cause the dialog to reset
// when it is popped up.
// This flag is defined in seriesCall.C
 NEWLIST = true;

 SetCursor(0,IDC_WAIT);

//*** GET INPUT FILE NAME ****
 newInfile();
 o_select->GetText(filename,256);

 shpfile[0] = '\0';
 strcpy(shpfile,filename);

//*** MAKE FILE NAME ***
 strcpy(dbffile,shpfile);
 pstr = strstr(dbffile,".shp");
 *pstr = '\0';
 strcat(dbffile,".dbf");
```

```
//**************************
//*** CHECK INPUT FILE ***
 fdbf.open(dbffile,ios::binary);
 fin.open(shpfile,ios::binary);
 if ((!fin) || (!fdbf))
 {
  fin.close();
  fdbf.close();

  o_filebox->SetSel(-1,false);  /* deselect all */

  MessageBox("Can't read input file!",0,MB_OK | MB_ICONERROR);
  return;
 }
 GetFileInfo(fin);
 fin.close();

 updateDialog();
 attriblst->ClearList();

//*** GET FIELD NAMES FROM DBASE FILE ****

 numfields = ProcessDbfHead(fdbf,len,fields,NumOfRecords,DataOffset);

 for(i = 0; i < numfields; i++)
    attriblst->AddString(fields[i].name);

 SetCursor(0,IDC_ARROW);
 fileState(true);
}

void TShpDialog::updateDialog()
{
 SetLinesValue(false);
 // Change the shape file type label to the appropriate value.
 if (shape == arc)
 {
  //This is an arc shape file, so we want these toggles operational.
  linestb->EnableWindow(true);
  polylinestb->EnableWindow(true);

  attriblbl->SetText("Attributes: ARC");
  SetLinesValue(true);
 }
 else if (shape == polygon)
```

```
    attriblbl->SetText("Attributes: POLYGON");
  else if (shape == multipoint)
    attriblbl->SetText("Attributes: MULTIPOINT");
  else
    attriblbl->SetText("Attributes: POINT");
}


// Do the convertion.
void TShpDialog::ShapeOkCallBack()
{
  int                   n=0;
  FIELD       selected[NumFields];
  RECORD*               Shapes;
  MULTIPOINT*           Multipt;
  ifstream    fin, fdbf;
  ofstream    fout;
  bool        isFile, isBinary;
  bool        doSeries;
  char          outfile[200];

  Shapes  = NULL;
  Multipt = NULL;

  doSeries = seriestb->GetCheck() == BF_CHECKED;
  SetdoSeriesValue(doSeries);
//*** Some of the variable/constants are defined in Xshp2dx.h ***
  SetCursor(0,IDC_WAIT);

//*** Make sure we have input and output selected ***

  if(!fileOpen())
  {
    MessageBox("Error: NO INPUT SELECTED!",0,MB_OK | MB_ICONERROR);
    SetCursor(0, IDC_ARROW);
    return;
  }

  isFile = filetb->GetCheck() == BF_CHECKED;
  isBinary = bintb->GetCheck() == BF_CHECKED;

  if(!isFile)
    fout.attach(1);     // attach fout to fd 1, which is stdout.
  else if (!validFile(outfile,s_select))
  {
    MessageBox("NO OUTPUT FILE SELECTED!",0,MB_OK | MB_ICONERROR);
    SetCursor(0, IDC_ARROW);
    return;
  }
  else if (FileExists(outfile) && (!overwrite(this)))
  {
    SetCursor(0,IDC_ARROW);
    return;
  }
  else if (isBinary)
    fout.open(outfile,ios::binary);
  else
    fout.open(outfile);

//*****************************************************
  for(n = 0; n < NumFields; n++)
  {
   selected[n].name = new char[SizeOfName];
   selected[n].name[0] = '\0';
  }

  // Get the selected attribute names from the list box
  // or from series dialog list, whichever is appropriate.
  if (GetSelectedNames(selected) == 0)
  {
   SetCursor(0, IDC_ARROW);
   return;
  }

  // Add ios::binary in these open statements if using on a pc.
  fin.open(shpfile,ios::binary);
  fdbf.open(dbffile,ios::binary);

  SetFilterValue(!isFile);
  SetBinaryValue(isBinary);

  float timetags[50];
  int   count;
  seriesdlg->getTimeTags(timetags,count);
  setTimeTags(timetags,count);

  // call the shape converter.
  DoRecords(fin,fdbf,fout,NumOfRecords,numfields,DataOffset,len,Shapes,Multipt,
        fields,selected);
```

```
 attriblst->SetSel(-1,false);

 for(n = 0; n < NumFields; n++)
    delete [] selected[n].name;

 SetCursor(0,IDC_ARROW);
}

int TShpDialog::GetSelectedNames(FIELD selected[])
{
 int  nitems;
 int  i;
 bool Series;
 char* buf[50];

 for(int j=0;j<50;j++)
  buf[j] = new char[SizeOfName];

 Series = seriestb->GetState();
 if (!Series)
 {
  nitems = attriblst->GetSelCount();
  if (nitems == 0)
  {
   MessageBox("There are no attributes Selected!",0, MB_OK | MB_ICONERROR);
   return (0);
  }
  attriblst->GetSelStrings(buf,50,SizeOfName);
 }
 else
 {
  seriesdlg->getSeriesList(buf,nitems);
  if (nitems == 0)
  {
   MessageBox("There are no attributes in the time series list, there needs to be at least one to
generate a series",
           0, MB_OK | MB_ICONERROR);
    return (0);
  }
 }

 for(i=0;i<nitems;i++)
  strcpy(selected[i].name,buf[i]);

 for(int j=0;j<50;delete [] buf[j++]);

 return (1);
}

// Toggle the file button between file and stdout.
// Hide or show the file save box as is appropriate.
void TShpDialog::fileToggle()
{
 bool checked;

 checked = (filetb->GetCheck() == BF_CHECKED);
 savebox->EnableBox(checked);
}

// Create and display the help box for the shape converter.
void TShpDialog::HelpShape()
{
 char buf[5000];

 buf[0] = '\0';
 strcat(buf,"Left File Selection Box: used to choose input file.\n");
 strcat(buf,"   Ok      -- Start the conversion and writes out result.\n");
 strcat(buf,"   Filter  -- Used to filter files in directory.\n");
 strcat(buf,"   Quit    -- Quit the current converter session.\n");
 strcat(buf,"   Help    -- Popup this help message.\n\n");

 strcat(buf,"Options:  Input file information and conversion options.\n");
 strcat(buf,"   Attributes          -- List of selectable attributes (choose one or more.)\n");
 strcat(buf,"   Create Time Series   -- If this toggle button is checked it allows the Attribute
Dialog button to be pressed.\n");
 strcat(buf,"   Attribute Dialog ... -- Pressing this button will popup a dialog that allows selection
of attributes for output to the Time Series.\n\n");
 strcat(buf,"Dx Structure: (appears Grayed Out unless shape type is Arcs\n");
 strcat(buf,"   Lines       -- The output dx structure will use lines to represent Arcs.\n");
 strcat(buf,"   Polylines    -- The output dx structure will use polylines to represent Arcs.\n\n");

 strcat(buf,"Output Options:\n");
 strcat(buf,"   Binary Output Toggle -- Allows choice between Binary and ASCII output.\n");
 strcat(buf,"   File Output Toggle  -- Allows choice between file or standard output for the
conversion.\n\n");

 strcat(buf,"Right file Selection Box: Used to choose output file.\n");
 strcat(buf,"   Only visible if file output is selected\n");
 strcat(buf,"   Filter -- used to filter files in directory.\n");

 MessageBox(buf,"Shape Converter Help",MB_OK | MB_ICONINFORMATION);
```

```
}

// ***************************************************
// **** Time Series Order Dialog box and related callbacks ****

void TShpDialog::SeriesToggle_Shape()
{
 if (seriestb->GetCheck() == BF_CHECKED)
 {
  attriblst->EnableWindow(false);
  attribpb->EnableWindow(true);
 }
 else
 {
  attriblst->EnableWindow(true);
  attribpb->EnableWindow(false);
 }
}



#ifndef SERIESCALL_H
#define SERIESCALL_H

#include "gisowl.h"
#include "seriesW.rh"

enum {END = -1, BEGIN = 0, AFTER = 1};

class TSeriesDialog: public TDialog
{
 public:
            TSeriesDialog(TWindow* parent,TResId resId);
            void SetupWindow();
  void setAttribList(char **, int);
  void getSeriesList(char **, int&);
  void getTimeTags(float*,int&);
  void BeginRBPressed();
            void EndRBPressed();
            void AfterRBPressed();
            int FindInsertPos();
            void InsertTimeTag();
            void RightArrowPressed();
            void LeftSeriesSelection();
            void LeftArrowPressed();
            int IsTimeValid(float time);
            void ChangeTimeTag();
            void RightSeriesSelection();
            int CheckTimeTags();
            void OkSeries();
            void ResetTimeTags();
            void HelpSeries();
 private:
  TListBox    *attriblst, *serieslst;
  TEdit       *timetagtxt;
  TRadioButton *BeginRB, *EndRB, *AfterRB;
  TGlyphButton *LarrowB, *RarrowB;

  DECLARE_RESPONSE_TABLE(TSeriesDialog);
};

#endif


#include <stdio.h>
#include "seriesCall.h"
#include "Xshp2dx.h"
#include "callutils.h"

/* needed for use with time series dialog */
 char  leftString[SizeOfName];
 int   leftitempos;
 char  rightString[SizeOfName];
 int   rightitempos;
 int   insertMethod;

/* Time series Attributes list */
/*******************************/
 char Attribs[50][SizeOfName];
 int  nAttribs;
 char SeriesAttrib[50][SizeOfName];
 int  nSeries;
 float timetags[50];
 float oldTimeTags[50];
/*******************************/

DEFINE_RESPONSE_TABLE1(TSeriesDialog,TDialog)
 EV_CHILD_NOTIFY(IDOK, BN_CLICKED,OkSeries),
 EV_CHILD_NOTIFY(IDHELP, BN_CLICKED, HelpSeries),
 EV_CHILD_NOTIFY(IDCANCEL, BN_CLICKED, CmCancel),
```

```
  EV_CHILD_NOTIFY(IDC_BEGINRB, BN_CLICKED, BeginRBPressed),
  EV_CHILD_NOTIFY(IDC_ENDRB, BN_CLICKED, EndRBPressed),
  EV_CHILD_NOTIFY(IDC_AFTERRB, BN_CLICKED, AfterRBPressed),
  EV_CHILD_NOTIFY(IDC_ATTRIBLST, LBN_SELCHANGE, LeftSeriesSelection),
  EV_CHILD_NOTIFY(IDC_SERIESORDERLST, LBN_SELCHANGE,
RightSeriesSelection),
  EV_CHILD_NOTIFY(IDC_LEFTARROW, BN_CLICKED, LeftArrowPressed),
  EV_CHILD_NOTIFY(IDC_RIGHTARROW, BN_CLICKED, RightArrowPressed),
  EV_CHILD_NOTIFY(IDC_TIMEDEFPB, BN_CLICKED, ResetTimeTags),
  EV_CHILD_NOTIFY(IDC_TIMETAGPB, BN_CLICKED, ChangeTimeTag),
END_RESPONSE_TABLE;

TSeriesDialog::TSeriesDialog(TWindow* parent,TResId resId): TDialog(parent,resId)
{
  attriblst  = new TListBox(this, IDC_ATTRIBLST);
  seriesIst  = new TListBox(this, IDC_SERIESORDERLST);
  timetagtxt = new TEdit(this, IDC_TIMETAGTXT);
  BeginRB    = new TRadioButton(this, IDC_BEGINRB);
  EndRB      = new TRadioButton(this, IDC_ENDRB);
  AfterRB    = new TRadioButton(this, IDC_AFTERRB);
  LarrowB    = new TGlyphButton(this, IDC_LEFTARROW);
  LarrowB->SetGlyph(IDB_LARROW);
  RarrowB    = new TGlyphButton(this, IDC_RIGHTARROW);
  RarrowB->SetGlyph(IDB_RARROW);
}

void TSeriesDialog::SetupWindow()
{
  TDialog::SetupWindow();

  int i, len;
  char *tmp;

  leftString[0] = '\0';
  rightString[0] = '\0';
  leftitempos = -1;
  rightitempos = -1;
  insertMethod = END;

  timetagtxt->SetText("");
  EndRB->Check();

  /** restore dialog box **/
  for(i=0; i<nAttribs; i++)
    attriblst->AddString(Attribs[i]);
  for(i=0;i<nSeries;i++)
  {
   seriesIst->AddString(SeriesAttrib[i]);
   timetags[i] = oldTimeTags[i];
  }
  timetags[i] = -1.0;

}

void TSeriesDialog::setAttribList(char **list, int count)
{
  /** a new attribute list mean everything needs to be reset **/
  nAttribs = count;

  int i;
  for(i=0;i<nAttribs;i++)
  {
   strcpy(Attribs[i],list[i]);
   oldTimeTags[i] = -1.0;
   timetags[i] = -1.0;
  }
  oldTimeTags[i] = -1.0;
  timetags[i] = -1.0;
  nSeries  = 0;
}

void TSeriesDialog::getSeriesList(char **list, int& count)
{
  count = nSeries;
  for(int i=0;i<nSeries;i++)
    strcpy(list[i],SeriesAttrib[i]);
}

void TSeriesDialog::getTimeTags(float *times, int& count)
{
  count = nSeries;
  for(int i=0;i<nSeries;i++)
    times[i] = timetags[i];
}

void TSeriesDialog::BeginRBPressed()
{
  insertMethod = BEGIN;
}
```

```
void TSeriesDialog::EndRBPressed()
{
 insertMethod = END;
}

void TSeriesDialog::AfterRBPressed()
{
 insertMethod = AFTER;
}

/*void PopdownSeriesDialog(Widget w, Widget *wa, caddr_t call_data)
{
 XmListDeselectAllItems(wa[W1_ATTRIBLST]);
 XmListDeselectAllItems(wa[W1_SERIESORDERLST]);
}
*/

int TSeriesDialog::FindInsertPos()
{
 int nitems = 0;

 if (insertMethod == BEGIN)
  return (0);
 else if (insertMethod == AFTER)
  return (rightitempos+1);
 else
 {
  nitems = serieslst->GetCount();
  return (nitems);
 }
}

void TSeriesDialog::InsertTimeTag()
{
 int     insertpos, i;
 float   fli;

 insertpos = FindInsertPos();
 i = serieslst->GetCount();

 while(i > insertpos)
 {
  timetags[i] = timetags[i-1];
  i--;
 }
 fli = (float) (i+1);
 if (insertMethod == END)
  timetags[i] = (timetags[i-1] < fli) ? fli : -999.0;
 else
  timetags[i] = -999.0;
}

void TSeriesDialog::RightArrowPressed()
{
 if (leftitempos >= 0)
 {
  /* must be done prior to adding to the list box. */
  InsertTimeTag();
  /* else -- begin of list is position 0 which is the value of BEGIN  */
  /*      -- end of list is position -1 which is the value of END     */
  /* and of course insertMethod is equal to either END,BEGIN,or AFTER */

  if ((insertMethod == AFTER) && (rightitempos >= 0))
   serieslst->InsertString(leftString,rightitempos+1);
  else if ((insertMethod == AFTER) && (rightitempos == -1))
  {
   MessageBox("Please select an item to insert After.",0, MB_OK | MB_ICONERROR);
   return;
  }
  else
   serieslst->InsertString(leftString,insertMethod);

  if ((insertMethod == BEGIN) && (rightitempos > 0))
   rightitempos++;


  serieslst->SetSel(-1,false);
  attriblst->DeleteString(leftitempos);
  leftString[0] = '\0';
  leftitempos = -1;
 }
 else
  MessageBox("Please select an item to move from the left list",0, MB_OK |
MB_ICONERROR);
}

void TSeriesDialog::LeftSeriesSelection()
{
 leftString[0] = '\0';
```

```
 leftitempos = -1;
 attriblst->GetSelString(leftString,256);
 leftitempos = attriblst->GetSelIndex();
}

void TSeriesDialog::LeftArrowPressed()
{
 int     i;

 if (rightitempos >= 0)
 {
  seriesIst->DeleteString(rightitempos);
  attriblst->InsertString(rightString, END);

  i = rightitempos;
  while(timetags[i] != -1.0)
  {
   timetags[i] = timetags[i+1];
   i++;
  }

  rightString[0] = '\0';
  rightitempos = -1;
 }
 else
   MessageBox("Please select an item to move from the right list",0,MB_OK |
MB_ICONERROR);
}

int TSeriesDialog::IsTimeValid(float time)
{
 float prev, next;

 /** Times should look like this: **/
 /**     prev < time < next      **/

 /** make sure we don't access beyond array bounds **/
 prev = (rightitempos == 0) ? -1.0 : timetags[rightitempos-1];
 next = (timetags[rightitempos+1] == -1.0) ? 999999.0 : timetags[rightitempos+1];
 next = (next == -999.0) ? 999999.0 : next;

 if ( (time <= prev) || (time >= next))
  return 0;
 return 1;
}
```

```
void TSeriesDialog::ChangeTimeTag()
{
 char tmp[10], *err;
 float time;

 timetagtxt->GetText(tmp,10);

 err = tmp;
 time = (float)strtod(tmp,&err);
 if (*err != '\0')
 {
  MessageBox("Invalid time tag value!",0,MB_OK | MB_ICONERROR);
  timetagtxt->SetText("");
  return;
 }

 if (!IsTimeValid(time))
 {
  MessageBox("Time value is out of sequence with the other values in the list\n\
Enter a different value or reorder the list.",0,MB_OK | MB_ICONERROR);
  timetagtxt->SetText("");
  return;
 }

 timetags[rightitempos] = time;

 sprintf(tmp,"%3.3f",time);
 timetagtxt->SetText(tmp);
}

void TSeriesDialog::RightSeriesSelection()
{
 char tmp[20];

 rightitempos = -1;
 rightString[0] = '\0';

 seriesIst->GetSelString(rightString,256);
 rightitempos = seriesIst->GetSelIndex();

 /** fill in time tag text field **/
 sprintf(tmp,"%3.3f",timetags[rightitempos]);
 timetagtxt->SetText(tmp);
}
```

```
int TSeriesDialog::CheckTimeTags()
{
 int i = 0;
 char tmp[300];

 while(timetags[i] != -1.0)
 {
  if (timetags[i] < 0)
  {
   sprintf(tmp,"The time tag of attribute #%d (%3.3f) is invalid!\n",i,timetags[i]);
   strcat(tmp,"Please change the value to a positive number.");

   MessageBox(tmp,0,MB_OK | MB_ICONERROR);
   return 0;
  }
  i++;
 }
 return 1;
}

void TSeriesDialog::OkSeries()
{
 if (CheckTimeTags() == 0)
  return;

 nSeries = serieslst->GetCount();

 int i;
 for(i=0;i<nSeries;i++)
 {
  serieslst->GetString(SeriesAttrib[i],i);
  oldTimeTags[i] = timetags[i];
 }
 oldTimeTags[i] = -1.0;

 nAttribs = attriblst->GetCount();
 for(i=0;i<nAttribs;i++)
  attriblst->GetString(Attribs[i],i);

 CmOk();
}

void TSeriesDialog::ResetTimeTags()
{
 int i=0;

 while ((timetags[i] != -1.0) && (timetags[i] < 50))
 {
  timetags[i] = (float) (i+1);
  i++;
 }
 serieslst->SetSel(-1,false);
 timetagtxt->SetText("");
}

void TSeriesDialog::HelpSeries()
{
 MessageBox("Sorry, no help yet!","Series Dialog Help", MB_OK |
MB_ICONINFORMATION);
}
```

```
#ifndef CALLBACKSIMG_H
#define CALLBACKSIMG_H

#include "gisowl.h"
#include "gisfiles.h"
#include "img2dxW.rh"


class TImgDialog: public TDialog
{
 public:
        TImgDialog(TWindow* parent,TResId resId);
   void SetupWindow();
   void newOutfile();
   void newInfile();
   void newOpenDir() {openbox->NewDir();}
   void newSaveDir() {savebox->NewDir();}
   void newOpenFilter() {openbox->NewFilter();}
   void newSaveFilter() {savebox->NewFilter();}
        void fileToggle();

   /** Callbacks **/
   void SetSelectorValuesIMG(int,int,int,int);
   void SetDefaultValuesIMG();
   void clearMinMax_IMG();
   void sensitizeMinMax();
   void deSensitizeMinMax();
   void OpenFileIMG();
   void OpenLayerIMG();
   void IMGOkCallBack();
   void HelpIMG();
//   void openFileIMG();
  private:
   GisFileBox *openbox;
   GisFileBox *savebox;
   TEdit     *o_filter, *s_filter;
   TListBox   *o_dirbox, *s_dirbox;
   TListBox   *o_filebox, *s_filebox;
   TEdit     *o_select, *s_select;
   TButton    *o_filterPB,*s_filterPB;

   /** for use in various callbacks **/
   TEdit  *minxW, *minyW, *maxxW, *maxyW;
   TButton *defPB;
   TListBox *layerbox;
   TCheckBox *filetb;
   TCheckBox *bintb;

   bool MultiLayers;

 DECLARE_RESPONSE_TABLE(TImgDialog);
};

#endif
```

```
// ERDAS/IMAGINE Converter
//////////////////////////////////////////////////////////////////////
//   File:          callbacksIMG.cpp
//   Author:        Kirk A. Moeller
//   Description:
//                  Implements all the callbacks that respond
//                  to user actions in the Erdas Imagine dialog.
//////////////////////////////////////////////////////////////////////

#include "callbacksIMG.h"
#include "img.h"
/*#include "globals.h"*/
#include "callutils.h"


/** True if all layers have same width & height, allow multiple selection
 ** of layers. */
/* img converter globals */
Convert *img;

DEFINE_RESPONSE_TABLE1(TImgDialog,TDialog)
  EV_CHILD_NOTIFY(IDCANCEL, BN_CLICKED, CmCancel),
  EV_CHILD_NOTIFY(IDC_IMG_LFPB, BN_CLICKED, newOpenFilter),
  EV_CHILD_NOTIFY(IDC_IMG_LDIRBOX, LBN_DBLCLK, newOpenDir),
  EV_CHILD_NOTIFY(IDC_IMG_RFPB, BN_CLICKED, newSaveFilter),
  EV_CHILD_NOTIFY(IDC_IMG_RDIRBOX, LBN_DBLCLK, newSaveDir),
  EV_CHILD_NOTIFY(IDC_IMG_DEFPB, BN_CLICKED, SetDefaultValuesIMG),
  EV_CHILD_NOTIFY(IDC_IMG_LFILEBOX, LBN_SELCHANGE, OpenFileIMG),
  EV_CHILD_NOTIFY(IDC_IMG_RFILEBOX, LBN_SELCHANGE, newOutfile),
  EV_CHILD_NOTIFY(IDC_IMG_LAYERBOX, LBN_SELCHANGE, OpenLayerIMG),
  EV_CHILD_NOTIFY(IDOK, BN_CLICKED, IMGOkCallBack),
  EV_CHILD_NOTIFY(IDHELP, BN_CLICKED, HelpIMG),
  EV_CHILD_NOTIFY(IDC_IMG_FILETB, BN_CLICKED, fileToggle),
END_RESPONSE_TABLE;

TImgDialog::TImgDialog(TWindow* parent,TResId resId): TDialog(parent,resId)
{
  /* open box controls */
  o_filter      = new TEdit(this,IDC_IMG_LFILTER);
  o_dirbox      = new TListBox(this,IDC_IMG_LDIRBOX);
  o_filebox     = new TListBox(this,IDC_IMG_LFILEBOX);
  o_select      = new TEdit(this,IDC_IMG_LSELECT);
  o_filterPB    = new TButton(this,IDC_IMG_LFPB);

  /* save box controls */
  s_filter      = new TEdit(this,IDC_IMG_RFILTER);
  s_dirbox      = new TListBox(this,IDC_IMG_RDIRBOX);
  s_filebox     = new TListBox(this,IDC_IMG_RFILEBOX);
  s_select      = new TEdit(this,IDC_IMG_RSELECT);
  s_filterPB    = new TButton(this,IDC_IMG_RFPB);

  /* for use in various callbacks */
  minxW  = new TEdit(this,IDC_IMG_MINXTXT);
  minyW  = new TEdit(this,IDC_IMG_MINYTXT);
  maxxW  = new TEdit(this,IDC_IMG_MAXXTXT);
  maxyW  = new TEdit(this,IDC_IMG_MAXYTXT);
  defPB  = new TButton(this,IDC_IMG_DEFPB);
  layerbox = new TListBox(this,IDC_IMG_LAYERBOX);
  filetb = new TCheckBox(this,IDC_IMG_FILETB);
  bintb  = new TCheckBox(this,IDC_IMG_BINTB);
}
void TImgDialog::SetupWindow()
{
  TDialog::SetupWindow();
  filetb->Check();
  bintb->Check();

  /* this stuff can't be done until after has been created */
  openbox = new GisFileBox(this,o_filter,o_dirbox,o_filebox,o_select,
                    o_filterPB, IDC_IMG_LDIRBOX, IDC_IMG_LFILEBOX,
                    "d:\\*.img");
  savebox = new GisFileBox(this,s_filter,s_dirbox,s_filebox,s_select,
                    s_filterPB,IDC_IMG_RDIRBOX, IDC_IMG_RFILEBOX,
                    "d:\\*.dx");

  /** Do some initialization **/
//  setGisType(IMG);
  clearMinMax_IMG();
  deSensitizeMinMax();
  MultiLayers = true;
//  XmListDeleteAllItems(wa[WI_LAYERSSL_IMG]);
//
XtVaSetValues(wa[WI_LAYERSSL_IMG],XmNselectionPolicy,XmMULTIPLE_SELECT,NU
LL);

  fileState(false);
}

void TImgDialog::newOutfile()
{
```

```
 char path[256], file[256];

 savebox->GetPath(path);
 strcat(path,"\\");
 s_filebox->GetSelString(file,256);
 strcat(path,file);
 s_select->SetText(path);
}

void TImgDialog::newInfile()
{
 char path[256], file[256];

 openbox->GetPath(path);
 strcat(path,"\\");
 o_filebox->GetSelString(file,256);
 strcat(path,file);
 o_select->SetText(path);
}

// Set the values of min/max edit boxes.
void TImgDialog::SetSelectorValuesIMG(int lx, int ly, int hx, int hy)
{
 minxW->SetText(itoa(lx));
 minyW->SetText(itoa(ly));
 maxxW->SetText(itoa(hx));
 maxyW->SetText(itoa(hy));
}

// This procedure gets called if the user presses the default
// button.  The default min/max values will be placed in the
// min/max edit boxes.

void TImgDialog::SetDefaultValuesIMG()
{
 minxW->SetText("0");
 minyW->SetText("0");
 maxxW->SetText( itoa(getWidth()) );
 maxyW->SetText( itoa(getHeight()) );
}

void TImgDialog::clearMinMax_IMG()
{
 minxW->SetText("");
 minyW->SetText("");
 maxxW->SetText("");
 maxyW->SetText("");
}

void TImgDialog::sensitizeMinMax()
{
 minxW->EnableWindow(true);
 minyW->EnableWindow(true);
 maxxW->EnableWindow(true);
 maxyW->EnableWindow(true);
 defPB->EnableWindow(true);
}

void TImgDialog::deSensitizeMinMax()
{
 minxW->EnableWindow(false);
 minyW->EnableWindow(false);
 maxxW->EnableWindow(false);
 maxyW->EnableWindow(false);
 defPB->EnableWindow(false);
}

#if 0
void PopupDialogIMG(Widget w, Widget *wa, caddr_t call_data)
{
//  If either of these are true we want to output to stdout
//  and adjust the dialog box accordingly
 if (PERSISTENT || FILTER)
 {
  XtUnmanageChild(wa[WI_RFORM_IMG]);
  XmToggleButtonSetState(wa[WI_FILETB_IMG],False,False);
 }
}
#endif

// This callback procedure will take the open the selected file
// and call function which will retrieve a list of layers contained
// in the imagine file.  This list of layers is then displayed in
// the listbox in the imagine converter dialog box.

void TImgDialog::OpenFileIMG()
{
 char     filename[256];
 int      num_layers;
 int      i = 0;
```

```
 SetCursor(0, IDC_WAIT);

 // RESET SOME STUFF IN CASE THIS IS NOT THE FIRST CONVERSION
 fileState(false);
 clearMinMax_IMG();

 layerbox->ClearList();
 deSensitizeMinMax();

//*** GET INPUT FILE NAME ****
 newInfile();
 o_select->GetText(filename,256);

 if (img)
  delete img;

 // instantiate the conveter class and get the layers.
 img = new Convert(filename);
 MultiLayers = (img->ReadLayers()) ? true : false;
 num_layers = img->getnumlayers();

 if (MultiLayers && (num_layers > 0))
 {
  sensitizeMinMax();
  setWidth(img->getwidth());
  setHeight(img->getheight());
  SetDefaultValuesIMG();
  fileState(true);
 }

 // Place the list of layers in the list box.
 for(i = 0; i < num_layers; i++)
 {
  layerbox->AddString(img->layer_name[i]);
 }
 SetCursor(0, IDC_ARROW);
}

// This callback procedure gets called when the user selects one
// of the layers.  The layer information in the imagine file will
// be accessed and information such as image size will be retrieved.

void TImgDialog::OpenLayerIMG()
{
 int errval;
 int item_pos = 0;

 SetCursor(0, IDC_WAIT);

 if (MultiLayers)
 {
  SetCursor(0, IDC_ARROW);
  return;
 }
 else
 {
  int n = layerbox->GetSelCount();
  if (n > 1)
   MessageBox("Please delselect the other layer first!",0,MB_OK |
MB_ICONINFORMATION);

  if (n != 1)
  {
   SetCursor(0, IDC_ARROW);
   return;
  }
 }

 img->DestroyOldData();

 item_pos = layerbox->GetSelIndex();

 errval = img->ValidPixels(item_pos);
 if (errval == WRONGPIXTYPE)
 {
   MessageBox("Sorry, Unsupported pixel type detected in Layer!",
         0,MB_OK | MB_ICONERROR);
   // RESET SOME STUFF
   fileState(false);
   clearMinMax_IMG();

   layerbox->SetSel(-1,false);
   deSensitizeMinMax();
   SetCursor(0, IDC_ARROW);
   return;
 }

 // Turn on various controls, now that a layer has been selected.
 sensitizeMinMax();
```

```
 setWidth(img->getwidth(item_pos));
 setHeight(img->getheight(item_pos));
 SetDefaultValuesIMG();

 fileState(true);
 SetCursor(0, IDC_ARROW);
}

// This callbacks procedure calls the functions which will write the
// raster data and color table information(if present) out to a file
// or stdout in a format understandable to Data Explorer.  The data
// was retrieved in the OpenLayer callbacks above.

void TImgDialog::IMGOkCallBack()
{
 char      *outfile;
 int n = 0;
 int n2 = 0;
 bool isFile, isBinary;
 int MinMax[4];
 int *pos_list;
 int pos_count;
 int i;
 char tmp[10];

 SetCursor(0,IDC_WAIT);

 // Initialization
 pos_list = new int[10];
 outfile = new char[200];
 outfile[0] = '\0';

 // Make sure both the file and layer is open before allowing execution to
 // continue.
 n = o_filebox->GetSelCount();
 layerbox->GetSelIndexes(pos_list,10);
 pos_count = layerbox->GetSelCount();
 pos_list[pos_count] = -1;

 if(!n || !pos_count)
 {
  MessageBox("Error: NO INPUT SELECTED!",0,MB_OK | MB_ICONERROR);
  SetCursor(0,IDC_ARROW);
  return;
 }

isFile   = filetb->GetCheck() == BF_CHECKED;
isBinary = bintb->GetCheck() == BF_CHECKED;

if (!isFile)
 strcpy(outfile,"stdout");
else if (!validFile(outfile,s_select))
{
   MessageBox("NO OUTPUT FILE SELECTED!",0,MB_OK | MB_ICONERROR);
   SetCursor(0,IDC_ARROW);
   return;
}
else if (FileExists(outfile) && (!overwrite(this)))
{
   SetCursor(0,IDC_ARROW);
   return;
}

minxW->GetText(tmp,10);
MinMax[0] = atoi(tmp);
minyW->GetText(tmp,10);
MinMax[1] = atoi(tmp);
maxxW->GetText(tmp,10);
MinMax[2] = atoi(tmp);
maxyW->GetText(tmp,10);
MinMax[3] = atoi(tmp);


img->OpenOutput(outfile, isBinary);
i = 0;
while (pos_list[i] != -1)
{
 img->ReadData(pos_list[i]);
 // This is the line that does the creation of the output.
 img->WriteData((isBinary == true),MinMax);
 i++;
}
if (i == 1)
 img->WriteEnd();
else
{
 img->WriteDxGroup(pos_list);
}

delete [] pos_list;
SetCursor(0,IDC_ARROW);
```

```
}

// Either show or hide to file save box depending on file toggle state.
void TImgDialog::fileToggle()
{
 bool checked;

 checked = (filetb->GetCheck() == BF_CHECKED);
 savebox->EnableBox(checked);
}

// This procedure creates and displays the help dialog for the
// erdas imagine converter.

void TImgDialog::HelpIMG()
{

   char buf[2000];

   buf[0] = '\0';
   strcat(buf,"Left File Selection Box: used to choose input file.\n");
   strcat(buf,"   Ok      -- Start the conversion and writes out result.\n");
   strcat(buf,"   Filter  -- Used to filter files in directory.\n");
   strcat(buf,"   Cancel  -- Cancel the current converter session.\n");
   strcat(buf,"   Help    -- Popup this help message.\n\n");

   strcat(buf,"Conversion Options:\n");
   strcat(buf,"   min/max edit boxes  -- Display the size in pixels of the image. (These can be
modified.)\n");
   strcat(buf,"   Default Button      -- Resets the min/max values to there original values.\n");
   strcat(buf,"   Layers List Box     -- Displays a list of layers present in the currently opened
file.\n");
   strcat(buf,"   Stdout/File Button  -- Allows choice between file or standard output for the
conversion.\n");
   strcat(buf,"   Binary/ASCII Button -- Allows choice between Binary and ASCII output.\n\n");
   strcat(buf,"Right file Selection Box: Used to choose output file.\n");
   strcat(buf,"   Only visible if file output is selected\n");
   strcat(buf,"   Filter -- used to filter files in directory.\n\n");
   strcat(buf,"NOTES:\n");
   strcat(buf,"    -- The min/max boxes, default buttuon, and clip button will not operate until a
layer is selected.\n");
   strcat(buf,"    -- Only one layer can be converted at a time.\n");
   strcat(buf,"    -- For large images the ascii file can become very large.\n");
   strcat(buf,"         However the Binary file is considerably smaller.\n");

   MessageBox(buf,"Erdas Imagine Help",MB_OK | MB_ICONINFORMATION);
}
```

```
/*************************************************************************
** File:       FileOutCk.h
** Author:     Kirk A. Moeller
** Description:
**      This function checks if the user gave an output file and if so
**      adds a .dx extension if there not already present.
**      If no file was selected the function returns 0, otherwise it
**      will return 1;
*************************************************************************/

#ifndef CALLUTILS_H
#define CALLUTILS_H

#include "gisowl.h"

enum {Dem, Erdas, Dlg, Modflow, IMG, none};

bool fileOpen();
void fileState(bool newstate);

int gisType();
void setGisType(int newtype);

int getType();
int ChangeType(int newtype = none);

int getWidth();
void setWidth(int newWidth);

int getHeight();
void setHeight(int newHeight);

int validFile(char*,TEdit*);
int FileExists(char *file);

int  overwrite(TDialog*);

char* itoa(int i);

#endif
```

```
/*************************************************************************
** File:       callutils.cpp
** Author:     Kirk A. Moeller
** Description:
**         This file contain a couple of utility functions used by
**         various callbacks in the converters.
*************************************************************************/

#include "callutils.h"
#include <windows.h>
#include <string.h>
#include <fstream.h>
#include <stdio.h>

/** The variables that these functions modify and return
 ** were originally global.  These functions seemed the best way
 ** of limiting the scope of the variables.
**/

/*********************************************/
static bool isFileOpen = false;

bool fileOpen()
{
 return isFileOpen;
}

void fileState(bool newstate)
{
 isFileOpen = newstate;
}

/*********************************************/
static int GISTYPE = none;

int gisType()
{
 return (GISTYPE);
}

void setGisType(int newtype)
{
 GISTYPE = newtype;
}

/*********************************************/
static int WIDTH  = 0;
static int HEIGHT = 0;

int getWidth()
{
 return WIDTH;
}

void setWidth(int newWidth)
{
 WIDTH = (newWidth < 0) ? 0 : newWidth;
}

int getHeight()
{
 return HEIGHT;
}

void setHeight(int newHeight)
{
 HEIGHT = (newHeight < 0) ? 0 : newHeight;
}

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


/** This function checks if the user gave an output file and if so
**  adds a .dx extension if there not already present.
**  If no file was selected the function returns 0, otherwise it
**  will return 1;
**/
int validFile(char* outfile,TEdit* selectTxt)
{
 char *p, file[256];

 selectTxt->GetText(file,256);

 p = strrchr(file,'\\');;
 if (p == NULL) return 0;

 p++;
 if ( (*p == '\0') || (*p == ' ') )
  return 0;
 else if ( !(strstr(p,".dx")) )
```

```
  {
   strcat(file,".dx");
   selectTxt->SetText(file);
  }
  strcpy(outfile,file);
  return 1;
}

int FileExists(char *file)
{
  int exists;

  ifstream fin(file);

  exists = (fin) ? 1 : 0;

  fin.close();
  return exists;
}


/************************************************
**  OVERWRITE FUNCTIONS **
************************************************/

/** Used by callbacks to popup a overwrite warning box
**  and then wait for the user response.
**/
int overwrite(TDialog* dlg)
{
  int answer;

  answer = dlg->MessageBox("OverWrite File!","File Exists!",
                    MB_OKCANCEL | MB_ICONQUESTION);

  /* ID0K seems to be undefined for some reason */
  if (answer == IDOK)
    return 1;
  else
    return 0;
}

/************************************************/

// converter an integer to an ascii string.

char* itoa(int i)
{
  static char buf[256];
  sprintf(buf, "%d", i);
  return buf;
}
```

```
#ifndef GISFILES_H
#define GISFILES_H

#include <windows.h>
#include <winuser.h>
#include "gisowl.h"
#include <stdlib.h>

class GisFileBox
{
 public:
   GisFileBox(TDialog*,TEdit*,TListBox*,TListBox*,TEdit*,TButton*,
         int,int,char far*);
   ~GisFileBox() {}

   void EnableBox(bool);
   void NewFilter();
   void NewDir();
   void GetPath(char far*);
   void GetWild(char far*);
 private:
   void    initBox();
   void    UpdateFileBox();
   void    UpdatePath(char far*);
   TDialog  *TheDialog;
   TEdit    *filtertxt;
   TListBox *dirbox;
   TListBox *filebox;
   TEdit    *selecttxt;
   TButton  *filterPB;
            int     dirboxid;
            int     fileboxid;
   char far path[256];
            char    drive;
            int     openbox;
            char    wild[20];
};

#endif
```

```
// ****************************************
// File:          gisfiles.cpp
// Author:        Kirk A. Moeller
// Description:
//        This is the implementation file of the class which implements the file
//        selection functionality in the converter dialog boxes.
//        This class was created in order to simulate was occurs in the X/Motif
//         XmFileSelectionBox.
// ****************************************
#include "gisfiles.h"

GisFileBox::GisFileBox(TDialog* tmpdlg, TEdit* tmpfilter, TListBox* tmpdirbox,
                TListBox* tmpfilebox, TEdit* tmpselect,
                TButton* tmpfilterPB, int tmpdirid, int tmpfileid,
                char far *tmppath)
: TheDialog(tmpdlg), filtertxt(tmpfilter), dirbox(tmpdirbox),
  filebox(tmpfilebox), selecttxt(tmpselect), filterPB(tmpfilterPB),
  dirboxid(tmpdirid), fileboxid(tmpfileid)
{
  path[0] = '\0';
  strcpy(path,tmppath);

  drive = path[0];
  initBox();
}

void GisFileBox::initBox()
{
  filtertxt->SetText(path);
  UpdateFileBox();
}

void GisFileBox::EnableBox(bool state)
{
  filtertxt->EnableWindow(state);
  dirbox->EnableWindow(state);
  filebox->EnableWindow(state);
  selecttxt->EnableWindow(state);
  filterPB->EnableWindow(state);
}

void GisFileBox::GetPath(char far * NewPath)
{
  char far *p;

  if (path[1] == ':')
    p = path + 2;
  else
    p = path;

  NewPath[0] = '\0';
  strcpy(NewPath,p);

  int i = strlen(NewPath);
  while (NewPath[i] != '\\')
    i--;
  NewPath[i] = '\0';
}

void GisFileBox::GetWild(char far *wild)
{
  char far * p;

  int i = strlen(path);

  wild[0] = '\0';
  while (path[i] != '\\')
    i--;
  p = path + i + 1;

  strcpy(wild,p);
}

void GisFileBox::UpdateFileBox()
{
  char far DirPath[256];
  char far FilePath[256];
  char tmp[256];

  DirPath[0]  = '\0';
  FilePath[0] = '\0';
  if (path[1] != ':')
  {
    DirPath[0]  = drive;
    DirPath[1]  = '\0';
    FilePath[0] = drive;
    FilePath[1] = '\0';
  }
  strcat(DirPath,path);
  strcat(FilePath,path);
```

```
  // Change DirPath to correct directory path.
  int i = strlen(DirPath);
  while(DirPath[i] != '\\')
    i--;
  while(DirPath[i] != '\\')
    i--;
  DirPath[i+1] = '\0';
  strcat(DirPath,"*.*");

  if
(!DlgDirList(TheDialog->HWindow,DirPath,dirboxid,NULL,DDL_DIRECTORY|DDL_EXCL
USIVE))
    dirbox->MessageBox("Error: Invalid path");
  if (!DlgDirList(TheDialog->HWindow,FilePath,fileboxid,NULL,DDL_READWRITE))
    dirbox->MessageBox("Error: Invalid path");

  GetPath(tmp);
  strcat(tmp,"\\");
  selecttxt->SetText(tmp);
}

void GisFileBox::NewFilter()
{
  char far newpath[256];

  filtertxt->GetText(newpath,256);
  UpdatePath(newpath);
  UpdateFileBox();
  filtertxt->SetText(path);
}

void GisFileBox::NewDir()
{
  char far sel[256], *psel, *wild;
  int len;

  wild   = new char far[20];
  sel[0]  = '\0';
  dirbox->GetSelString(sel,256);

  // remove the [ ] from the string.
  psel = sel + 1;
  len = strlen(sel);
  sel[len-1] = '\0';
  GetWild(wild);
  int i = strlen(path);
  while (path[i] != '\\')
    i--;
  path[i] = '\0';

  if (strcmp(psel,"..") != 0)
  {
                        if (path[3] != '\0')
                                strcat(path,"\\");
                        strcat(path,psel);
  }
  else      // go up one directory level
  {
                        int k=(strlen(path)-1);
                        while (path[k--] != '\\')
                                ;
                        path[k+1] = '\0';
  }
  strcat(path,"\\");
  strcat(path,wild);
  UpdateFileBox();
  delete wild;
  filtertxt->SetText(path);
}

void GisFileBox::UpdatePath(char far* newpath)
{
  // Update the drive and path information.
  path[0] = '\0';

  // Get the new drive, if any.
  if (path[1] == ':')
    drive = path[0];
  else
  {
    path[0] = drive;
    path[1] = '\0';
  }
  strcpy(path,newpath);
}
```

```
/* File: ConvW.rh */

/* MAIN WINDOW */
#define IDD_MAINDLG 101
#define IDD_DEMDLG 102
#define IDD_DLGDLG          103
#define IDC_DEMPB 106
#define IDC_DLGPB 107
#define IDC_ERDASPB 108
#define IDC_MODFLOWPB 110
#define IDC_SHAPEPB 111
#define IDC_IMAGINEPB 112

/* DEM, MODFLOW, ERDAS DEFINES */
#define IDC_LAYERSLBL 115
#define IDC_LAYERSSL        116
#define IDC_BINTB  117
#define IDC_FILETB 118
#define IDC_RSELECT         119
#define IDC_RFILEBOX        120
#define IDC_RDIRBOX         121
#define IDC_RFILTER         122
#define IDC_CLIPPB 123
#define IDC_DEFPB  124
#define IDC_MAXYTXT         125
#define IDC_MAXXTXT         126
#define IDC_MINYTXT         127
#define IDC_MINXTXT         128
#define IDC_RFPB    129
#define IDC_LFPB    130
#define IDC_RGRP    131
#define IDC_LSELECT         132
#define IDC_LDIRBOX         133
#define IDC_LFILEBOX        134
#define IDC_LFILTER         135
#define IDC_LGRP    136

/* DLG DEFINES */
#define IDC_DLG_FILETB      140
#define IDC_DLG_BINTB       141
#define IDC_DLG_BINPB       142
#define IDC_DLG_CLIPPB      143
#define IDC_DLG_RFPB        144
#define IDC_DLG_RSELECT     145
#define IDC_DLG_RFILEBOX    146
#define IDC_DLG_RDIRBOX     147
#define IDC_DLG_RFILTER     148
#define IDC_DLG_LFPB        149
#define IDC_DLG_LSELECT     150
#define IDC_DLG_LFILEBOX    151
#define IDC_DLG_LDIRBOX     152
#define IDC_DLG_LFILTER     153
#define IDC_DLG_STATTXT     154
```

```
/****************************************************************************

gis2dx.rc

produced by Borland Resource Workshop

****************************************************************************/
#include "convW.rh"

IDD_MAINDLG DIALOGEX 0, 0, 275, 225
STYLE DS_MODALFRAME | DS_3DLOOK | DS_CONTEXTHELP | WS_CHILD |
WS_VISIBLE
FONT 8, "MS Sans Serif", 400, 0
{
 CONTROL "Quit", IDCANCEL, "BUTTON", BS_PUSHBUTTON | BS_CENTER |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 4, 202, 50, 14
 CONTROL "Help", IDHELP, "BUTTON", BS_PUSHBUTTON | BS_CENTER | WS_CHILD |
WS_VISIBLE | WS_TABSTOP, 220, 202, 50, 14
 CONTROL "Dataset Import and Conversion", -1, "static", SS_CENTER | WS_CHILD |
WS_VISIBLE, 0, 5, 276, 8
 CONTROL "Univeristy of Montana", -1, "static", SS_CENTER | WS_CHILD | WS_VISIBLE,
0, 20, 276, 8
 CONTROL "Distributed Applications and Systems Lab (DASL)", -1, "static", SS_CENTER |
WS_CHILD | WS_VISIBLE, 0, 30, 276, 8
 CONTROL "AND", -1, "static", SS_CENTER | WS_CHILD | WS_VISIBLE, 0, 40, 276, 8
 CONTROL "IBM Visualization Data Explorer", -1, "static", SS_CENTER | WS_CHILD |
WS_VISIBLE, 0, 48, 276, 8
 CONTROL "DEM", IDC_DEMPB, "button", BS_PUSHBUTTON | BS_CENTER |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 8, 78, 50, 14
 CONTROL "DLG", IDC_DLGPB, "button", BS_PUSHBUTTON | BS_CENTER |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 8, 95, 50, 14
 CONTROL "ERDAS", IDC_ERDASPB, "button", BS_PUSHBUTTON | BS_CENTER |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 8, 112, 50, 14
 CONTROL "MODFLOW", IDC_MODFLOWPB, "button", BS_PUSHBUTTON |
BS_CENTER | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 8, 129, 50, 14
 CONTROL "ArcShape", IDC_SHAPEPB, "button", BS_PUSHBUTTON | BS_CENTER |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 8, 146, 50, 14
 CONTROL "ERDAS/IMG", IDC_IMAGINEPB, "button", BS_PUSHBUTTON | BS_CENTER
| WS_CHILD | WS_VISIBLE | WS_TABSTOP, 8, 164, 50, 14
 CONTROL "USGS/DEM To DX", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 108,
80, 76, 8
 CONTROL "USGS/DLG To DX", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 108,
97, 76, 8
 CONTROL "ERDAS/LAN To DX", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 108,
114, 76, 8
 CONTROL "USGS/ModFlow (typical) To DX", -1, "static", SS_LEFT | WS_CHILD |
WS_VISIBLE, 108, 131, 104, 8
 CONTROL "ArcShape to DX [beta]", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 108,
148, 76, 8
 CONTROL "ERDAS/IMAGINE To DX [beta]", -1, "static", SS_LEFT | WS_CHILD |
WS_VISIBLE, 108, 166, 84, 8
 CONTROL "Frame3", -1, "static", SS_ETCHEDFRAME | WS_CHILD | WS_VISIBLE, 0, 194,
280, 1
}


IDD_DEMDLG DIALOGEX 0, 0, 415, 240
STYLE DS_MODALFRAME | DS_3DLOOK | DS_CENTER | DS_CONTEXTHELP |
WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "DEM Converter"
FONT 8, "MS Sans Serif"
{
 CONTROL "OK", IDOK, "BUTTON", BS_PUSHBUTTON | BS_CENTER | WS_CHILD |
WS_VISIBLE | WS_TABSTOP, 4, 217, 32, 14
 CONTROL "Quit", IDCANCEL, "BUTTON", BS_PUSHBUTTON | BS_CENTER |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 83, 217, 32, 14
 CONTROL "Help", IDHELP, "BUTTON", BS_PUSHBUTTON | BS_CENTER | WS_CHILD |
WS_VISIBLE | WS_TABSTOP, 123, 217, 32, 14
 CONTROL "Select File To Convert", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 4,
13, 76, 8
 CONTROL "Filter", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 4, 34, 20, 8
 CONTROL "", IDC_LGRP, "button", BS_GROUPBOX | WS_CHILD | WS_VISIBLE |
WS_GROUP, 0, 0, 160, 237
 CONTROL "", IDC_LFILTER, "edit", ES_LEFT | WS_CHILD | WS_VISIBLE |
WS_BORDER | WS_TABSTOP, 4, 43, 140, 12
 CONTROL "Directories", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 4, 61, 36, 8
 CONTROL "", IDC_LFILEBOX, "listbox", LBS_STANDARD | WS_CHILD | WS_VISIBLE |
WS_TABSTOP, 80, 72, 70, 113
 CONTROL "Selection", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 4, 185, 32, 8
 CONTROL "", IDC_LSELECT, "edit", ES_LEFT | WS_CHILD | WS_VISIBLE |
WS_BORDER | WS_TABSTOP, 4, 194, 148, 12
 CONTROL "", -1, "static", SS_ETCHEDFRAME | WS_CHILD | WS_VISIBLE, 0, 211, 159, 1
 CONTROL "", 0, "button", BS_GROUPBOX | WS_CHILD | WS_VISIBLE | WS_GROUP,
256, 1, 156, 236
 CONTROL "Statistics", -1, "static", SS_CENTER | WS_CHILD | WS_VISIBLE, 172, 9, 68, 8
 CONTROL "Select File To Save As", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 260,
14, 76, 8
 CONTROL "Filter", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 260, 31, 20, 8
```

```
CONTROL "Directories", -I, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 260, 62, 36, 8
CONTROL "Files", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 336, 62, 20, 8
CONTROL "Selection", -I, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 260, 185, 32, 8
CONTROL "", IDC_RFILTER, "edit", ES_LEFT | WS_CHILD | WS_VISIBLE |
WS_BORDER | WS_TABSTOP, 260, 44, 144, 12
CONTROL "", IDC_RSELECT, "edit", ES_LEFT | WS_CHILD | WS_VISIBLE |
WS_BORDER | WS_TABSTOP, 260, 194, 144, 12
CONTROL "Filter", IDC_RFPB, "button", BS_PUSHBUTTON | BS_CENTER | WS_CHILD |
WS_VISIBLE | WS_TABSTOP, 312, 217, 32, 14
CONTROL "Filter", IDC_LFPB, "button", BS_PUSHBUTTON | BS_CENTER | WS_CHILD |
WS_VISIBLE | WS_TABSTOP, 43, 217, 32, 14
CONTROL "Min X:", -I, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 164, 30, 24, 8
CONTROL "Min Y:", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 164, 45, 24, 8
CONTROL "Max X:", -I, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 164, 59, 24, 8
CONTROL "Max Y:", -I, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 164, 73, 24, 8
CONTROL "", IDC_MINXTXT, "edit", ES_LEFT | WS_CHILD | WS_VISIBLE |
WS_BORDER | WS_TABSTOP, 192, 29, 56, 12
CONTROL "", IDC_MINYTXT, "edit", ES_LEFT | WS_CHILD | WS_VISIBLE |
WS_BORDER | WS_TABSTOP, 192, 43, 56, 12
CONTROL "", IDC_MAXXTXT, "edit", ES_LEFT | WS_CHILD | WS_VISIBLE |
WS_BORDER | WS_TABSTOP, 192, 57, 56, 12
CONTROL "", IDC_MAXYTXT, "edit", ES_LEFT | WS_CHILD | WS_VISIBLE |
WS_BORDER | WS_TABSTOP, 192, 71, 56, 12
CONTROL "Default", IDC_DEFPB, "button", BS_PUSHBUTTON | BS_CENTER |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 196, 86, 50, 14
CONTROL "Files", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 80, 61, 20, 8
CONTROL "", -I, "static", SS_ETCHEDFRAME | WS_CHILD | WS_VISIBLE, 256, 211, 156,
I
CONTROL "", IDC_LDIRBOX, "listbox", LBS_STANDARD | WS_CHILD | WS_VISIBLE |
WS_TABSTOP, 4, 72, 70, 113
CONTROL "", IDC_RDIRBOX, "listbox", LBS_STANDARD | WS_CHILD | WS_VISIBLE |
WS_TABSTOP, 260, 73, 70, 113
CONTROL "", IDC_RFILEBOX, "listbox", LBS_STANDARD | WS_CHILD | WS_VISIBLE |
WS_TABSTOP, 336, 73, 70, 113
CONTROL "File Output", IDC_FILETB, "button", BS_AUTOCHECKBOX | WS_CHILD |
WS_VISIBLE | WS_TABSTOP, 176, 220, 65, 12, WS_EX_DLGMODALFRAME
CONTROL "Binary Output", IDC_BINTB, "button", BS_AUTOCHECKBOX | WS_CHILD |
WS_VISIBLE | WS_TABSTOP, 176, 204, 64, 12, WS_EX_DLGMODALFRAME
CONTROL "Frame3", -1, "static", SS_ETCHEDFRAME | WS_CHILD | WS_VISIBLE, 160,
189, 96, I
CONTROL "Output Options:", -I, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 164, 193,
60, 8
CONTROL "Frame4", -I, "static", SS_ETCHEDFRAME | WS_CHILD | WS_VISIBLE, 160,
121, 96, I
CONTROL "Layers", IDC_LAYERSLBL, "static", SS_LEFT | WS_CHILD | WS_VISIBLE,
163, 125, 60, 8
CONTROL "ListBox5", IDC_LAYERSSL, "listbox", LBS_NOTIFY | LBS_MULTIPLESEL |
WS_CHILD | WS_VISIBLE | WS_BORDER | WS_VSCROLL | WS_TABSTOP, 168, 134, 79,
55
CONTROL "Frame5", -1, "static", SS_GRAYFRAME | WS_CHILD | WS_VISIBLE, 160, 18,
96, 1
}

IDD_DLGDLG DIALOGEX 0, 0, 423, 247
STYLE DS_MODALFRAME | DS_3DLOOK | DS_CONTEXTHELP | WS_POPUP |
WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "DLG Converter"
FONT 8, "MS Sans Serif"
{
CONTROL "OK", IDOK, "BUTTON", BS_PUSHBUTTON | BS_CENTER | WS_CHILD |
WS_VISIBLE | WS_TABSTOP, 4, 224, 32, 14
CONTROL "Quit", IDCANCEL, "BUTTON", BS_PUSHBUTTON | BS_CENTER |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 80, 224, 32, 14
CONTROL "Help", IDHELP, "BUTTON", BS_PUSHBUTTON | BS_CENTER | WS_CHILD |
WS_VISIBLE | WS_TABSTOP, 119, 224, 32, 14
CONTROL "", -1, "button", BS_GROUPBOX | WS_CHILD | WS_VISIBLE | WS_GROUP, 0,
0, 156, 246
CONTROL "", 0, "button", BS_GROUPBOX | WS_CHILD | WS_VISIBLE | WS_GROUP,
265, 1, 156, 244
CONTROL "Select File to Convert", -I, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 4, 13,
76, 8
CONTROL "Filter", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 4, 34, 20, 8
CONTROL "", IDC_DLG_LFILTER, "edit", ES_LEFT | WS_CHILD | WS_VISIBLE |
WS_BORDER | WS_TABSTOP, 4, 43, 148, 12
CONTROL "Directories", -I, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 4, 61, 36, 8
CONTROL "Files", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 80, 60, 20, 8
CONTROL "", IDC_DLG_LDIRBOX, "listbox", LBS_STANDARD | WS_CHILD |
WS_VISIBLE | WS_TABSTOP, 4, 72, 70, 113
CONTROL "", IDC_DLG_LFILEBOX, "listbox", LBS_STANDARD | WS_CHILD |
WS_VISIBLE | WS_TABSTOP, 79, 72, 70, 113
CONTROL "Selection", -I, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 4, 194, 32, 8
CONTROL "", IDC_DLG_LSELECT, "edit", ES_LEFT | WS_CHILD | WS_VISIBLE |
WS_BORDER | WS_TABSTOP, 4, 202, 148, 12
CONTROL "Filter", IDC_DLG_LFPB, "button", BS_PUSHBUTTON | BS_CENTER |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 42, 224, 32, 14
CONTROL "", -I, "static", SS_ETCHEDFRAME | WS_CHILD | WS_VISIBLE, 0, 220, 156, I
CONTROL "Select File to Save As", -I, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 269,
14, 76, 8
CONTROL "Filter", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 269, 35, 20, 8
CONTROL "", IDC_DLG_RFILTER, "edit", ES_LEFT | WS_CHILD | WS_VISIBLE |
```

```
WS_BORDER | WS_TABSTOP, 269, 44, 148, 12
 CONTROL "Directories", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 269, 62, 36, 8
 CONTROL "Files", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 345, 62, 20, 8
 CONTROL "", IDC_DLG_RDIRBOX, "listbox", LBS_STANDARD | WS_CHILD |
WS_VISIBLE | WS_TABSTOP, 269, 73, 70, 113
 CONTROL "", IDC_DLG_RFILEBOX, "listbox", LBS_STANDARD | WS_CHILD |
WS_VISIBLE | WS_TABSTOP, 344, 73, 70, 113
 CONTROL "Selection", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 269, 195, 32, 8
 CONTROL "", IDC_DLG_RSELECT, "edit", ES_LEFT | WS_CHILD | WS_VISIBLE |
WS_BORDER | WS_TABSTOP, 269, 203, 148, 12
 CONTROL "", -1, "static", SS_ETCHEDFRAME | WS_CHILD | WS_VISIBLE, 265, 222, 156,
1
 CONTROL "Filter", IDC_DLG_RFPB, "button", BS_PUSHBUTTON | BS_CENTER |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 317, 225, 32, 14
 CONTROL "Statistics", -1, "static", SS_CENTER | WS_CHILD | WS_VISIBLE, 160, 4, 100, 8
 CONTROL "", IDC_DLG_STATTXT, "static", SS_LEFT | SS_SUNKEN | WS_CHILD |
WS_VISIBLE, 160, 26, 100, 168
 CONTROL "Frame3", -1, "static", SS_ETCHEDFRAME | WS_CHILD | WS_VISIBLE, 156,
198, 108, I
 CONTROL "Output Options:", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 159, 204,
60, 8
 CONTROL "Binary Output", IDC_DLG_BINTB, "button", BS_AUTOCHECKBOX |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 170, 215, 60, 12,
WS_EX_DLGMODALFRAME
 CONTROL "File Output", IDC_DLG_FILETB, "button", BS_AUTOCHECKBOX |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 170, 230, 60, 12,
WS_EX_DLGMODALFRAME
}
```

/* File: **shp2dxW.rh** */

```
/* SHAPE DEFINES */
#define IDD_SHAPEDLG        105
#define IDC_SHAPE_ATTLBL    155
#define IDC_SHAPE_SERIESTB          156
#define IDC_SHAPE_ATTRIBPB          157
#define IDC_SHAPE_FILETB    158
#define IDC_SHAPE_BINTB     159
#define IDC_SHAPE_ATTBOX    160
#define IDC_SHAPE_LSELECT   161
#define IDC_SHAPE_LFPB      162
#define IDC_SHAPE_RFPB      163
#define IDC_SHAPE_RSELECT   164
#define IDC_SHAPE_RFILEBOX          165
#define IDC_SHAPE_RDIRBOX           166
#define IDC_SHAPE_RFILTER   167
#define IDC_SHAPE_LFILEBOX          168
#define IDC_SHAPE_LDIRBOX           169
#define IDC_SHAPE_LFILTER   170
#define IDC_SHAPE_POLYRB    171
#define IDC_SHAPE_LINESRB   172
```

```
/* File: shp2dxW.rc */

#include "shp2dxW.rh"

IDD_SHAPEDLG DIALOGEX 0, 0, 401, 246
EXSTYLE WS_EX_DLGMODALFRAME | WS_EX_CONTEXTHELP
STYLE DS_MODALFRAME | DS_3DLOOK | DS_CONTEXTHELP | WS_POPUP |
WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "SHAPE Converter"
FONT 8, "MS Sans Serif"
{
 CONTROL "OK", IDOK, "BUTTON", BS_PUSHBUTTON | BS_CENTER | WS_CHILD |
WS_VISIBLE | WS_TABSTOP, 4, 224, 32, 14
 CONTROL "Quit", IDCANCEL, "BUTTON", BS_PUSHBUTTON | BS_CENTER |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 82, 224, 32, 14
 CONTROL "Help", IDHELP, "BUTTON", BS_PUSHBUTTON | BS_CENTER | WS_CHILD |
WS_VISIBLE | WS_TABSTOP, 120, 224, 32, 14
 CONTROL "", -1, "button", BS_GROUPBOX | WS_CHILD | WS_VISIBLE | WS_GROUP, 0,
0, 156, 246
 CONTROL "", 0, "button", BS_GROUPBOX | WS_CHILD | WS_VISIBLE | WS_GROUP,
244, 1, 156, 244
 CONTROL "Options", -1, "static", SS_CENTER | WS_CHILD | WS_VISIBLE, 160, 5, 76, 8
 CONTROL "Attributes: ", IDC_SHAPE_ATTLBL, "static", SS_LEFT | WS_CHILD |
WS_VISIBLE, 160, 22, 76, 8
 CONTROL "", IDC_SHAPE_ATTBOX, "listbox", LBS_NOTIFY | LBS_MULTIPLESEL |
WS_CHILD | WS_VISIBLE | WS_BORDER | WS_VSCROLL | WS_TABSTOP, 159, 34, 81,
70
 CONTROL "", 3, "button", BS_GROUPBOX | WS_CHILD | WS_VISIBLE | WS_GROUP,
160, 145, 80, 52
 CONTROL "Lines", IDC_SHAPE_LINESRB, "button", BS_AUTORADIOBUTTON |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 180, 164, 48, 12
 CONTROL "Polylines", IDC_SHAPE_POLYRB, "button", BS_AUTORADIOBUTTON |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 180, 177, 48, 12
 CONTROL "Select File to Convert", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 4, 13,
76, 8
 CONTROL "Filter", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 4, 33, 20, 8
 CONTROL "", IDC_SHAPE_LFILTER, "edit", ES_LEFT | WS_CHILD | WS_VISIBLE |
WS_BORDER | WS_TABSTOP, 4, 43, 148, 12
 CONTROL "Directories", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 4, 61, 36, 8
 CONTROL "Files", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 80, 60, 20, 8
 CONTROL "", IDC_SHAPE_LDIRBOX, "listbox", LBS_STANDARD | WS_CHILD |
WS_VISIBLE | WS_TABSTOP, 4, 72, 70, 113
 CONTROL "", IDC_SHAPE_LFILEBOX, "listbox", LBS_STANDARD | WS_CHILD |
WS_VISIBLE | WS_TABSTOP, 80, 72, 70, 113
 CONTROL "Select File to Save As", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 248,
14, 76, 8
 CONTROL "Filter", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 248, 35, 20, 8
 CONTROL "", IDC_SHAPE_RFILTER, "edit", ES_LEFT | WS_CHILD | WS_VISIBLE |
WS_BORDER | WS_TABSTOP, 248, 44, 144, 12
 CONTROL "Directories", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 248, 62, 36, 8
 CONTROL "Files", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 324, 62, 20, 8
 CONTROL "", IDC_SHAPE_RDIRBOX, "listbox", LBS_STANDARD | WS_CHILD |
WS_VISIBLE | WS_TABSTOP, 248, 73, 70, 113
 CONTROL "", IDC_SHAPE_RFILEBOX, "listbox", LBS_STANDARD | WS_CHILD |
WS_VISIBLE | WS_TABSTOP, 323, 73, 70, 113
 CONTROL "Selection", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 248, 194, 32, 8
 CONTROL "", IDC_SHAPE_RSELECT, "edit", ES_LEFT | WS_CHILD | WS_VISIBLE |
WS_BORDER | WS_TABSTOP, 248, 202, 148, 12
 CONTROL "", -1, "static", SS_ETCHEDFRAME | WS_CHILD | WS_VISIBLE, 244, 220, 156,
1
 CONTROL "Filter", IDC_SHAPE_RFPB, "button", BS_PUSHBUTTON | BS_CENTER |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 300, 224, 32, 14
 CONTROL "Filter", IDC_SHAPE_LFPB, "button", BS_PUSHBUTTON | BS_CENTER |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 43, 224, 32, 14
 CONTROL "Selection", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 4, 194, 32, 8
 CONTROL "", IDC_SHAPE_LSELECT, "edit", ES_LEFT | WS_CHILD | WS_VISIBLE |
WS_BORDER | WS_TABSTOP, 4, 202, 148, 12
 CONTROL "", -1, "static", SS_ETCHEDFRAME | WS_CHILD | WS_VISIBLE, 0, 220, 156, 1
 CONTROL "Attribute Dialog ...", IDC_SHAPE_ATTRIBPB, "button", BS_PUSHBUTTON |
BS_CENTER | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 160, 118, 80, 14
 CONTROL "Create Time Series", IDC_SHAPE_SERIESTB, "button",
BS_AUTOCHECKBOX | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 160, 103, 80, 12,
WS_EX_DLGMODALFRAME
 CONTROL "Frame3", -1, "static", SS_ETCHEDFRAME | WS_CHILD | WS_VISIBLE, 156,
138, 88, 1
 CONTROL "DX Structure", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 164, 151, 60,
8
 CONTROL "Frame4", -1, "static", SS_ETCHEDFRAME | WS_CHILD | WS_VISIBLE, 156,
201, 88, 1
 CONTROL "Output Options", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 159, 207,
60, 8
 CONTROL "Binary Output", IDC_SHAPE_BINTB, "button", BS_AUTOCHECKBOX |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 164, 217, 60, 12,
WS_EX_DLGMODALFRAME
 CONTROL "File Output", IDC_SHAPE_FILETB, "button", BS_AUTOCHECKBOX |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 164, 231, 60, 12,
WS_EX_DLGMODALFRAME
 CONTROL "Frame5", -1, "static", SS_GRAYFRAME | WS_CHILD | WS_VISIBLE, 156, 14,
88, 2
}
```

```
/* File: seriesW.rh */

/* TIME SERIES DIALOG DEFINES */
#define IDD_SERIESDLG       113
#define IDC_TIMEDEFPB       200
#define IDC_LEFTARROW       201
#define IDC_RIGHTARROW      202
#define IDC_TIMETAGTXT      203
#define IDC_TIMETAGPB       204
#define IDC_SERIESORDERLST          205
#define IDC_AFTERRB         206
#define IDC_ENDRB 207
#define IDC_BEGINRB         208
#define IDC_ATTRIBLST       209
#define IDB_LARROW          210
#define IDB_RARROW 211
```

```
/* File: seriesW.rc */

#include "seriesW.rh"

IDB_LARROW BITMAP
{
'42 4D F6 00 00 00 00 00 00 00 76 00 00 00 28 00'
'00 00 10 00 00 00 10 00 00 00 01 00 04 00 00 00'
'00 00 80 00 00 00 00 00 00 00 00 00 00 00 00 00'
'00 00 10 00 00 00 00 00 00 00 00 00 80 00 00 80'
'00 00 00 80 80 00 80 00 00 00 80 00 80 00 80 80'
'00 00 C0 C0 C0 00 80 80 80 00 00 00 FF 00 00 FF'
'00 00 00 FF FF 00 FF 00 00 00 FF 00 FF 00 FF FF'
'00 00 FF FF FF 00 77 77 77 77 77 77 77 00 77 77'
'77 77 77 77 00 00 77 77 77 77 77 00 00 00 77 77'
'77 77 00 00 00 00 77 77 77 00 00 00 00 00 77 77'
'00 00 00 00 00 00 77 00 00 00 00 00 00 00 00 00'
'00 00 00 00 00 00 00 00 00 00 00 00 00 00 77 00'
'00 00 00 00 00 00 77 77 00 00 00 00 00 00 77 77'
'77 00 00 00 00 00 77 77 77 77 00 00 00 00 77 77'
'77 77 77 70 00 00 77 77 77 77 77 77 70 00 77 77'
'77 77 77 77 77 70'
}


IDB_RARROW BITMAP
{
'42 4D F6 00 00 00 00 00 00 00 76 00 00 00 28 00'
'00 00 10 00 00 00 10 00 00 00 01 00 04 00 00 00'
'00 00 80 00 00 00 00 00 00 00 00 00 00 00 00 00'
'00 00 10 00 00 00 00 00 00 00 00 00 80 00 00 80'
'00 00 00 80 80 00 80 00 00 00 80 00 80 00 80 80'
'00 00 C0 C0 C0 00 80 80 80 00 00 00 FF 00 00 FF'
'00 00 00 FF FF 00 FF 00 00 00 FF 00 FF 00 FF FF'
'00 00 FF FF FF 00 00 77 77 77 77 77 77 77 00 00'
'77 77 77 77 77 77 00 00 00 77 77 77 77 77 00 00'
'00 00 77 77 77 77 00 00 00 00 00 77 77 77 00 00'
'00 00 00 00 77 77 00 00 00 00 00 00 00 77 00 00'
'00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
'00 00 00 00 00 77 00 00 00 00 00 00 77 77 00 00'
'00 00 00 77 77 77 00 00 00 00 77 77 77 77 00 00'
'00 77 77 77 77 77 77 00 00 77 77 77 77 77 77 00 77'
'77 77 77 77 77 77'
}
```

```
IDD_SERIESDLG DIALOGEX 0, 0, 353, 212
EXSTYLE WS_EX_DLGMODALFRAME | WS_EX_CONTEXTHELP
STYLE DS_MODALFRAME | DS_3DLOOK | DS_CONTEXTHELP | WS_POPUP |
WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Order Time Series"
FONT 8, "MS Sans Serif"
{
 CONTROL "OK", IDOK, "BUTTON", BS_PUSHBUTTON | BS_CENTER | WS_CHILD |
WS_VISIBLE | WS_TABSTOP, 4, 194, 50, 14
 CONTROL "Cancel", IDCANCEL, "BUTTON", BS_PUSHBUTTON | BS_CENTER |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 60, 194, 50, 14
 CONTROL "Help", IDHELP, "BUTTON", BS_PUSHBUTTON | BS_CENTER | WS_CHILD |
WS_VISIBLE | WS_TABSTOP, 299, 194, 50, 14
 CONTROL "List of Attributes names", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 4,
4, 60, 8
 CONTROL "ListBox1", IDC_ATTRIBLST, "listbox", LBS_NOTIFY | WS_CHILD |
WS_VISIBLE | WS_BORDER | WS_VSCROLL | WS_TABSTOP, 4, 14, 130, 154
 CONTROL "Press Arrow to move", -1, "static", SS_CENTER | WS_CHILD | WS_VISIBLE,
136, 18, 80, 8
 CONTROL "a selected attribute", -1, "static", SS_CENTER | WS_CHILD | WS_VISIBLE, 136,
28, 80, 8
 CONTROL "", IDC_RIGHTARROW, "button", BS_PUSHBUTTON | BS_BITMAP |
BS_CENTER | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 151, 41, 50, 14
 CONTROL "", IDC_LEFTARROW, "button", BS_PUSHBUTTON | BS_CENTER |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 151, 61, 50, 14
 CONTROL "ListBox2", IDC_SERIESORDERLST, "listbox", LBS_NOTIFY | WS_CHILD |
WS_VISIBLE | WS_BORDER | WS_VSCROLL | WS_TABSTOP, 219, 13, 130, 73
 CONTROL "Frame1", -1, "static", SS_ETCHEDFRAME | WS_CHILD | WS_VISIBLE, 134,
87, 220, 1
 CONTROL "Time Tag:", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 232, 98, 36, 8
 CONTROL "", IDC_TIMETAGTXT, "edit", ES_LEFT | WS_CHILD | WS_VISIBLE |
WS_BORDER | WS_TABSTOP, 269, 97, 29, 12
 CONTROL "Enter", IDC_TIMETAGPB, "button", BS_PUSHBUTTON | BS_CENTER |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 299, 97, 27, 14
 CONTROL "Select an insertion method:", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE,
232, 121, 88, 8
 CONTROL "Beginning of List", IDC_BEGINRB, "button", BS_AUTORADIOBUTTON |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 248, 135, 68, 12
 CONTROL "End of List", IDC_ENDRB, "button", BS_AUTORADIOBUTTON | WS_CHILD |
WS_VISIBLE | WS_TABSTOP, 248, 150, 68, 12
 CONTROL "After selected item", IDC_AFTERRB, "button", BS_AUTORADIOBUTTON |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 248, 164, 68, 12
 CONTROL "Reset Times", IDC_TIMEDEFPB, "button", BS_PUSHBUTTON | BS_CENTER |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 117, 194, 50, 14
 CONTROL "Frame2", -1, "static", SS_ETCHEDFRAME | WS_CHILD | WS_VISIBLE, 1, 184,
```

```
351, 1
 CONTROL "Attributes in time series", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE,
219, 3, 80, 8
 CONTROL "Frame3", -1, "static", SS_ETCHEDFRAME | WS_CHILD | WS_VISIBLE, 134,
15, 85, 1
}
```

```
/* File: img2dxW.rh */

/* IMAGINE DEFINES */

#define IDD_IMAGINEDLG 104
#define IDC_IMG_LFILTER     175
#define IDC_IMG_CLIPPB      176
#define IDC_IMG_FILETB      177
#define IDC_IMG_BINTB       178
#define IDC_IMG_LAYERBOX    179
#define IDC_IMG_DEFPB       180
#define IDC_IMG_MAXYTXT     181
#define IDC_IMG_MAXXTXT     182
#define IDC_IMG_MINYTXT     183
#define IDC_IMG_MINXTXT     184
#define IDC_IMG_RSELECT     185
#define IDC_IMG_RFILEBOX    186
#define IDC_IMG_RDIRBOX     187
#define IDC_IMG_RFILTER     188
#define IDC_IMG_LFILEBOX    189
#define IDC_IMG_LDIRBOX     190
#define IDC_IMG_LSELECT     191
#define IDC_IMG_RFPB        192
#define IDC_IMG_LFPB        193
```

```
/* File: img2dxW.rc */

#include "img2dxW.rh"

IDD_IMAGINEDLG DIALOGEX 0, 0, 403, 242
STYLE DS_MODALFRAME | DS_3DLOOK | DS_CONTEXTHELP | WS_POPUP |
WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Imagine Converter"
FONT 8, "MS Sans Serif"
{
 CONTROL "OK", IDOK, "BUTTON", BS_PUSHBUTTON | BS_CENTER | WS_CHILD |
WS_VISIBLE | WS_TABSTOP, 4, 224, 32, 14
 CONTROL "Quit", IDCANCEL, "BUTTON", BS_PUSHBUTTON | BS_CENTER |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 76, 224, 32, 14
 CONTROL "Help", IDHELP, "BUTTON", BS_PUSHBUTTON | BS_CENTER | WS_CHILD |
WS_VISIBLE | WS_TABSTOP, 112, 224, 32, 14
 CONTROL "", -1, "button", BS_GROUPBOX | WS_CHILD | WS_VISIBLE | WS_GROUP, 0,
0, 156, 241
 CONTROL "", 0, "button", BS_GROUPBOX | WS_CHILD | WS_VISIBLE | WS_GROUP,
244, 0, 156, 241
 CONTROL "Filter", IDC_IMG_LFPB, "button", BS_PUSHBUTTON | BS_CENTER |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 40, 224, 32, 14
 CONTROL "Filter", IDC_IMG_RFPB, "button", BS_PUSHBUTTON | BS_CENTER |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 300, 224, 32, 14
 CONTROL "Select File to Convert", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 4, 13,
76, 8
 CONTROL "Filter", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 4, 34, 20, 8
 CONTROL "", IDC_IMG_LFILTER, "edit", ES_LEFT | WS_CHILD | WS_VISIBLE |
WS_BORDER | WS_TABSTOP, 4, 43, 148, 12
 CONTROL "Directories", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 4, 61, 36, 8
 CONTROL "Files", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 80, 61, 20, 8
 CONTROL "", IDC_IMG_LDIRBOX, "listbox", LBS_STANDARD | WS_CHILD |
WS_VISIBLE | WS_TABSTOP, 4, 72, 70, 113
 CONTROL "", IDC_IMG_LFILEBOX, "listbox", LBS_STANDARD | WS_CHILD |
WS_VISIBLE | WS_TABSTOP, 80, 72, 70, 113
 CONTROL "Selection", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 4, 194, 32, 8
 CONTROL "", IDC_IMG_LSELECT, "edit", ES_LEFT | WS_CHILD | WS_VISIBLE |
WS_BORDER | WS_TABSTOP, 4, 202, 148, 12
 CONTROL "", -1, "static", SS_ETCHEDFRAME | WS_CHILD | WS_VISIBLE, 0, 220, 156, 1
 CONTROL "Select File to Save As", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 248,
14, 76, 8
 CONTROL "Filter", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 248, 35, 20, 8
 CONTROL "", IDC_IMG_RFILTER, "edit", ES_LEFT | WS_CHILD | WS_VISIBLE |
WS_BORDER | WS_TABSTOP, 248, 44, 148, 12
 CONTROL "Directories", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 248, 62, 36, 8
 CONTROL "Files", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 324, 62, 20, 8
 CONTROL "", IDC_IMG_RDIRBOX, "listbox", LBS_STANDARD | WS_CHILD |
WS_VISIBLE | WS_TABSTOP, 248, 73, 70, 113
 CONTROL "", IDC_IMG_RFILEBOX, "listbox", LBS_STANDARD | WS_CHILD |
WS_VISIBLE | WS_TABSTOP, 323, 73, 70, 113
 CONTROL "Selection", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 248, 194, 32, 8
 CONTROL "", IDC_IMG_RSELECT, "edit", ES_LEFT | WS_CHILD | WS_VISIBLE |
WS_BORDER | WS_TABSTOP, 248, 202, 148, 12
 CONTROL "", -1, "static", SS_ETCHEDFRAME | WS_CHILD | WS_VISIBLE, 244, 220, 156,
1
 CONTROL "Statistics", -1, "static", SS_CENTER | WS_CHILD | WS_VISIBLE, 160, 4, 76, 8
 CONTROL "Min X:", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 160, 30, 24, 8
 CONTROL "Min Y:", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 160, 45, 24, 8
 CONTROL "Max X:", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 160, 59, 24, 8
 CONTROL "Max Y:", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 160, 73, 24, 8
 CONTROL "", IDC_IMG_MINXTXT, "edit", ES_LEFT | WS_CHILD | WS_VISIBLE |
WS_BORDER | WS_TABSTOP, 188, 29, 48, 12
 CONTROL "", IDC_IMG_MINYTXT, "edit", ES_LEFT | WS_CHILD | WS_VISIBLE |
WS_BORDER | WS_TABSTOP, 188, 43, 48, 12
 CONTROL "", IDC_IMG_MAXXTXT, "edit", ES_LEFT | WS_CHILD | WS_VISIBLE |
WS_BORDER | WS_TABSTOP, 188, 57, 48, 12
 CONTROL "", IDC_IMG_MAXYTXT, "edit", ES_LEFT | WS_CHILD | WS_VISIBLE |
WS_BORDER | WS_TABSTOP, 188, 71, 48, 12
 CONTROL "Default", IDC_IMG_DEFPB, "button", BS_PUSHBUTTON | BS_CENTER |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 188, 86, 48, 14
 CONTROL "Layers", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 160, 107, 76, 8
 CONTROL "", IDC_IMG_LAYERBOX, "listbox", LBS_STANDARD | LBS_MULTIPLESEL
| WS_CHILD | WS_VISIBLE | WS_TABSTOP, 164, 116, 72, 60
 CONTROL "Frame3", -1, "static", SS_GRAYFRAME | WS_CHILD | WS_VISIBLE, 156, 17,
88, 2
 CONTROL "Frame4", -1, "static", SS_ETCHEDFRAME | WS_CHILD | WS_VISIBLE, 156,
104, 88, 1
 CONTROL "Frame5", -1, "static", SS_ETCHEDFRAME | WS_CHILD | WS_VISIBLE, 156,
191, 88, 1
 CONTROL "Output Options:", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 160, 196,
60, 8
 CONTROL "Binary Output", IDC_IMG_BINTB, "button", BS_AUTOCHECKBOX |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 172, 209, 60, 12,
WS_EX_DLGMODALFRAME
 CONTROL "File Output", IDC_IMG_FILETB, "button", BS_AUTOCHECKBOX |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 172, 224, 60, 12,
WS_EX_DLGMODALFRAME
}
```

Appendix E

Source code for X/Motif version of GIS2DX. Some non-GUI code has been omitted.

```
// DEM, DLG, ERDAS, MODFLOW, ARC/INFO, SHAPE, ORACLE, DXF, ERDAS/IMAGINE
Converters
//////////////////////////////////////////////////////////////////
//  File:           globals.h
//  Author:         Kirk A. Moeller
//  Description:
//                  global flags used by all converters.
//////////////////////////////////////////////////////////////////

extern int PERSISTENT;     /* do one conversion and quit, stdout mode          */
extern int SPECIFIC;       /* execution begins directly at a particular converter */
extern int DIR_FLAG;       /* flag that indicates the user specified a directory  */
extern int FILTER;         /* begin in standard output mode                    */
extern char *DEF_DIR;      /* string containing the current directory          */
extern char *GIS2DX_HOME;  /* string to hold to path to the executable         */
extern Widget* g_fw_plist; /* Widget array used in the oracle converter        */
```

```
// DEM, DLG, ERDAS, MODFLOW, ARC/INFO, ORACLE, DXF, ERDAS/IMAGINE Converters
//////////////////////////////////////////////////////////////////////
//   File:          Xheaders.h
//   Author:        David Thompson, Petr Votava, and others.
//   Description:
//                  A file containing all of the includes
//                  necessary to use all of the X/Motif stuff.
//////////////////////////////////////////////////////////////////////

#ifndef XHEADERS_H
#define XHEADERS_H

#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Shell.h>
#include <Xm/Xm.h>
#include <Xm/Protocols.h>

#include <stdlib.h>
#include <stdio.h>
#include <malloc.h>
#include <memory.h>
#include <Xm/Xm.h>
#include <X11/Shell.h>
#include <X11/StringDefs.h>
#include <Xm/DialogS.h>
#include <Xm/DrawingA.h>
#include <Xm/FileSB.h>
#include <Xm/Form.h>
#include <Xm/Label.h>
#include <Xm/List.h>
#include <Xm/MessageB.h>
#include <Xm/MwmUtil.h>
#include <Xm/PanedW.h>
#include <Xm/PushB.h>
#include <Xm/RowColumn.h>
#include <Xm/ScrolledW.h>
#include <Xm/SelectioB.h>
#include <Xm/Separator.h>
#include <Xm/Text.h>
#include <Xm/TextF.h>
#include <Xm/ToggleB.h>
#include <Xm/ArrowB.h>

#endif
```

```
//////////////////////////////////////////////////////////////////
//   File:          main.C
//   Author:        David Thompson
//                  Vijayant Palaiya
//                  Dick Thompson
//                  Petr Votava
//                  Kirk A. Moeller
//   Description:
//                  This file contains the main() function of
//                  the gis2dx application.  The dialog boxes are
//                  created, the resources are processed, and the
//                  command line arguments are processed.
//////////////////////////////////////////////////////////////////

#include <stdlib.h>
#include <unistd.h>       /* For getcwd */

#include "XHeaders.h"

#include "convW.h"
#include "arcW.h"
#include "shp2dxW.h"
#include "ora_fw.h"
#include "dxfW.h"
#include "img2dxW.h"

#include "callbacksARC.h"
#include "shp_callbacks.h"
#include "ora_callbacks.h"
#include "callbacksDXF.h"
#include "callbacksIMG.h"

#include "version.h"
#include "machine.h"

// contents of globals.h
// the "real" declarations
int PERSISTENT;
int SPECIFIC;
int DIR_FLAG;
int FILTER;
char *DEF_DIR;
char *GIS2DX_HOME;

Widget* g_fw_plist;

//VJ
extern XtAppContext appContext;

extern Display * sDisplay;
extern Screen * sScreen;

extern const char *PROGRAM_CLASS;
extern char *PROGRAM_NAME;
//VJ

// Get fallback resources

#if (XmVersion > 1000)
   /* fallbackResources are only used from Motif 1.1  **/
   static String fallbackResources[] = {
#   if defined(vax11c) || defined(__DECC)
#     include "$A$sedit.ad$5nh"
#   else
#     include "Conv.ad.h"
#   endif /* vax11c */
"*filter1:                  sed 's/^[       ]*$//' | sed '\\n\
/^$/ {\\n\
N\\n\
/^\\\\n$/D\\n\
    }'",                          /* to work around bug in ad2c !!!!*/
"*appDefaultsVersion:    777",          /* to recognize a fallback case */
      NULL
   };

#endif  /* XmVersion ... */


void CheckLicense(Widget w)
{
  char *lic;
  FILE *havelic;
  int n;

  n = 0;
  lic = (char*) malloc(sizeof(char) * 80);
  lic[0] = '\0';

  /* Check for a License */
  get_lic(lic);
  if (lic[0] == '\0')
```

```
 {
    XtSetSensitive(w, False);
 }
 else
 {
  strcat(lic,"/sysgen/license.dat");
  havelic = fopen(lic,"r");
  if (!havelic)
    XtSetSensitive(w, False);
 }
 free(lic);
}


void main(int argc, char* argv[])
{
 char **arg;
 int k;
 char *tmp;
 char bitmapfile[40];
 unsigned int width_return, height_return;
 Pixmap   bitmap;
 int      x_hot_return, y_hot_return;
 Drawable   draw;


 /* (Display *) */ sDisplay = NULL; //VJ
 PERSISTENT = 0;
 SPECIFIC = 0;
 DIR_FLAG = 0;
 FILTER = 0;

 DEF_DIR = (char *)malloc(sizeof(char)*80);
 GIS2DX_HOME = new char[100];
 GIS2DX_HOME[0] = '\0';

 strcpy(GIS2DX_HOME,argv[0]);

 /* We want GIS2DX_HOME to have the path where gis2dx */
 /* is located */
 if (( PROGRAM_NAME = strrchr(argv[0], (int) '/')) == NULL)
 {
    PROGRAM_NAME = argv[0];
    GIS2DX_HOME[0] = '\0';
    getcwd(GIS2DX_HOME,100);
    strcat(GIS2DX_HOME,"/");
 }
 else
 {
    tmp = strrchr(GIS2DX_HOME,(int) '/');
    tmp++;
    *tmp = '\0';
    PROGRAM_NAME++;
 }

 Widget *main;
 Widget *shp_warr;
 Widget *dxfwarr;
 Widget *imgwarr;
 Widget *arcwarr;

 XtToolkitInitialize();
 appContext = XtCreateApplicationContext();
 XtAppSetFallbackResources(appContext, fallbackResources);

/*   this ifdef for hpux removed per nancy's request. Kirk M.

 sDisplay = XtOpenDisplay(appContext, NULL, PROGRAM_NAME,
            PROGRAM_CLASS, NULL, 0, (Cardinal *)&argc, argv);
*/
 sDisplay = XtOpenDisplay(appContext, NULL, PROGRAM_NAME,
            PROGRAM_CLASS, NULL, 0, &argc, argv);
 if (!sDisplay)
 {
    XtWarning ("can't open display, exiting...");
    exit (1);
 }

 /*** BEGIN WIDGET CREATION ***/

 /** MAIN, DEM, DLG, ERDAS, MODFLOW **/
 tu_applicationshell_widget(NULL, NULL, &main);

 draw = XRootWindowOfScreen(XtScreen(main[WI_APPLICATIONSHELL]));
 bitmapfile[0] = '\0';
 strcpy(bitmapfile,"./bitmaps/IBM.bmp");
 XReadBitmapFile(sDisplay,draw, bitmapfile, &width_return,
          &height_return, &bitmap, &x_hot_return, &y_hot_return);
```

```
XtVaSetValues(main[WI_APPLICATIONSHELL],XmNiconPixmap, bitmap, NULL);

/** ARC SHAPE **/
tu_FConv_Shape_widget("Shape Converter",main[WI_FORM],&shp_warr);
AddShapeCallbacks(shp_warr,main[WI_SHAPEPB]);

/** DXF **/
tu_DialogShellDXF_widget("DXF CONVERTER",main[WI_FORM],&dxfwarr);
AddDXFCallbacks(dxfwarr,main[WI_DXFPB]);

/** ERDAS IMAGINE **/
tu_IMGDlgShell_widget("ERDAS Imagine Converter",main[WI_FORM],&imgwarr);
AddIMGCallbacks(imgwarr,main,main[WI_IMGPB]);

/** ARC/INFO **/
tu_FConv_Arc_widget("Arc Converter",main[WI_FORM],&arcwarr);
AddCallbacksARC(arcwarr,main,main[WI_ARCPB]);
CheckLicense(main[WI_ARCPB]);

/*** END WIDGET CREATION ***/

if(argc == 1)
XtRealizeWidget(main[WI_APPLICATIONSHELL]);
else
{
   arg = (char **)malloc(sizeof(char *)*5);
   for(k = 0; k < 5; k++)
   {
    arg[k] = (char *)malloc(sizeof(char)*80);
    arg[k][0] = '\0';
   }
   for(k = 0; k < argc; k++)
   {
      if(!strcmp(argv[k], "-all"))
        strcpy(arg[0], argv[k]);
      else if(!strcmp(argv[k], "-lan"))
        strcpy(arg[0], argv[k]);
      else if(!strcmp(argv[k], "-dem"))
        strcpy(arg[0], argv[k]);
      else if(!strcmp(argv[k], "-dlg"))
        strcpy(arg[0], argv[k]);
      else if(!strcmp(argv[k], "-mod"))
        strcpy(arg[0], argv[k]);
      else if(!strcmp(argv[k], "-arcinfo"))
        strcpy(arg[0], argv[k]);
      else if(!strcmp(argv[k], "-shape"))
        strcpy(arg[0], argv[k]);

      else if(!strcmp(argv[k], "-oracle"))
        strcpy(arg[0], argv[k]);

      else if(!strcmp(argv[k], "-imagine"))
        strcpy(arg[0], argv[k]);
      else if(!strcmp(argv[k], "-dxf"))
        strcpy(arg[0], argv[k]);
      else if(!strcmp(argv[k], "-p"))
        strcpy(arg[1], argv[k]);
      else if (!strcmp(argv[k], "-filter"))
        strcpy(arg[3],argv[k]);
      else if(!strcmp(argv[k], "-dir"))
      {
        strcpy(arg[2], argv[k+1]);
        k++;
      }
}

if(arg[1])
{
 if(!strcmp(arg[1], "-p"))
   PERSISTENT = 1;
}
if(arg[2])
{
 DIR_FLAG = 1;
 strcpy(DEF_DIR,arg[2]);
}
if (arg[3])
{
 if (!strcmp(arg[3], "-filter"))
   FILTER = 1;
}
if(arg[0][0] == '\0'|| !strcmp(arg[0], "-all"))
{
 XtRealizeWidget(main[WI_APPLICATIONSHELL]);
}
else if(!strcmp(arg[0], "-lan"))
{
 SPECIFIC = 1;
 XtRealizeWidget(main[WI_FCONV]);
 XtCallCallbacks(main[WI_ERDASPB], XmNactivateCallback, (caddr_t) main);
```

```
}
else if(!strcmp(arg[0], "-dem"))
{
 SPECIFIC = 1;
 XtRealizeWidget(main[WI_FCONV]);
 XtCallCallbacks(main[WI_DEMPB], XmNactivateCallback, (caddr_t) main);
}
else if(!strcmp(arg[0], "-dlg"))
{
 SPECIFIC = 1;
 XtRealizeWidget(main[WI_FCONV1]);
 XtCallCallbacks(main[WI_DLGPB], XmNactivateCallback, (caddr_t) main);
}
else if(!strcmp(arg[0], "-mod"))
{
 SPECIFIC = 1;
 XtRealizeWidget(main[WI_FCONV]);
 XtCallCallbacks(main[WI_MODFLOWPB], XmNactivateCallback, (caddr_t) main);
}
else if(!strcmp(arg[0], "-arcinfo"))
{
 SPECIFIC = 1;
 XtRealizeWidget(arcwarr[WI_FILECONVPU_ARC]);
 XtCallCallbacks(main[WI_ARCPB], XmNactivateCallback, (caddr_t) main);
}
else if(!strcmp(arg[0], "-shape"))
{
 SPECIFIC = 1;
 XtRealizeWidget(shp_warr[WI_FILECONVPU_SHAPE]);
 XtCallCallbacks(main[WI_SHAPEPB], XmNactivateCallback, (caddr_t) main);
}
else if(!strcmp(arg[0], "-oracle"))
{
 SPECIFIC = 1;
 tu_orafw_DS_widget("ORACLE Converter", main[WI_ORACLEPB],&g_fw_plist);
 AddOraFwCallbacks(g_fw_plist);
 XtRealizeWidget(g_fw_plist[WI_ORAFW_FORM]);
 XtCallCallbacks(main[WI_ORACLEPB], XmNactivateCallback, (caddr_t) main);
}
else if(!strcmp(arg[0], "-imagine"))
{
 SPECIFIC = 1;
 XtRealizeWidget(imgwarr[WI_IMGFORM]);
 XtCallCallbacks(main[WI_IMGPB], XmNactivateCallback, (caddr_t) main);
}
  else if(!strcmp(arg[0], "-dxf"))
  {
   SPECIFIC = 1;
   XtRealizeWidget(dxfwarr[WI_FILECONVDXF]);
   XtCallCallbacks(main[WI_DXFPB], XmNactivateCallback, (caddr_t) main);
  }
 }
 XtAppMainLoop(appContext);
}
```

```
// DEM, MODFLOW, ERDAS, DLG Converters
//////////////////////////////////////////////////////////////////////
//   File:          callbacks.h
//   Author:          David Thompson, Dick Thompson, Petr Votava
//                  Kirk Moeller, and others
//   Description:
//                  Header file specifying prototypes for the
//                  callback functions for the Converters listed
//                  above.
//////////////////////////////////////////////////////////////////////

#ifndef CALLBACKS_H
#define CALLBACKS_H

#include "XHeaders.h"
#include <fstream.h>

/* STRUCTURES */
class rubber_band_data
{
 public:
            int start_x, start_y, last_x, last_y;
            int max_x, max_y;
            GC gc;
};

/* Defines */
#define MIN(a,b) ((a) < (b) ? (a) : (b))
#define MAX(a,b) ((a) > (b) ? (a) : (b))

/*
#define Dem 0
#define LAN 1
#define Dlg 2
#define MOD 3
#define Arc 4
#define IMG 5
*/
/* PROTOTYPES */
/* utilities */
char* between(int, int, int);

void AddCallbacks(Widget w[]);
//void PopupDialog(Widget w, Widget dialog, caddr_t call_data);
void MakeVisible(Widget w);
void MakeInvisible(Widget w);
void ChangeMask(Widget w, char* mask);

void PopupDialogMisc(Widget *wa);
void PopupDialogERDAS(Widget w, Widget *wa, caddr_t call_data);
void PopupDialogMODFLOW(Widget w, Widget *wa, caddr_t call_data);
void PopupDialogDEM(Widget w, Widget *wa, caddr_t call_data);
void PopdownDialog(Widget w, Widget dialog, caddr_t call_data);
void Quit(Widget w, caddr_t client_data, caddr_t call_data);

void PopupSelector(Widget w, Widget *wa, caddr_t call_data);
void start_rubber_band(Widget w, Widget* wa, XEvent *event);
void track_rubber_band(Widget w, Widget* wa, XEvent *event);
void end_rubber_band(Widget w, Widget* wa, XEvent *event);
void clear_rubber_band(Widget w, Widget* wa, caddr_t call_data);
void ReturnSelector(Widget w, Widget* wa, caddr_t call_data);
void SetSelectorValues(Widget *wa, int minx, int miny, int maxx, int maxy);

void OpenFile(Widget w, Widget* WA, XmListCallbackStruct *list);
void OpenFileDem(Widget w, Widget *WA, ifstream& in);
void OpenFileMOD(Widget w, Widget *WA, ifstream& in);
void OpenFileDlg(Widget w, Widget *WA, ifstream& in);
void OpenFileLAN(Widget w, Widget *WA, ifstream& in);
void ResetFileOpenBox(Widget *wa);

void PlaceDefault(Widget w, Widget* WA, caddr_t list);
void ClearMinMax(Widget *WA);

void OkCallBack(Widget w, Widget* WA, XmFileSelectionBoxCallbackStruct *cbs);
void OtherRun(Widget w, Widget* WA, char* file1, char* file2, Boolean isFile);
void DLGRun(Widget w, Widget* WA, char* file1, char* file2, Boolean isFile);

void ToggleMeFile (Widget widget, Widget savebox, XmAnyCallbackStruct *call_data);

void HelpSelector(Widget w, Widget *WA, XmAnyCallbackStruct *call_data);
void HelpTop(Widget w, Widget *WA, XmAnyCallbackStruct *call_data);
void HelpAll(Widget w, Widget *WA, XmAnyCallbackStruct *call_data);

#endif
```

```
// DEM, DLG, ERDAS, MODFLOW Converters
//////////////////////////////////////////////////////////////////////
//   File:          callbacks.C
//   Author:        David Thompson, Dick Thompson, Petr Votava,
//                  Kirk Moeller, and others.
//   Description:
//                  Implementation of all of the callbacks
//                  functions necessary to implement all of the
//                  operations of the dialogs associated with the
//                  converters list above.
//////////////////////////////////////////////////////////////////////

#include <unistd.h>
#include <iostream.h>

#include "callbacks.h"
#include "ora_callbacks.h"
#include "callbacksARC.h"
#include "callbacksIMG.h"

#include "convW.h"
#include "ora_fw.h"
#include "arcW.h"

#include "dem2dx.h"
#include "dlg2dx.h"
#include "mod2dx.h"
#include "e2dx.h"

#include "cursor.h"
#include "errorbox.h"
#include "callutils.h"
#include "globals.h"

//extern "C" gethostname(char*,int);

/* globals */
rubber_band_data data;
CursorClass gCursor;
float SCALE;

/* converter globals */
int BANDS = 0;

char* between(int i, int j, int ROW)
{
 static char buf[256];
 if(ROW)
   sprintf(buf, "%d - %d", MAX(0, MIN(i,j)), MIN(getWidth(), MAX(i,j)));
 else
   sprintf(buf, "%d - %d", MAX(0, MIN(i,j)), MIN(getHeight(), MAX(i,j)));
 return buf;
}


/*
 ** This function registers the callbacks for the DEM, DLG, ERDAS, and
 ** MODFLOW converters.  It also registers the callbacks for the clipping
 ** box, which is used by several of the converter dialogs.
 ** The callbacks for the other converter dialogs are registered in
 ** other files:
 **   callbacksARC.C  shp_callbacks.C  ora_callbacks.C  callbacksDXF.C
 **   callbacksDXF.C  callbacksIMG.C
*/

void AddCallbacks(Widget w[])
{
  Widget tmpw1;
  Atom   WM_DELETE_WINDOW;

// Note: The ARC      pushbutton is registered in callbacksARC.C
//       The ArcShape  pushbutton is registered in shp_callbacks.C
//       The ERDAS/IMG pushbutton is registered in callbacksIMG.C
//       The DXF      pushbutton is registered in callbacksDXF.C
// APPLICATION SHELL callbacks
   XtAddCallback(w[WI_CANCELPB1], XmNactivateCallback,
            (XtCallbackProc) Quit, NULL);
   XtAddCallback(w[WI_HELPPB1], XmNactivateCallback,
            (XtCallbackProc) HelpTop, w);
   XtAddCallback(w[WI_DEMPB], XmNactivateCallback,
            (XtCallbackProc) PopupDialogDEM, w);
   XtAddCallback(w[WI_ERDASPB], XmNactivateCallback,
            (XtCallbackProc) PopupDialogERDAS, w);
   XtAddCallback(w[WI_MODFLOWPB], XmNactivateCallback,
            (XtCallbackProc) PopupDialogMODFLOW, w);
   XtAddCallback(w[WI_ORACLEPB], XmNactivateCallback,
            (XtCallbackProc) Start_Ora, w);


// FCONV callbacks (Callbacks for the DEM, ERDAS, and MODFLOW converters)
```

```
    XtAddCallback(w[WI_SELECTORPB], XmNactivateCallback,
            (XtCallbackProc) PopupSelector, w);
    XtAddCallback(w[WI_FILESELBOXOPEN], XmNcancelCallback,
            (XtCallbackProc) PopdownDialog, w[WI_FILECONVPU]);

    tmpw1 = XmSelectionBoxGetChild(w[WI_FILESELBOXOPEN],XmDIALOG_FILE_LIST);
    XtAddCallback(tmpw1, XmNbrowseSelectionCallback,
            (XtCallbackProc) OpenFile, w);

    tmpw1 = XmSelectionBoxGetChild(w[WI_FILESELBOXCONV],XmDIALOG_TEXT);
    XtAddCallback(tmpw1, XmNactivateCallback,
            (XtCallbackProc) OkCallBack, w);
//          tmpw1 = XmSelectionBoxGetChild(w[WI_FILESELBOXOPEN],
//                                  XmDIALOG_HELP_BUTTON);
    XtAddCallback(w[WI_FILESELBOXOPEN], XmNhelpCallback,
            (XtCallbackProc) HelpAll, w);
    XtAddCallback(w[WI_DEFAULTPB], XmNactivateCallback,
            (XtCallbackProc) PlaceDefault, w);
    XtAddCallback(w[WI_FILESELBOXOPEN], XmNokCallback,
            (XtCallbackProc) OkCallBack, w);
    XtAddCallback(w[WI_FILESAVETB], XmNvalueChangedCallback,
            (XtCallbackProc) ToggleMeFile, w[WI_RFORM_DEM]);

    WM_DELETE_WINDOW = XmInternAtom(XtDisplay(w[WI_FCONV]),
                        "WM_DELETE_WINDOW",False);
    XmAddWMProtocolCallback(w[WI_FCONV],WM_DELETE_WINDOW,
            (XtCallbackProc)PopdownDialog,w[WI_FILECONVPU]);

// SELECTORBOX callbacks (i.e the clipping box, in various dialogs.
    XtAddCallback(w[WI_CANCELPB], XmNactivateCallback,
            (XtCallbackProc) PopdownDialog, NULL);
    XtAddCallback(w[WI_HELPPB], XmNactivateCallback,
            (XtCallbackProc) HelpSelector, w);
    XtAddCallback(w[WI_DRAWINGAREA], XmNexposeCallback,
            (XtCallbackProc) clear_rubber_band, w);
    XtAddEventHandler(w[WI_DRAWINGAREA], ButtonPressMask, FALSE,
                (XtEventHandler) start_rubber_band, w);
    XtAddEventHandler(w[WI_DRAWINGAREA], ButtonMotionMask, FALSE,
                (XtEventHandler) track_rubber_band, w);
    XtAddEventHandler(w[WI_DRAWINGAREA], ButtonReleaseMask, FALSE,
                (XtEventHandler) end_rubber_band, w);
    XtAddCallback(w[WI_OKPB], XmNactivateCallback,
            (XtCallbackProc) ReturnSelector, w);

}
```

```
void PopupDialog(Widget w, Widget dialog, caddr_t call_data)
{
   XtManageChild(dialog);
}

void MakeVisible(Widget w)
{
 XtSetMappedWhenManaged(w,True);
}

void MakeInvisible(Widget w)
{
 XtSetMappedWhenManaged(w,False);
}

void ChangeMask(Widget w, char* mask)
{
 XmString xms = (XmString) NULL;

 xms = XmStringCreate(mask, XmSTRING_DEFAULT_CHARSET);
 XtVaSetValues(w, XmNdirMask, xms, NULL);
 if (xms) XmStringFree(xms);
}

void PopupDialogMisc(Widget *wa)
{
 if(PERSISTENT || FILTER)
 {
  XtUnmanageChild(wa[WI_RFORM_DEM]);
  XmToggleButtonSetState(wa[WI_FILESAVETB],False,False);
 }
 XmTextSetString(wa[WI_MINXTXT], "");
 XmTextSetString(wa[WI_MINYTXT], "");
 XmTextSetString(wa[WI_MAXXTXT], "");
 XmTextSetString(wa[WI_MAXYTXT], "");
 XmListDeleteAllItems(wa[WI_LAYERSSL]);
}

void PopupDialogERDAS(Widget w, Widget *wa, caddr_t call_data)
{
 if(!SPECIFIC)
 {
  gCursor.SetAppShellWidget(w);
  gCursor.DontEnter();
```

```
 }
 fileState(False);
 setGisType(LAN);

 // Set the title and file filters to appropriate values.
 ChangeMask(wa[WI_FILESELBOXOPEN],"*.lan");
 XtVaSetValues(wa[WI_FCONV], XtNtitle,  "ERDAS Converter", NULL);

 // Do some initialization stuff which is common to the three
 // converters which use this particular dialog box.
 PopupDialogMisc(wa);

 MakeVisible(wa[WI_LAYERSLBL]);
 MakeVisible(wa[WI_LAYERSSL]);
 MakeVisible(wa[WI_SELECTORPB]);
 XtManageChild(wa[WI_FILECONVPU]);
}


void PopupDialogMODFLOW(Widget w, Widget *wa, caddr_t call_data)
{
 if(!SPECIFIC)
 {
  gCursor.SetAppShellWidget(w);
  gCursor.DontEnter();
 }
 fileState(False);
 setGisType(MOD);

 // Set title and filters to appropriate values.
 ChangeMask(wa[WI_FILESELBOXOPEN],"*");
 XtVaSetValues(wa[WI_FCONV], XtNtitle,  "MODFLOW Converter", NULL);

 // Do some initialization stuff which is common to the three
 // converters which use this particular dialog box.
 PopupDialogMisc(wa);

 XtManageChild(wa[WI_FILECONVPU]);
 MakeInvisible(wa[WI_LAYERSLBL]);
 MakeInvisible(wa[WI_LAYERSSL]);
 MakeInvisible(wa[WI_SELECTORPB]);
}


void PopupDialogDEM(Widget w, Widget *wa, caddr_t call_data)
{
 if(!SPECIFIC)
 {
  gCursor.SetAppShellWidget(w);
  gCursor.DontEnter();
 }
 fileState(False);
 setGisType(Dem);

 // Set title and filter to appropriate values
 ChangeMask(wa[WI_FILESELBOXOPEN],"*.dem");
 XtVaSetValues(wa[WI_FCONV], XtNtitle,  "DEM Converter", NULL);

 // Do some initialization stuff which is common to the three
 // converters which use this particular dialog box.
 PopupDialogMisc(wa);

 XtManageChild(wa[WI_FILECONVPU]);
 MakeVisible(wa[WI_SELECTORPB]);
 MakeInvisible(wa[WI_LAYERSLBL]);
 MakeInvisible(wa[WI_LAYERSSL]);
}

void PopdownDialog(Widget w, Widget dialog, caddr_t call_data)
{
 int nothing = 0;
 if (dialog)
 {
  if(SPECIFIC)
    Quit(w, NULL, NULL);
  else
  {
    XtUnmanageChild(dialog);
    gCursor.SetAppShellWidget(w);
    gCursor.Normal();
  }
 }
 else
   XtUnmanageChild(XtParent(w));
}


void Quit(Widget w, caddr_t client_data, caddr_t call_data)
{
```

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


// ** The next few functions deal with the operation of the clipbox.
// ** The clipbox is used in the following dialogs:
// ** DEM, ERDAS, ARC, ERDAS/IMG

void PopupSelector(Widget w, Widget *wa, caddr_t call_data)
{
 Arg args[4];
 int n;
 XmString xms = (XmString) NULL;
 int WIDTH, HEIGHT;

 WIDTH  = getWidth();
 HEIGHT = getHeight();
 SCALE = 300.0/(float)MAX(WIDTH, HEIGHT);
 int width = (int) (SCALE * WIDTH), height = (int)(SCALE * HEIGHT);

 Display   *display;
 Colormap          colormap;
 XGCValues         values;
 Widget            canvas;
 Pixel             currentForeground, background;

 // Do nothing if no file selected
 if(!fileOpen()) return;
 // Insert code for getting width and height of picture.

 // Set width and height of drawing area.
  XtVaSetValues(wa[WI_DRAWINGAREA],XmNwidth, width,XmNheight, height,NULL);

 // Set Width label below drawing area.
    xms = XmStringCreate(itoa(WIDTH), XmSTRING_DEFAULT_CHARSET);
    XtVaSetValues(wa[WI_WIDTHLBL], XmNlabelString, xms, NULL);
    if (xms) XmStringFree(xms);

 // Set Height label right of drawing area.
    xms = XmStringCreate(itoa(HEIGHT), XmSTRING_DEFAULT_CHARSET);
    XtVaSetValues(wa[WI_HEIGHTLBL], XmNlabelString, xms, NULL);
    if(xms) XmStringFree(xms);

 // create the GC used by the rubber banding functions.
 canvas  = wa[WI_DRAWINGAREA];
 display = XtDisplay ( canvas );
 XtVaGetValues ( canvas, XmNcolormap, &colormap, NULL );

 XtVaGetValues (canvas,
          XmNforeground, &currentForeground,
          XmNbackground, &background,
          NULL );

 values.foreground = currentForeground ^ background;
 values.function   = GXxor;
 values.line_style = LineSolid;

 data.gc = XtGetGC (canvas,GCForeground | GCFunction | GCLineStyle,&values );

 data.last_x = data.start_x = 0;
 data.last_y = data.start_y = height;
 data.max_x = width; data.max_y = height;

 xms = XmStringCreate(" ", XmSTRING_DEFAULT_CHARSET);
 XtVaSetValues( wa[WI_XSIZELBL], XmNlabelString, xms, NULL);
 XtVaSetValues( wa[WI_YSIZELBL], XmNlabelString, xms, NULL);
 if (xms) XmStringFree(xms);

 PopupDialog(w, wa[WI_SELECTORPU], NULL);
}


void start_rubber_band(Widget w, Widget* wa, XEvent *event)
{
 XmString xms = (XmString) NULL;
 XPoint p[5];

 // Remove last selection.

 if(data.last_x != data.start_x || data.last_y != data.start_y)
 {
    p[0].x = data.start_x; p[0].y = data.start_y;
    p[1].x = data.start_x; p[1].y = data.last_y;
    p[2].x = data.last_x; p[2].y = data.last_y;
    p[3].x = data.last_x; p[3].y = data.start_y;
    p[4].x = data.start_x; p[4].y = data.start_y;

    XDrawLines(XtDisplay(w), XtWindow(w), data.gc, p, 5, CoordModeOrigin);
 }
```

```
 data.last_x = data.start_x = event->xbutton.x;
 data.last_y = data.start_y = event->xbutton.y;

 p[0].x = data.start_x; p[0].y = data.start_y;
 p[1].x = data.start_x; p[1].y = data.last_y;
 p[2].x = data.last_x; p[2].y = data.last_y;
 p[3].x = data.last_x; p[3].y = data.start_y;
 p[4].x = data.start_x; p[4].y = data.start_y;

 XDrawLines(XtDisplay(w), XtWindow(w), data.gc, p, 5, CoordModeOrigin);

 xms = XmStringCreate(" ", XmSTRING_DEFAULT_CHARSET);
 XtVaSetValues( wa[WI_XSIZELBL], XmNlabelString, xms, NULL);
 XtVaSetValues( wa[WI_YSIZELBL], XmNlabelString, xms, NULL);
 if (xms) XmStringFree(xms);
}

void track_rubber_band(Widget w, Widget* wa, XEvent *event)
{
 XmString xms = (XmString) NULL;
 XPoint p[5];

 p[0].x = data.start_x; p[0].y = data.start_y;
 p[1].x = data.start_x; p[1].y = data.last_y;
 p[2].x = data.last_x; p[2].y = data.last_y;
 p[3].x = data.last_x; p[3].y = data.start_y;
 p[4].x = data.start_x; p[4].y = data.start_y;

 XDrawLines(XtDisplay(w), XtWindow(w), data.gc, p, 5, CoordModeOrigin);

 data.last_x = event->xbutton.x;
 data.last_y = event->xbutton.y;

 p[0].x = data.start_x; p[0].y = data.start_y;
 p[1].x = data.start_x; p[1].y = data.last_y;
 p[2].x = data.last_x; p[2].y = data.last_y;
 p[3].x = data.last_x; p[3].y = data.start_y;
 p[4].x = data.start_x; p[4].y = data.start_y;

 XDrawLines(XtDisplay(w), XtWindow(w), data.gc, p, 5, CoordModeOrigin);

 xms = XmStringCreate(between((int)(data.start_x/SCALE),(int)(data.last_x/SCALE), 1),
            XmSTRING_DEFAULT_CHARSET);
 XtVaSetValues( wa[WI_XSIZELBL], XmNlabelString, xms, NULL);
 if (xms) XmStringFree(xms);

 if(gisType() == Dem)
  xms = XmStringCreate(between((int)((data.max_y - data.start_y)/SCALE),
            (int)((data.max_y - data.last_y)/SCALE), 0),
            XmSTRING_DEFAULT_CHARSET);
 else
  xms = XmStringCreate(between((int)((data.start_y)/SCALE),
            (int)((data.last_y)/SCALE), 0),
            XmSTRING_DEFAULT_CHARSET);
 XtVaSetValues( wa[WI_YSIZELBL], XmNlabelString, xms, NULL);
 if (xms) XmStringFree(xms);
}

void end_rubber_band(Widget w, Widget* wa, XEvent *event)
{
 XmString xms = (XmString) NULL;

 xms = XmStringCreate(between((int)(data.start_x/SCALE), (int)(data.last_x/SCALE), 1),
            XmSTRING_DEFAULT_CHARSET);
 XtVaSetValues( wa[WI_XSIZELBL], XmNlabelString, xms, NULL);
 if (xms) XmStringFree(xms);

 if(gisType() == Dem)
  xms = XmStringCreate(between((int)((data.max_y - data.start_y)/SCALE),
            (int)((data.max_y - data.last_y)/SCALE), 0),
            XmSTRING_DEFAULT_CHARSET);
 else
  xms = XmStringCreate(between((int)((data.start_y)/SCALE),
            (int)((data.last_y)/SCALE), 0),
            XmSTRING_DEFAULT_CHARSET);
 XtVaSetValues( wa[WI_YSIZELBL], XmNlabelString, xms, NULL);
 if (xms) XmStringFree(xms);
}

void clear_rubber_band(Widget w, Widget* wa, caddr_t call_data)
{
 XPoint p[5];

 XClearWindow(XtDisplay(w), XtWindow(w));

 p[0].x = data.start_x; p[0].y = data.start_y;
 p[1].x = data.start_x; p[1].y = data.last_y;
 p[2].x = data.last_x; p[2].y = data.last_y;
 p[3].x = data.last_x; p[3].y = data.start_y;
 p[4].x = data.start_x; p[4].y = data.start_y;
```

```
 XDrawLines(XtDisplay(w), XtWindow(w), data.gc, p, 5, CoordModeOrigin);
}

// This callbacks get called when the user finishes clipping.
// It determines the min/max values and sets the min/max edit boxes
// in the dialog.

void ReturnSelector(Widget w, Widget* wa, caddr_t call_data)
{
 int minx,miny,maxx,maxy;
 int WIDTH, HEIGHT;

 WIDTH  = getWidth();
 HEIGHT = getHeight();

 if (gisType() == Dem)
 {
  minx = (int)(MAX(0, MIN(data.start_x, data.last_x))/SCALE);
  miny = (int)(MAX(0, MIN(data.max_y - data.start_y,data.max_y - data.last_y))/SCALE);
  maxx = (int)(MIN(WIDTH,MAX(data.start_x, data.last_x))/SCALE);
  maxy = (int)(MIN(HEIGHT, MAX(data.max_y - data.start_y,data.max_y - data.last_y))/SCALE);
 }
 else
 {
  minx = (int)(MAX(0, MIN(data.start_x, data.last_x))/SCALE);
  miny = (int)(MAX(0, MIN(data.start_y,data.last_y))/SCALE);
  maxx = (int)(MIN(WIDTH,MAX(data.start_x, data.last_x))/SCALE);
  maxy = (int)(MIN(HEIGHT, MAX(data.start_y,data.last_y))/SCALE);
 }
 if (maxx > WIDTH)  maxx = WIDTH;
 if (maxy > HEIGHT) maxy = HEIGHT;
 if (minx < 0)      minx = 0;
 if (miny < 0)      miny = 0;


 if (gisType() == IMG)
    SetSelectorValuesIMG(minx,miny,maxx,maxy);
 else if (gisType() == Arc)
    SetSelectorValuesARC(minx,miny,maxx,maxy);
 else
    SetSelectorValues(wa,minx,miny,maxx,maxy);

 PopdownDialog(w, NULL, NULL);
}
```

```
void SetSelectorValues(Widget *wa, int minx, int miny, int maxx, int maxy)
{
   XmTextSetString(wa[WI_MINXTXT], itoa(minx));
   XmTextSetString(wa[WI_MINYTXT], itoa(miny));
   XmTextSetString(wa[WI_MAXXTXT], itoa(maxx));
   XmTextSetString(wa[WI_MAXYTXT], itoa(maxy));
}

// This is the open file function.  It open files for three converters.
// Those three converters are: DEM, ERDAS, MODFLOW.
void OpenFile(Widget w, Widget* WA, XmListCallbackStruct *list)
{
 char *filename;
 ErrorBox eb(w);

 XmStringGetLtoR(list->item, XmSTRING_DEFAULT_CHARSET, &filename);
 ifstream in(filename);
 if(!in)
 {
    in.close();
    ResetFileOpenBox(WA);

    eb.Error("Error: Can not read input file!");
    return;
 }

 fileState(True);
 if(gisType() == Dem)
    OpenFileDem(w,WA,in);
 else if(gisType() == MOD)
    OpenFileMOD(w,WA,in);
 else if(gisType() == LAN)
    OpenFileLAN(w,WA,in);
}

void OpenFileDem(Widget w, Widget *WA, ifstream& in)
{
 ErrorBox eb(w);
 int WIDTH, HEIGHT;

 if(!get_DEM_info(in, WIDTH, HEIGHT))
 {
    in.close();
    setWidth(WIDTH);
    setHeight(HEIGHT);
```

```
    PlaceDefault(w,WA,NULL);
  }
  else
  {
    in.close();
    ResetFileOpenBox(WA);
    ClearMinMax(WA);

    eb.Error("Error: Not a DEM file!");
  }
}

void OpenFileMOD(Widget w, Widget *WA, ifstream& in)
{
  ErrorBox eb(w);
  int WIDTH, HEIGHT;

  if(!get_MODFLOW_info(in, WIDTH, HEIGHT))
  {
    in.close();
    setWidth(WIDTH);
    setHeight(HEIGHT);
    PlaceDefault(w,WA,NULL);
  }
  else
  {
    in.close();
    ResetFileOpenBox(WA);
    ClearMinMax(WA);

    eb.Error("Error: Not a MODFLOW file!");
  }
}

void OpenFileLAN(Widget w, Widget *WA, ifstream& in)
{
  ErrorBox eb(w);
  char buf[20];
  int WIDTH, HEIGHT;

  XmListDeleteAllItems(WA[WI_LAYERSSL]);
  if(!get_ERDAS_info(in, WIDTH, HEIGHT, BANDS))
  {
    in.close();
    setWidth(WIDTH);
    setHeight(HEIGHT);
    PlaceDefault(w,WA,NULL);

    XmString str = (XmString)NULL;
    int i=0;
    for(i = 0; i < BANDS; i++)
    {
     sprintf(buf, "Band %d", i+1);
     str = XmStringCreateLtoR(buf, XmSTRING_DEFAULT_CHARSET);

     XmListAddItem(WA[WI_LAYERSSL],str,0);
     XmListSelectPos(WA[WI_LAYERSSL],i+1,False);

     if (str) XmStringFree(str);
    }
  }
  else
  {
    in.close();
    ResetFileOpenBox(WA);
    ClearMinMax(WA);

    eb.Error("Error: Not an ERDAS file!");
  }
}

void ResetFileOpenBox(Widget *wa)
{
  Widget tmp;

  fileState(False);

  tmp = XmSelectionBoxGetChild(wa[WI_FILESELBOXOPEN],XmDIALOG_FILE_LIST);
  XmListDeselectAllItems(tmp);

                                        t          m          p                                    =
XmSelectionBoxGetChild(wa[WI_FILESELBOXOPEN],XmDIALOG_APPLY_BUTTON);
  XtCallCallbacks(tmp, XmNactivateCallback, NULL);
}

void PlaceDefault(Widget w, Widget* WA, caddr_t list)
{
  XmTextSetString(WA[WI_MINXTXT], "0");
  XmTextSetString(WA[WI_MINYTXT], "0");
  XmTextSetString(WA[WI_MAXXTXT], itoa( getWidth() ));
```

```
  XmTextSetString(WA[WI_MAXYTXT], itoa( getHeight() ));
}

void ClearMinMax(Widget *WA)
{
  XmTextSetString(WA[WI_MINXTXT], "");
  XmTextSetString(WA[WI_MINYTXT], "");
  XmTextSetString(WA[WI_MAXXTXT], "");
  XmTextSetString(WA[WI_MAXYTXT], "");
}

// This is used by all three converters (DEM, ERDAS, MODFLOW).
// This is where the dx info get generated.
void OkCallBack(Widget w, Widget* WA, XmFileSelectionBoxCallbackStruct *cbs)
{
 char   *file1, *file2;
 Boolean  isFile;
 ErrorBox eb(w);

 Widget tmpw1;

 tmpw1 = XmSelectionBoxGetChild(WA[WI_FILESELBOXOPEN],XmDIALOG_TEXT);
 file1 = XmTextGetString(tmpw1);

 tmpw1 = XmSelectionBoxGetChild(WA[WI_FILESELBOXCONV],XmDIALOG_TEXT);
 file2 = XmTextGetString(tmpw1);

 isFile = XmToggleButtonGetState(WA[WI_FILESAVETB]);
/** Check out the output file name (if using file output).
 ** if it exists?, do we overwrite.
 ** add a .dx extension if there is no extension present.
**/
 if (!fileOpen())
 {
  eb.Error("No input file selected!");
  return;
 }
 else if (!FileExists(file1))
 {
  eb.Error("Input file does not exist or is unreadable!");
  return;
 }

 if (isFile)
 {
  if (!validFile(file2,WA[WI_FILESELBOXCONV],w))
  {
   eb.Error("NO OUTPUT FILE SELECTED!");
   return;
  }
  else if (FileExists(file2) && (!overwrite(w)))
    return;
 }
 // call the conversion
 OtherRun(w, WA, file1, file2, isFile);
}

void OtherRun(Widget w, Widget* WA, char* file1, char* file2, Boolean isFile)
{
 CursorClass cursor;
 char* xy[4];
 int sx, sy, ex, ey, binary;
 Boolean isBinary;
 ErrorBox eb(w);

 cursor.SetShellWidget(WA[WI_FCONV]);
 cursor.Wait();
 XFlush(XtDisplay(w));

 isBinary = XmToggleButtonGetState(WA[WI_BINARYTB]);

 ifstream in(file1);
 if(!in)
 {
   eb.Error("Error: Can not read input file!");
   cursor.Normal();
   return;
 }
 ofstream out;

 if(!isFile)
   out.attach(1);
 else
   out.open(file2);

 if(!out)
 {
   in.close();
   eb.Error("Error: Can not write output file!");
   cursor.Normal();
```

```
    return;
  }
  binary = (isBinary) ? 1 : 0;

  xy[0] = XmTextGetString(WA[WI_MINXTXT]);
  xy[1] = XmTextGetString(WA[WI_MINYTXT]);
  xy[2] = XmTextGetString(WA[WI_MAXXTXT]);
  xy[3] = XmTextGetString(WA[WI_MAXYTXT]);

  sx = atoi(xy[0]);
  ex = atoi(xy[2]);
  sy = atoi(xy[1]);
  ey = atoi(xy[3]);
  if(gisType() == Dem)
  {
    DEM test(in, out, sx, ex, sy, ey, binary);
  }
  else if(gisType() == LAN)
  {
    int *bands, numbds;
    if(XmListGetSelectedPos(WA[WI_LAYERSSL], &bands, &numbds))
    {
     ERDAS test(in, out, sx, ex, sy, ey, binary, numbds, bands);
    }
    else
    {
     eb.Error("Error: No Bands Selected to Convert!");
     cursor.Normal();
     return;
    }
  }
  else if(gisType() == MOD)
    MODFLOW(in, out, binary);

  out.flush();

  in.close();
  if (!isFile)
   out.detach();
  else
   out.close();

  if(PERSISTENT) Quit(WA[0], NULL, NULL);
  cursor.Normal();
}
```

```
// Hide or Show the file save box as is appropriate.
// Note: the button state has already changed when this
//  function is reached.
void ToggleMeFile (Widget widget, Widget savebox, XmAnyCallbackStruct *call_data)
{
 Boolean isFile;

 isFile = XmToggleButtonGetState(widget);
 if(!isFile)
    XtUnmanageChild(savebox);
 else
    XtManageChild(savebox);
}

// This procedure displays help in response to the help button
// being pressed in the main application window.
void HelpTop(Widget w, Widget *WA, XmAnyCallbackStruct *call_data)
{
   char buf[600];
   buf[0] = '\0';

/*
   if (!fork())
   {
    system("netscape www.cs.umt.edu");
    _exit(0);
   }
   return;
*/
   HelpBox h(w, 300);

   strcat(buf,"Choose the type of file to convert:\n\n\n");
   strcat(buf,"  DEM              - Digital Elevation Model (ASCII)\n\n");
   strcat(buf,"  DLG              - Digital Line Graph (ASCII)\n\n");
   strcat(buf,"  ERDAS            - ERDAS LAN/GIS Raster File (Binary)\n\n");
   strcat(buf,"  MODFLOW          - Modflow Heads/Drawdown File (Binary)\n\n");
   strcat(buf,"  ARC/INFO         - Grid or Coverage File (ASCII)\n\n");
   strcat(buf,"  ARCSHAPE         - Arc Shapefiles (Binary)\n\n");
   strcat(buf,"  ORACLE           - Oracle Tables to dx\n\n");
   strcat(buf,"  DXF              - DXF (AutoCad) (ASCII)\n\n");
   strcat(buf,"  ERDAS/IMAGINE - Erdas Imagine files (Binary)");
   h.Help(buf);
}
```

```
// Help for the clip box.
void HelpSelector(Widget w, Widget *WA, XmAnyCallbackStruct *call_data)
{
 HelpBox h(w, 80);
 h.Help("Click and Drag to select the portion of the file to be converted to DX format file.  If you are
unsatisfied with you selection, repeat until you are satisfied.");
}

// Displays help for the following converter dialogs:
// DEM, ERDAS, MODFLOW, ARC.
void HelpAll(Widget w, Widget *WA, XmAnyCallbackStruct *call_data)
{
 char buf[2000];
 int position;
 HelpBox h(w, 150);
 int ConvType;

 ConvType = gisType();

 switch(ConvType)
 {
  case Dem:  strcpy(buf,"DEM ");  break;

  case LAN:  strcpy(buf,"LAN ");  break;

  case MOD:  strcpy(buf,"MOD ");  break;

  case Arc:  strcpy(buf,"Arc ");  break;
 }

  strcat(buf,"Conversion Help\n\n");
  strcat(buf,"   Left File Selection Box -- Used to choose the input file for the conversion.\n");
  strcat(buf,"     OK Button  -- Used to start the conversion.\n");
  strcat(buf,"     Filter        -- Used to filter files in directory.\n");
  strcat(buf,"     Quit          -- Quit the current conversion session.\n");
  strcat(buf,"     Help          -- Pop up this help.\n\n");

  strcat(buf,"   Stats and Options         -- Input file information and user options\n");
  strcat(buf,"     Min X, Min Y, Max X, Max Y -- Grid size information about the input file.\n");
   strcat(buf,"     Default     -- Allow user to recover input file default information if changes have
been made.\n");


 if (ConvType != MOD)
  strcat(buf,"     Clip          -- Allow user to only convert a portion of the input file.\n");

 if(ConvType == LAN)
    strcat(buf,"       Layers List  -- Allow user to select which bands of LAN/GIS input file to
convert.\n");

 else if(ConvType == Arc)
 {
  strcat(buf,"     Grid          -- Allow user to filter through only Arc grid files \n");
  strcat(buf,"                       If this option is selected, the MinMax options and the Clip button will
show. \n\n");
  strcat(buf,"     Coverage  -- Allow user to filter through only Arc coverage files \n");
  strcat(buf,"                    If this option is selected, the attributes box will show, but MinMax and
Clip options will be disabled. \n");
  strcat(buf,"     Attributes -- Allow user to pick only certain attributes from the coverage.\n");
 }

  strcat(buf,"\n\n");
   strcat(buf,"     Binary OutputToggle  -- Allow the user to choose between ASCII or Binary data
format in output DX file.\n");
   strcat(buf,"     File Output Toggle     -- Allow the user to choose between file or standard output
for the conversion.\n\n");

  strcat(buf,"   Right File Selection Box (only visible if file output is selected) -- Used to choose the
output file for the conversion.\n");
  strcat(buf,"     Filter -- Used to filter files in directory.\n\n");

 if (ConvType == Arc)
 {
  strcat(buf,"\nNote: (Grids)\n");
  strcat(buf,"     Min, Max, Mean and Standard Dev. are written as comments to the output.\n");
 }


 h.Help(buf);
}
```

```
// DLG Converters
//////////////////////////////////////////////////////////////////////
//   File:          callbacksDlg.h
//   Author:        David Thompson, Dick Thompson, Petr Votava
//                  Kirk Moeller, and others
//   Description:
//                  Header file specifying prototypes for the
//                  callback functions for the Converters listed
//                  above.
//////////////////////////////////////////////////////////////////////

#ifndef CALLBACKSDLG_H
#define CALLBACKSDLG_H

#include "XHeaders.h"
#include <fstream.h>

void AddCallbacksDLG(Widget w[]);

void PopupDialogDLG(Widget w, Widget *wa, caddr_t call_data);
void PopdownDialogDLG(Widget w, Widget dialog, caddr_t call_data);
void DlgQuit(Widget w, caddr_t client_data, caddr_t call_data);

void OpenFileDLG(Widget w, Widget* WA, XmListCallbackStruct *list);

void OkCallBackDLG(Widget w, Widget* WA, XmFileSelectionBoxCallbackStruct *cbs);
void DLGRun(Widget w, Widget* WA, char* file1, char* file2, Boolean isFile);

void ToggleMeFileDLG(Widget widget, Widget savebox, XmAnyCallbackStruct *call_data);

void ResetFileOpenBoxDLG(Widget *wa);
void HelpDLG(Widget w, Widget *WA, XmAnyCallbackStruct *call_data);

#endif
```

```
// DLG Converters
//////////////////////////////////////////////////////////////////////
//   File:          callbacksDlg.C
//   Author:        David Thompson, Dick Thompson, Petr Votava,
//                  Kirk Moeller, and others.
//   Description:
//                  Implementation of all of the callbacks
//                  functions necessary to implement all of the
//                  operations of the dialogs associated with the
//                  converters list above.
//////////////////////////////////////////////////////////////////////

#include <unistd.h>
#include <iostream.h>

#include "callbacksDlg.h"
#include "convW.h"

#include "dlg2dx.h"

#include "cursor.h"
#include "errorbox.h"
#include "callutils.h"
#include "globals.h"

CursorClass Dlg_gCursor;

void AddCallbacksDLG(Widget w[])
{
  Widget tmpwI;
  Atom   WM_DELETE_WINDOW;

// FCONVI callbacks (The DLG converter callbacks)
   XtAddCallback(w[WI_DLGPB], XmNactivateCallback,
            (XtCallbackProc) PopupDialogDLG, w);
   XtAddCallback(w[WI_FILESELBOXOPEN1], XmNcancelCallback,
            (XtCallbackProc) PopdownDialogDLG, w[WI_FILECONVPU1]);

   tmpw1 = XmSelectionBoxGetChild(w[WI_FILESELBOXOPEN1],XmDIALOG_FILE_LIST);
   XtAddCallback(tmpw1, XmNbrowseSelectionCallback,
            (XtCallbackProc) OpenFileDLG, w);

   tmpw1 = XmSelectionBoxGetChild(w[WI_FILESELBOXCONV1],XmDIALOG_TEXT);
   XtAddCallback(tmpw1, XmNactivateCallback,(XtCallbackProc) OkCallBackDLG, w);
   XtAddCallback(w[WI_FILESELBOXOPEN1], XmNhelpCallback,
            (XtCallbackProc) HelpDLG, w);
   XtAddCallback(w[WI_FILESELBOXOPEN1], XmNokCallback,
            (XtCallbackProc) OkCallBackDLG, w);
   XtAddCallback(w[WI_FILESAVETB1], XmNvalueChangedCallback,
            (XtCallbackProc) ToggleMeFileDLG, w[WI_RFORM_DLG]);

   WM_DELETE_WINDOW = XmInternAtom(XtDisplay(w[WI_FCONV1]),
                        "WM_DELETE_WINDOW",False);
   XmAddWMProtocolCallback(w[WI_FCONV1],WM_DELETE_WINDOW,
           (XtCallbackProc)PopdownDialogDLG,w[WI_FILECONVPU1]);
}

void PopupDialogDLG(Widget w, Widget *wa, caddr_t call_data)
{
 if(!SPECIFIC)
 {
  Dlg_gCursor.SetAppShellWidget(w);
  Dlg_gCursor.DontEnter();
 }
 fileState(False);
 setGisType(Dlg);

 if(PERSISTENT || FILTER)
 {
    XtUnmanageChild(wa[WI_RFORM_DLG]);
    XmToggleButtonSetState(wa[WI_FILESAVETB1],False,False);
 }

 XmTextSetString(wa[WI_STATTXT], "");
 XtManageChild(wa[WI_FILECONVPU1]);
}

void PopdownDialogDLG(Widget w, Widget dialog, caddr_t call_data)
{
 int nothing = 0;
 if (dialog)
 {
  if(SPECIFIC)
    DlgQuit(w, NULL, NULL);
  else
  {
    XtUnmanageChild(dialog);
    Dlg_gCursor.SetAppShellWidget(w);
    Dlg_gCursor.Normal();
```

```
  }
 }
 else
  XtUnmanageChild(XtParent(w));
}

void DlgQuit(Widget w, caddr_t client_data, caddr_t call_data)
{
 XCloseDisplay(XtDisplay(w));
 exit(0);
}

void OpenFileDLG(Widget w, Widget* WA, XmListCallbackStruct *list)
{
 int nodes, lines, areas, areainfo, lineinfo;
 char name[41], type[21], buf[150], temp[15];
 char *filename;
 ErrorBox eb(w);

 XmStringGetLtoR(list->item, XmSTRING_DEFAULT_CHARSET, &filename);
 ifstream in(filename);
 if(!in)
 {
   in.close();
   ResetFileOpenBoxDLG(WA);

   eb.Error("Error: Can not read input file!");
   return;
 }

 fileState(True);

 if(!get_DLG_info(in, name, type, nodes, lines, areas, areainfo, lineinfo))
 {
   in.close();
   (areainfo)? sprintf(temp, "present") : sprintf(temp, "not present");
    sprintf(buf, "Area name:\n%s\n\nMap type:\n%s\n\n Nodes: %d\n Lines: %d\n Areas: %d\n\n
Polygon info: %s",
         name, type, nodes, lines, areas, temp);

   XmTextSetString(WA[WI_STATTXT], buf);
 }
 else
 {
   ResetFileOpenBoxDLG(WA);
   in.close();
   buf[0] = 0;
   XmTextSetString(WA[WI_STATTXT], buf);
   eb.Error("Error: Not a DLG file!");
 }
}

void OkCallBackDLG(Widget w, Widget* WA, XmFileSelectionBoxCallbackStruct *cbs)
{
 char   *file1, *file2;
 Boolean  isFile;
 ErrorBox eb(w);

 Widget tmpw1;

 tmpw1 = XmSelectionBoxGetChild(WA[WI_FILESELBOXOPEN1],XmDIALOG_TEXT);
 file1 = XmTextGetString(tmpw1);

 tmpw1 = XmSelectionBoxGetChild(WA[WI_FILESELBOXCONV1],XmDIALOG_TEXT);
 file2 = XmTextGetString(tmpw1);

 isFile = XmToggleButtonGetState(WA[WI_FILESAVETB1]);
/** Check out the output file name (if using file output).
 ** if it exists?, do we overwrite.
 ** add a .dx extension if there is no extension present.
**/
 if (!fileOpen())
 {
  eb.Error("No input file selected!");
  return;
 }
 else if (!FileExists(file1))
 {
  eb.Error("Input file does not exist or is unreadable!");
  return;
 }

 if (isFile)
 {
   if (!validFile(file2,WA[WI_FILESELBOXCONV1],w))
   {
    eb.Error("NO OUTPUT FILE SELECTED!");
    return;
   }
   else if (FileExists(file2) && (!overwrite(w)))
```

```
      return;
 }
 // call the conversion
 DLGRun(w, WA, file1, file2, isFile);
}

void DLGRun(Widget w, Widget* WA, char* file1, char* file2, Boolean isFile)
{
 int binary;
 CursorClass cursor;
 Boolean isBinary;
 ErrorBox eb(w);

 cursor.SetShellWidget(WA[WI_FCONV1]);
 cursor.Wait();
 XFlush(XtDisplay(w));

 isBinary = XmToggleButtonGetState(WA[WI_BINARYTB1]);

 ifstream in(file1);
 if(!in)
 {
   eb.Error("Error: Can not read input file!");
   return;
 }

 ofstream out;
 if(!isFile)
   out.attach(1);
 else
   out.open(file2);

 if(!out)
 {
   in.close();
   eb.Error("Error: Can not write output file!");
   return;
 }
 binary = (isBinary) ? 1 : 0;
 DLG test(in, out, binary);
 out.flush();

 in.close();
 if (!isFile)
  out.detach();
 else
  out.close();


 if(PERSISTENT) DlgQuit(WA[0], NULL, NULL);
 cursor.Normal();
}

// Hide or Show the file save box as is appropriate.
// Note: the button state has already changed when this
//  function is reached.
void ToggleMeFileDLG(Widget widget, Widget savebox, XmAnyCallbackStruct *call_data)
{
 Boolean isFile;

 isFile = XmToggleButtonGetState(widget);
 if(!isFile)
   XtUnmanageChild(savebox);
 else
   XtManageChild(savebox);
}

void ResetFileOpenBoxDLG(Widget *wa)
{
 Widget tmp;

 fileState(False);

 tmp = XmSelectionBoxGetChild(wa[W1_FILESELBOXOPEN1],XmDIALOG_FILE_LIST);
 XmListDeselectAllItems(tmp);

                                    t          m          p                                    =
XmSelectionBoxGetChild(wa[W1_FILESELBOXOPEN1],XmDIALOG_APPLY_BUTTON);
 XtCallCallbacks(tmp, XmNactivateCallback, NULL);
}

void HelpDLG(Widget w, Widget *WA, XmAnyCallbackStruct *call_data)
{
 char buf[2000];
 int position;
 HelpBox h(w, 150);

 strcat(buf,"Dlg Conversion Help\n\n");
 strcat(buf,"  Left File Selection Box -- Used to choose the input file for the conversion.\n");
 strcat(buf,"     OK Button  -- Used to start the conversion.\n");
```

```
 strcat(buf,"     Filter        -- Used to filter files in directory.\n");
 strcat(buf,"     Quit          -- Quit the current conversion session.\n");
 strcat(buf,"     Help          -- Pop up this help.\n\n");

 strcat(buf,"   Stats and Options         -- Input file information and user options\n");

 strcat(buf,"     Statistics  -- Node, line, and area information about the input DLG file.\n");
 strcat(buf,"     Default   -- Allow user to recover input file default information if changes have been
made.\n");

 strcat(buf,"\n\n");
 strcat(buf,"      Binary OutputToggle  -- Allow the user to choose between ASCII or Binary data
format in output DX file.\n");
 strcat(buf,"     File Output Toggle     -- Allow the user to choose between file or standard output for
the conversion.\n\n");

 strcat(buf,"   Right File Selection Box (only visible if file output is selected) -- Used to choose the
output file for the conversion.\n");
 strcat(buf,"     Filter -- Used to filter files in directory.\n\n");

 h.Help(buf);
}
```

```
// SHAPE Converter
//////////////////////////////////////////////////////////////////////
// File:          shp_callbacks.h
// Author:        Kirk A. Moeller
// Description:
//                Header file for shape converter callbacks.
//////////////////////////////////////////////////////////////////////

#ifndef SHAPECALLBACKS_H
#define SHAPECALLBACKS_H

#include "XHeaders.h"
#include "Xshp2dx.h"
#include "errorbox.h"

/* PROTOTYPES */
void AddShapeCallbacks(Widget w[], Widget shapepb);
void PopupDialogSHAPE(Widget w, Widget *wa, caddr_t call_data);
void PopupDialogSeries(Widget w, Widget *wa, caddr_t call_data);
void PopdownDialogShape(Widget w, Widget dialog, caddr_t call_data);
void ShpQuit(Widget w, caddr_t client_data, caddr_t call_data);
void HelpShape(Widget w, Widget *WA, XmAnyCallbackStruct *call_data);
void OverWrite(Widget w, Widget* WA, XmFileSelectionBoxCallbackStruct *cbs);
void LinesToggle(Widget w, Widget *WA, caddr_t call_data);
void PolylinesToggle(Widget w, Widget *WA, caddr_t call_data);
void OpenFileShape(Widget w, Widget *wa, XmListCallbackStruct *list);
int GetSelectedNames(ErrorBox *eb,FIELD selected[], Widget wa[]);
void updateDialog(Widget attriblbl, Widget label, Widget linesRB);
void ShapeOkCallBack(Widget w, Widget *wa, XmFileSelectionBoxCallbackStruct *cbs);
void ShapeApplyCallBack(Widget w, Widget *wa, XmFileSelectionBoxCallbackStruct *cbs);
void ToggleMeFileShape (Widget widget, Widget *wa, XmAnyCallbackStruct *call_data);

void SeriesToggle_Shape(Widget w, Widget *wa, caddr_t call_data);

#endif
```

```
// SHAPE Converter
//////////////////////////////////////////////////////////////////////
//   File:          shp_callbacks.C
//   Author:        Kirk A. Moeller
//   Description:
//                  Implementation of the callbacks functions
//                  which respond to user actions in the
//                  shape converter dialog.
//////////////////////////////////////////////////////////////////////

#include <fstream.h>
#include <st
#include <unistd.h>
#include "shp2dxW.h"
#include "shp_callbacks.h"
#include "cursor.h"
#include "globals.h"
#include "seriesW.h"
#include "seriesCall.h"
#include "callutils.h"

/* globals */
CursorClass shp_gCursor;

Widget      *serieswarr;
SeriesDialog  *seriesdlg;
Boolean           NEWLIST;

/* shape converter globals */
  char          *outfile;
  char          shpfile[200];
  char          dbffile[200];
  short             len[NumFields];
  FIELD             fields[NumFields];
  int               NumOfRecords;
  short             DataOffset;
  int               numfields;
  extern SHAPETYPE shape;   // DEFINED IN Xshp2dx.C

// This procedure registers the calllbacks for the widget in
// the shape converter dialog box.

void AddShapeCallbacks(Widget w[],Widget shapepb)
{
  Widget tmpw1;
  Atom   WM_DELETE_WINDOW;

  XtAddCallback(shapepb, XmNactivateCallback,
          (XtCallbackProc) PopupDialogSHAPE, w);
  XtAddCallback(w[WI_TB_SHAPE], XmNarmCallback,
          (XtCallbackProc) LinesToggle, w);
  XtAddCallback(w[WI_TB1_SHAPE], XmNarmCallback,
          (XtCallbackProc) PolylinesToggle, w);
  XtAddCallback(w[WI_FILESELBOXOPEN_SHAPE], XmNcancelCallback,
          (XtCallbackProc) PopdownDialogShape, w[WI_FILECONVPU_SHAPE]);

                                t       m       p       w       l                       =
XmSelectionBoxGetChild(w[WI_FILESELBOXOPEN_SHAPE],XmDIALOG_FILE_LIST);
  XtAddCallback(tmpw1, XmNbrowseSelectionCallback,
          (XtCallbackProc) OpenFileShape, w);
  tmpw1 = XmSelectionBoxGetChild(w[WI_FILESELBOXCONV_SHAPE],XmDIALOG_TEXT);
  XtAddCallback(tmpw1, XmNactivateCallback,
          (XtCallbackProc) ShapeOkCallBack, w);

  XtAddCallback(w[WI_FILESELBOXOPEN_SHAPE], XmNokCallback,
          (XtCallbackProc) ShapeOkCallBack, w);
  XtAddCallback(w[WI_FILESELBOXOPEN_SHAPE], XmNapplyCallback,
          (XtCallbackProc) ShapeApplyCallBack, w);

  XtAddCallback(w[WI_FILETB_SHAPE], XmNvalueChangedCallback,
          (XtCallbackProc) ToggleMeFileShape, w);

  XtAddCallback(w[WI_FILESELBOXOPEN_SHAPE], XmNhelpCallback,
          (XtCallbackProc) HelpShape, w);
  XtAddCallback(w[WI_ORDERDLGPB_SHAPE], XmNactivateCallback,
          (XtCallbackProc) PopupDialogSeries, w);
  XtAddCallback(w[WI_CREATESERIESTB_SHAPE], XmNvalueChangedCallback,
          (XtCallbackProc) SeriesToggle_Shape, w);

  WM_DELETE_WINDOW = XmInternAtom(XtDisplay(w[WI_FCONV_SHAPE]),
                    "WM_DELETE_WINDOW", False);
  XmAddWMProtocolCallback(w[WI_FCONV_SHAPE],WM_DELETE_WINDOW,
          (XtCallbackProc)PopdownDialogShape,w[WI_FILECONVPU_SHAPE]);


/** Create The Series Dialog Box **/
/** This is done here, to insure the dialog is create only once. **/
  tu_seriesdlg_widget("Order Time Series",
                w[WI_FCONV_SHAPE],&serieswarr);
```

```
 seriesdlg = new SeriesDialog(serieswarr);
}


/***************************** SHAPE CALLBACKS ************************/

void PopupDialogSHAPE(Widget w, Widget *wa, caddr_t call_data)
{
 XmString xms = (XmString) NULL;

 if(!SPECIFIC)
   {
    shp_gCursor.SetAppShellWidget(w);
    shp_gCursor.DontEnter();
   }

// give the file boxes the appropriate filters.
 if(DIR_FLAG)
 {
    xms = XmStringCreate(DEF_DIR, XmSTRING_DEFAULT_CHARSET);
    XtVaSetValues(wa[WI_FILESELBOXOPEN_SHAPE], XmNdirectory, xms, NULL);
    XtVaSetValues(wa[WI_FILESELBOXCONV_SHAPE], XmNdirectory, xms, NULL);
    if (xms) XmStringFree(xms);
 }

// Change dialog to standard out mode if these conditions are true.
 if (PERSISTENT || FILTER)
 {
  XtUnmanageChild(wa[WI_RFORM_SHAPE]);
  XmToggleButtonSetState(wa[WI_FILETB_SHAPE],False,False);
 }

 /** Time series Option is initially NO! **/
 XmToggleButtonSetState(wa[WI_CREATESERIESTB_SHAPE],False,False);
 SeriesToggle_Shape(wa[WI_CREATESERIESTB_SHAPE],wa,NULL);

 XmListDeleteAllItems(wa[WI_ATTRIBLST_SHAPE]);

 xms = XmStringCreate("Attributes:", XmSTRING_DEFAULT_CHARSET);
 XtVaSetValues(wa[WI_ATTRIBLBL_SHAPE],XmNlabelString,xms,NULL);
 if (xms) XmStringFree(xms);

 XtManageChild(wa[WI_FILECONVPU_SHAPE]);
 fileState(False);
}
```

```
void PopupDialogSeries(Widget w, Widget *wa, caddr_t call_data)
{
 static XmStringTable Attribs;
 static int       nAttribs;
 ErrorBox eb(w);

 if (!fileOpen())
 {
  eb.Error("Please Open a Shape File first.");
  return;
 }

 if (NEWLIST)
 {
  XtVaGetValues(wa[WI_ATTRIBLST_SHAPE],
          XmNitems,&Attribs,
          XmNitemCount, &nAttribs, NULL);
  seriesdlg->setAttribList(Attribs,nAttribs);
 }
 seriesdlg->PopupSeriesDialog(w);

 if (NEWLIST) NEWLIST = False;
}


// These two functions toggle wether lines or polyline info
// is generated.
void LinesToggle(Widget w, Widget *wa, caddr_t call_data)
{
 SetLinesValue(True);
}



void PolylinesToggle(Widget w, Widget *wa, caddr_t call_data)
{
 SetLinesValue(False);
}


// Open the input shape file.
// In this case that consists of reading the dbase file and
// getting the attribute names.
// These names are then displayed in the list box.
void OpenFileShape(Widget w, Widget *wa, XmListCallbackStruct *list)
```

```
{
 CursorClass cursor;
 char *filename;
 XmString xms = (XmString) NULL;
 int i;
 Widget tmpw1;
 char *pstr;
 ifstream fin, fdbf;
 ErrorBox eb(w);

 // desensitize the polylines/lines radio box.
 XtSetSensitive(wa[WI_LINESRBLBL_SHAPE],False);
 XtSetSensitive(wa[WI_LINESRB_SHAPE],False);

//*** INITIALIZE GLOBALS ***
 NumOfRecords =0;
 DataOffset=0;
 numfields=0;

// Make sure series dialog lists are empty!
// Set a flag that will cause the dialog to reset
// when it is popped up.
// This flag is defined in seriesCall.C
 NEWLIST = True;

 cursor.SetShellWidget(wa[WI_FILECONVPU_SHAPE]);
 cursor.Wait();
 XFlush(XtDisplay(w));


//*** GET INPUT FILE NAME ****
 XmStringGetLtoR(list->item, XmSTRING_DEFAULT_CHARSET, &filename);

 shpfile[0] = '\0';
 strcpy(shpfile,filename);

//*** MAKE FILE NAME ***
 strcpy(dbffile,shpfile);
 pstr = strstr(dbffile,".shp");
 *pstr = '\0';
 strcat(dbffile,".dbf");

//*************************
//*** CHECK INPUT FILE ***
 fdbf.open(dbffile);
 fin.open(shpfile);
 if ((!fin) || (!fdbf))
 {
  fin.close();
  fdbf.close();

                                                    t    m    p    w    1                =
XmSelectionBoxGetChild(wa[WI_FILESELBOXOPEN_SHAPE],XmDIALOG_FILE_LIST);
  XmListDeselectAllItems(tmpw1);

  eb.Error("Can't read input file!");
  return;
 }
 GetFileInfo(fin);
 fin.close();


updateDialog(wa[WI_ATTRIBLBL_SHAPE],wa[WI_LINESRBLBL_SHAPE],wa[WI_LINES
RB_SHAPE]);
 XmListDeleteAllItems(wa[WI_ATTRIBLST_SHAPE]);

//*** GET FIELD NAMES FROM DBASE FILE ****

 numfields = ProcessDbfHead(fdbf,len,fields,NumOfRecords,DataOffset);

 for(i = 0; i < numfields; i++)
 {
   xms = XmStringCreateLocalized(fields[i].name);
   XmListAddItemUnselected(wa[WI_ATTRIBLST_SHAPE],xms,0);
   if (xms)  XmStringFree(xms);
 }

 cursor.Normal();
 fileState(True);
}

void updateDialog(Widget attriblbl, Widget label, Widget linesRB)
{
 XmString xms = (XmString) NULL;

 SetLinesValue(False);
 // Change the shape file type label to the appropriate value.
 if (shape == arc)
 {
  //This is an arc shape file, so we want these toggles operational.
```

```
  XtSetSensitive(label,True);
  XtSetSensitive(linesRB,True);

  xms = XmStringCreate("Attributes: ARC", XmSTRING_DEFAULT_CHARSET);
  SetLinesValue(True);
 }
 else if (shape == polygon)
  xms = XmStringCreate("Attributes: POLYGON", XmSTRING_DEFAULT_CHARSET);
 else if (shape == multipoint)
  xms = XmStringCreate("Attributes: MULTIPOINT", XmSTRING_DEFAULT_CHARSET);
 else
  xms = XmStringCreate("Attributes: POINT", XmSTRING_DEFAULT_CHARSET);

 XtVaSetValues(attriblbl,XmNlabelString,xms,NULL);
 if (xms) XmStringFree(xms);
}


// Do the convertion.
void ShapeOkCallBack(Widget w, Widget *wa, XmFileSelectionBoxCallbackStruct *cbs)
{
 int                n=0;
 CursorClass        cursor;
 FIELD      selected[NumFields];
 RECORD*            Shapes;
 MULTIPOINT*        Multipt;
 ifstream   fin, fdbf;
 ofstream   fout;
 Boolean    isFile;
 Boolean    doSeries;
 char       outfile[200];

 Shapes  = NULL;
 Multipt = NULL;

 doSeries = XmToggleButtonGetState(wa[WI_CREATESERIESTB_SHAPE]);
 SetdoSeriesValue(doSeries);
//*** Some of the variable/constants are defined in Xshp2dx.h ***

 ErrorBox eb(w);
 cursor.SetShellWidget(wa[WI_FILECONVPU_SHAPE]);
 cursor.Wait();
 XFlush(XtDisplay(w));

//*** Make sure we have input and output selected ***
 if(!fileOpen())
 {
  eb.Error("Error: NO INPUT SELECTED!");
  cursor.Normal();
  return;
 }

 isFile = XmToggleButtonGetState(wa[WI_FILETB_SHAPE]);
 if(!isFile)
  fout.attach(1);     // attach fout to fd 1, which is stdout.
 else if (!validFile(outfile,wa[WI_FILESELBOXCONV_SHAPE],w))
 {
    eb.Error("NO OUTPUT FILE SELECTED!");
    cursor.Normal();
    return;
 }
 else if (FileExists(outfile) && (!overwrite(w)))
 {
    cursor.Normal();
    return;
 }
 else
    fout.open(outfile);

//*******************************************************
 for(n = 0; n < NumFields; n++)
 {
  selected[n].name = new char[SizeOfName];
  selected[n].name[0] = '\0';
 }

 // Get the selected attribute names from the list box
 // or from series dialog list, whichever is appropriate.
 if (GetSelectedNames(&eb, selected,wa) == 0)
 {
  cursor.Normal();
  return;
 }

 // Add ios::binary in these open statements if using on a pc.
 fin.open(shpfile);
 fdbf.open(dbffile);

 SetFilterValue(!isFile);
 SetBinaryValue(XmToggleButtonGetState(wa[WI_BINARYTB_SHAPE]));
```

```
 float timetags[50];
 int   count;
 seriesdlg->getTimeTags(timetags,count);
 setTimeTags(timetags,count);

 // call the shape converter.
 DoRecords(fin,fdbf,fout,NumOfRecords,numfields,DataOffset,len,Shapes,Multipt,
        fields,selected);

 XmListDeselectAllItems(wa[WI_ATTRIBLST_SHAPE]);

 for(n = 0; n < NumFields; n++)
    delete [] selected[n].name;

 if(PERSISTENT) ShpQuit(wa[0], NULL, NULL);
 cursor.Normal();
}

int GetSelectedNames(ErrorBox *eb, FIELD selected[], Widget wa[])
{
 XmStringTable Xnames;
 int         nitems = 0;
 int         i;
 Boolean      Series;

 Series = XmToggleButtonGetState(wa[WI_CREATESERIESTB_SHAPE]);
 if (!Series)
 {
    XtVaGetValues(wa[WI_ATTRIBLST_SHAPE],
              XmNselectedItems, &Xnames,
              XmNselectedItemCount, &nitems, NULL);

    if (nitems == 0)
    {
     eb->Error("Please selected at least one attribute from the list.");
     return (0); /* error */
    }
 }
 else
 {
  seriesdlg->getSeriesList(Xnames, nitems);
  if (nitems == 0)
  {
    eb->Error("There are no attributes in the time series list, there needs to be at least one to generate
a series");
      return (0); /* error */
   }
  }
  for(i=0;i<nitems;i++)
   XmStringGetLtoR(Xnames[i], XmSTRING_DEFAULT_CHARSET, &selected[i].name);

 return (1);
}

void ShapeApplyCallBack(Widget w, Widget* wa, XmFileSelectionBoxCallbackStruct *cbs)
{
 XmListDeleteAllItems(wa[WI_ATTRIBLST_SHAPE]);
}



void PopdownDialogShape(Widget w, Widget dialog, caddr_t call_data)
{
 if (dialog)
 {
    if(SPECIFIC)
     ShpQuit(w, NULL, NULL);
    else
    {
     XtUnmanageChild(dialog);
     shp_gCursor.SetAppShellWidget(w);
     shp_gCursor.Normal();
    }
  }
  else
   XtUnmanageChild(XtParent(w));
}



void ShpQuit(Widget w, caddr_t client_data, caddr_t call_data)
{
 XCloseDisplay(XtDisplay(w));
 exit(0);
}


// Toggle the file button between file and stdout.
// Hide or show the file save box as is appropriate.
```

```
void ToggleMeFileShape(Widget widget, Widget *wa, XmAnyCallbackStruct *call_data)
{
 Boolean isFile;

 isFile = XmToggleButtonGetState(widget);
 if(!isFile)
    XtUnmanageChild(wa[WI_RFORM_SHAPE]);
 else
    XtManageChild(wa[WI_RFORM_SHAPE]);
}


// Create and display the help box for the shape converter.
void HelpShape(Widget w, Widget *WA, XmAnyCallbackStruct *call_data)
{

 char buf[5000];
 HelpBox h(w, 500);

 buf[0] = '\0';
 strcat(buf,"Left File Selection Box: used to choose input file.\n");
 strcat(buf,"    Ok      -- Start the conversion and writes out result.\n");
 strcat(buf,"    Filter  -- Used to filter files in directory.\n");
 strcat(buf,"    Quit    -- Quit the current converter session.\n");
 strcat(buf,"    Help    -- Popup this help message.\n\n");

 strcat(buf,"Options:  Input file information and conversion options.\n");
 strcat(buf,"    Attributes              -- List of selectable attributes (choose one or more.)\n");
 strcat(buf,"    Create Time Series    -- If this toggle button is checked it allows the Attribute Dialog
button to be pressed.\n");
 strcat(buf,"    Attribute Dialog ...    -- Pressing this button will popup a dialog that allows selection
of attributes for output to the Time Series.\n\n");
 strcat(buf,"Dx Structure: (appears Grayed Out unless shape type is Arcs\n");
 strcat(buf,"    Lines         -- The output dx structure will use lines to represent Arcs.\n");
 strcat(buf,"    Polylines     -- The output dx structure will use polylines to represent Arcs.\n\n");

 strcat(buf,"Output Options:\n");
 strcat(buf,"    Binary Output Toggle -- Allows choice between Binary and ASCII output.\n");
  strcat(buf,"    File Output Toggle      -- Allows choice between file or standard output for the
conversion.\n\n");

 strcat(buf,"Right file Selection Box: Used to choose output file.\n");
 strcat(buf,"    Only visible if file output is selected\n");
 strcat(buf,"    Filter -- used to filter files in directory.\n");
 h.Help(buf);
}

// ************************************************
// **** Time Series Order Dialog box and related callbacks ****

void SeriesToggle_Shape(Widget w, Widget *wa, caddr_t call_data)
{
 if (XmToggleButtonGetState(wa[WI_CREATESERIESTB_SHAPE]))
 {
  XtSetSensitive(wa[WI_ATTRIBLST_SHAPE],False);
  XtSetSensitive(wa[WI_ORDERDLGPB_SHAPE],True);
 }
 else
 {
  XtSetSensitive(wa[WI_ATTRIBLST_SHAPE],True);
  XtSetSensitive(wa[WI_ORDERDLGPB_SHAPE],False);
 }
}
```

```
//////////////////////////////////////////////////////////////////////
//  File:           seriesCall.h
//  Author:         Kirk A. Moeller
//  Description:
//                  Header file for time series dialog callbacks.
//////////////////////////////////////////////////////////////////////

#ifndef SERIESCALL_H
#define SERIESCALL_H

#include "XHeaders.h"
#include "errorbox.h"

enum {END, BEGIN, AFTER};    /* i.e. where to insert in series list */

class SeriesDialog
{
 public:
   SeriesDialog(Widget w[]);

   void PopupSeriesDialog(Widget w);

   void setAttribList(XmStringTable list, int count);
   void getSeriesList(XmStringTable& list, int& count);
   void getTimeTags(float *times, int& count);

   static void BeginTBPressed(Widget w, XtPointer ClientData, caddr_t call_data);
   static void EndTBPressed(Widget w, XtPointer ClientData, caddr_t call_data);
   static void AfterTBPressed(Widget w, XtPointer ClientData, caddr_t call_data);

   static void PopdownSeriesDialog(Widget w, XtPointer ClientData, caddr_t call_data);


   int FindInsertPos();
   void InsertTimeTag();

   static void RightArrowPressed(Widget w, XtPointer ClientData, caddr_t call_data);
   static void LeftSeriesSelection(Widget w, XtPointer ClientData, XtPointer call_data);

   static void LeftArrowPressed(Widget w, XtPointer ClientData, caddr_t call_data);
   int IsTimeValid(float time);
   static void ChangeTimeTag(Widget w, XtPointer ClientData, caddr_t call_data);
   static void RightSeriesSelection(Widget w, XtPointer ClientData, XtPointer call_data);

   int  CheckTimeTags(ErrorBox *eb);

   static void OkSeries(Widget w, XtPointer ClientData, caddr_t call_data);
   static void CancelSeries(Widget w, XtPointer ClientData, caddr_t call_data);
   static void ResetTimeTags(Widget w, XtPointer ClientData, caddr_t call_data);
   static void HelpSeries(Widget w, XtPointer ClientData, caddr_t call_data);

 private:
   Widget *wa;
   Widget attriblst, serieslst;
   Widget timetagtxt;
   Widget BeginRB, EndRB, AfterRB;
   int    old_nSeries;
};


#endif
```

```
// ************************************************************
// File:          SeriesCall.C
// Author:      Kirk A. Moeller
// Description:
//         Implementation file for the class which handles all of the
//         functionality of the Series dialog which is used both by the
//        Arc/Info and ArcShape converters.
// ************************************************************
#include "seriesCall.h"
#include "seriesW.h"
#include "errorbox.h"
#include "callutils.h"

/* needed for use with time series dialog */
 XmString  leftxms;
 int       leftitempos;
 XmString  rightxms;
 int       rightitempos;
 int       insertMethod;

/* Time series Attributes list */
/*******************************/
 XmString Attribs[50];
 int         nAttribs;
 XmString SeriesAttrib[50];
 int         nSeries;
 float       timetags[50];
 float       oldTimeTags[50];
/*******************************/


SeriesDialog::SeriesDialog(Widget w[])
{

  XtAddCallback(w[WI_OKPB_SERIES], XmNactivateCallback,
        (XtCallbackProc) SeriesDialog::OkSeries, (XtPointer) this);
  XtAddCallback(w[WI_TIMEDEFPB], XmNactivateCallback,
        (XtCallbackProc) SeriesDialog::ResetTimeTags, (XtPointer) this);
  XtAddCallback(w[WI_HELPPB_SERIES], XmNactivateCallback,
        (XtCallbackProc) SeriesDialog::HelpSeries, (XtPointer) this);
  XtAddCallback(w[WI_BEGINTB], XmNarmCallback,
        (XtCallbackProc) SeriesDialog::BeginTBPressed, (XtPointer) this);
  XtAddCallback(w[WI_ENDTB], XmNarmCallback,
        (XtCallbackProc) SeriesDialog::EndTBPressed, (XtPointer) this);
  XtAddCallback(w[WI_AFTERTB], XmNarmCallback,
        (XtCallbackProc) SeriesDialog::AfterTBPressed, (XtPointer) this);

  XtAddCallback(w[WI_CANCELPB_SERIES], XmNactivateCallback,
        (XtCallbackProc) SeriesDialog::CancelSeries, (XtPointer) this);

  XtAddCallback(w[WI_ATTRIBLST], XmNdefaultActionCallback,
        (XtCallbackProc) SeriesDialog::LeftSeriesSelection, (XtPointer) this);
  XtAddCallback(w[WI_ATTRIBLST], XmNbrowseSelectionCallback,
        (XtCallbackProc) SeriesDialog::LeftSeriesSelection, (XtPointer) this);

  XtAddCallback(w[WI_SERIESORDERLST], XmNdefaultActionCallback,
        (XtCallbackProc) SeriesDialog::RightSeriesSelection, (XtPointer) this);
  XtAddCallback(w[WI_SERIESORDERLST], XmNbrowseSelectionCallback,
        (XtCallbackProc) SeriesDialog::RightSeriesSelection, (XtPointer) this);

  XtAddCallback(w[WI_LEFTARROW], XmNactivateCallback,
        (XtCallbackProc) SeriesDialog::LeftArrowPressed, (XtPointer) this);
  XtAddCallback(w[WI_RIGHTARROW], XmNactivateCallback,
        (XtCallbackProc) SeriesDialog::RightArrowPressed, (XtPointer) this);

  XtAddCallback(w[WI_TIMETAGTXT], XmNactivateCallback,
        (XtCallbackProc) SeriesDialog::ChangeTimeTag, (XtPointer) this);
  XtAddCallback(w[WI_TIMETAGPB], XmNactivateCallback,
        (XtCallbackProc) SeriesDialog::ChangeTimeTag, (XtPointer) this);

  attriblst  = w[WI_ATTRIBLST];
  serieslst  = w[WI_SERIESORDERLST];
  timetagtxt = w[WI_TIMETAGTXT];
  BeginRB    = w[WI_BEGINTB];
  EndRB      = w[WI_ENDTB];
  AfterRB    = w[WI_AFTERTB];
  wa = w;
}

void SeriesDialog::PopupSeriesDialog(Widget w)
{
  int i, len;
  char *tmp;
  ErrorBox eb(w);
  XmStringTable Xnames;
  int count;

  leftxms = (XmString) NULL;
  rightxms = (XmString) NULL;
  leftitempos = -1;
```

```
 rightitempos = -1;
 insertMethod = END;

 XmTextSetString(timetagtxt,"");
// XmToggleButtonSetState(BeginRB,False,False);
 XmToggleButtonSetState(EndRB,True,False);
// XmToggleButtonSetState(AfterRB,False,False);


/** Save the current series list box state **/
 XtVaGetValues(serieslst,
          XmNitems, &Xnames,
          XmNitemCount,&count,NULL);

 nSeries = count;
 old_nSeries = count;
 for(i=0;i<nSeries;i++)
 {
  SeriesAttrib[i] = (XmString) NULL;
  SeriesAttrib[i] = XmStringCopy(Xnames[i]);
  oldTimeTags[i] = timetags[i];
 }
 oldTimeTags[i] = -1.0;

/** Save the current attribute list box state **/
 XtVaGetValues(attriblst,
          XmNitems, &Xnames,
          XmNitemCount,&count,NULL);

 nAttribs = count;
 for(i=0;i<nAttribs;i++)
 {
  Attribs[i] = (XmString) NULL;
  Attribs[i] = XmStringCopy(Xnames[i]);
 }

/** manage the dialog **/
 XtManageChild(wa[WI_SERIESDLG]);

}

void SeriesDialog::setAttribList(XmStringTable list, int count)
{
 int i;

 XmListDeleteAllItems(attriblst);
 XmListDeleteAllItems(serieslst);

 /** a new attribute list means everything needs to be reset. **/

 XmListAddItemsUnselected(attriblst,list,count,1);

 for(i=0;i<count;i++)
 {
  oldTimeTags[i] = -1.0;
  timetags[i] = -1.0;
 }
 oldTimeTags[i] = -1.0;
 timetags[i] = -1.0;
 nSeries = 0;
}

void SeriesDialog::getSeriesList(XmStringTable& list, int& count)
{
 XtVaGetValues(serieslst,XmNitems,&list,NULL);
 count = nSeries;
}

void SeriesDialog::getTimeTags(float *times, int& count)
{
 count = nSeries;
 for(int i=0;i<nSeries;i++)
  times[i] = timetags[i];
}

void SeriesDialog::BeginTBPressed(Widget w, XtPointer ClientData, caddr_t call_data)
{
 insertMethod = BEGIN;
}

void SeriesDialog::EndTBPressed(Widget w, XtPointer ClientData, caddr_t call_data)
{
 insertMethod = END;
}

void SeriesDialog::AfterTBPressed(Widget w, XtPointer ClientData, caddr_t call_data)
{
 insertMethod = AFTER;
}
```

```
void SeriesDialog::PopdownSeriesDialog(Widget w, XtPointer ClientData, caddr_t call_data)
{
 SeriesDialog *dlg = (SeriesDialog*) ClientData;

 XtUnmanageChild(dlg->wa[WI_SERIESDLG]);
}

int SeriesDialog::FindInsertPos()
{
 int nitems = 0;

 if (insertMethod == BEGIN)
  return (0);
 else if (insertMethod == AFTER)
  return (rightitempos);
 else
 {
  XtVaGetValues(serieslst,XmNitemCount, &nitems,NULL);
  return (nitems);
 }
}

void SeriesDialog::InsertTimeTag()
{
 int     insertpos, i;
 float   fli;

 insertpos = FindInsertPos();
 XtVaGetValues(serieslst,XmNitemCount,&i,NULL);
 while(i > insertpos)
 {
  timetags[i] = timetags[i-1];
  i--;
 }

 fli = (float) (i+1);
 if (insertMethod == END)
  timetags[i] = (timetags[i-1] < fli) ? fli : -999.0;
 else
  timetags[i] = -999.0;
}

void SeriesDialog::RightArrowPressed(Widget w, XtPointer ClientData, caddr_t call_data)
{
 ErrorBox eb(w);
 SeriesDialog *dlg = (SeriesDialog*) ClientData;

 if (leftitempos > 0)
 {
  /* must be done prior to adding to the list box. */
  dlg->InsertTimeTag();
  /* else -- begin of list is position 1 which is the value of BEGIN  */
  /*      -- end of list is position 0 which is the value of END      */
  /* and of course insertMethod is equal to either END,BEGIN,or AFTER */

  if ((insertMethod == AFTER) && (rightitempos > 0))
   XmListAddItemUnselected(dlg->serieslst,leftxms,(rightitempos+1));
  else if ((insertMethod == AFTER) && (rightitempos == -1))
  {
   eb.Error("Please select an item to insert After.");
   return;
  }
  else
   XmListAddItemUnselected(dlg->serieslst,leftxms,insertMethod);

  if ((insertMethod == BEGIN) && (rightitempos > 0))
   rightitempos++;


  XmListDeselectAllItems(dlg->serieslst);
  XmListDeletePos(dlg->attriblst,leftitempos);
  XmStringFree(leftxms);
  leftxms = (XmString) NULL;
  leftitempos = -1;
  nSeries++;
 }
 else
  eb.Error("Please select an item to move from the left list");
}

void SeriesDialog::LeftSeriesSelection(Widget w, XtPointer ClientData, XtPointer call_data)
{
 XmListCallbackStruct *cbs = (XmListCallbackStruct*) call_data;

 leftxms = (XmString) NULL;
 leftitempos = -1;
 leftxms = XmStringCopy(cbs->item);
 leftitempos = cbs->item_position;
}
```

```
void SeriesDialog::LeftArrowPressed(Widget w, XtPointer ClientData, caddr_t call_data)
{
 ErrorBox eb(w);
 int     i;

 SeriesDialog *dlg = (SeriesDialog*) ClientData;

 if (rightitempos > 0)
 {
  XmListDeletePos(dlg->serieslst,rightitempos);
  XmListAddItemUnselected(dlg->attriblst,rightxms,END);

  i = rightitempos-1;
  while(timetags[i] != -1.0)
  {
   timetags[i] = timetags[i+1];
   i++;
  }

  XmStringFree(rightxms);
  rightxms = (XmString) NULL;
  rightitempos = -1;
  nSeries--;
 }
 else
  eb.Error("Please select an item to move from the right list");
}

int SeriesDialog::IsTimeValid(float time)
{
 float prev, next;

 /** Times should look like this: **/
 /**     prev < time < next      **/

 /** make sure we don't access beyond array bounds **/
 prev = (rightitempos == 1) ? -1.0 : timetags[rightitempos-2];
 next = (timetags[rightitempos] == -1.0) ? 999999.0 : timetags[rightitempos];
 next = (next == -999.0) ? 999999.0 : next;

 if ( (time <= prev) || (time >= next))
  return 0;
 return 1;
}
```

```
void SeriesDialog::ChangeTimeTag(Widget w, XtPointer ClientData, caddr_t call_data)
{
 char *tmp, *err;
 float time;
 ErrorBox eb(w,100);

 SeriesDialog *dlg = (SeriesDialog*) ClientData;

 tmp = XmTextGetString(dlg->timetagtxt);

 err = tmp;
 time = (float)strtod(tmp,&err);
 if (*err != '\0')
 {
  eb.Error("Invalid time tag value!");
  XmTextSetString(dlg->timetagtxt,"");
  return;
 }

 if (!dlg->IsTimeValid(time))
 {
  eb.Error("Time value is out of sequence with the other values in the list\n\
Enter a different value or reorder the list.");
  XmTextSetString(dlg->timetagtxt,"");
  return;
 }

 timetags[rightitempos-1] = time;

 sprintf(tmp,"%3.3f",time);
 XmTextSetString(dlg->timetagtxt,tmp);
}

void SeriesDialog::RightSeriesSelection(Widget w, XtPointer ClientData, XtPointer call_data)
{
 XmListCallbackStruct *cbs = (XmListCallbackStruct*) call_data;
 char tmp[20];

 SeriesDialog *dlg = (SeriesDialog*) ClientData;

 rightitempos = -1;
 rightxms = (XmString) NULL;
 rightxms = XmStringCopy(cbs->item);
 rightitempos = cbs->item_position;
```

```
 /** fill in time tag text field **/
 sprintf(tmp,"%3.3f",timetags[rightitempos-1]);
 XmTextSetString(dlg->timetagtxt, tmp);
}

int SeriesDialog::CheckTimeTags(ErrorBox *eb)
{
 int i = 0;
 char tmp[300];

 while(timetags[i] != -1.0)
 {
  if (timetags[i] < 0)
  {
   sprintf(tmp,"The time tag of attribute #%d (%3.3f) is invalid!\n",i+1,timetags[i]);
   strcat(tmp,"Please change the value to a positive number.");

   eb->Error(tmp);
   return 0;
  }
  i++;
 }
 return 1;
}

void SeriesDialog::OkSeries(Widget w, XtPointer ClientData, caddr_t call_data)
{
 int i;
 ErrorBox eb(w,50);
 SeriesDialog *dlg = (SeriesDialog*) ClientData;

 if (dlg->CheckTimeTags(&eb) == 0)
  return;

 PopdownSeriesDialog(w, ClientData, NULL);
}

void SeriesDialog::CancelSeries(Widget w, XtPointer ClientData, caddr_t call_data)
{
 int i;

 SeriesDialog *dlg = (SeriesDialog*) ClientData;

 XmListDeleteAllItems(dlg->attriblst);
 for(i=0;i<nAttribs;i++)
 {
  XmListAddItemUnselected(dlg->attriblst,Attribs[i],0);
  XmStringFree(Attribs[i]);
 }

 XmListDeleteAllItems(dlg->serieslst);
 nSeries = dlg->old_nSeries;
 for(i=0;i<nSeries;i++)
 {
  XmListAddItemUnselected(dlg->serieslst,SeriesAttrib[i],0);
  XmStringFree(SeriesAttrib[i]);
  timetags[i] = oldTimeTags[i];
 }
 timetags[i] = -1.0;

 PopdownSeriesDialog(w, ClientData, NULL);
}

void SeriesDialog::ResetTimeTags(Widget w, XtPointer ClientData, caddr_t call_data)
{
 int i=0;
 SeriesDialog *dlg = (SeriesDialog*) ClientData;

 while ((timetags[i] != -1.0) && (timetags[i] < 50))
 {
  timetags[i] = (float) (i+1);
  i++;
 }
 XmListDeselectAllItems(dlg->serieslst);
 XmTextSetString(dlg->timetagtxt,"");
}

void SeriesDialog::HelpSeries(Widget w, XtPointer ClientData, caddr_t call_data)
{
 ErrorBox eb(w);

 eb.Error("Sorry, no help yet!");
}
```

```
// ERDAS/IMAGINE Converter
////////////////////////////////////////////////////////////////////////
//   File:          callbacksIMG.h
//   Author:        Kirk A. Moeller
//   Description:
//                  Header file for the erdas imagine callback
//                  functions.
////////////////////////////////////////////////////////////////////////
#ifndef CALLBACKSIMG_H
#define CALLBACKSIMG_H

#include "XHeaders.h"

/* PROTOTYPES */
void AddIMGCallbacks(Widget w[], Widget mainw[], Widget imgpb);

void SetSelectorValuesIMG(int,int,int,int);
void SetDefaultValuesIMG(Widget w, Widget *wa, caddr_t call_data);
void clearMinMax_IMG(Widget *wa);
void sensitizeMinMax(Widget *wa);
void deSensitizeMinMax(Widget* wa);

void PopupDialogIMG(Widget w, Widget *wa, caddr_t call_data);

void OpenFileIMG(Widget w, Widget *wa, XmListCallbackStruct *list);
void OpenLayerIMG(Widget w, Widget *wa, XmListCallbackStruct *list);

void IMGOkCallBack(Widget w, Widget *wa, XmFileSelectionBoxCallbackStruct *cbs);
void IMGApplyCallBack(Widget w, Widget *wa, XmFileSelectionBoxCallbackStruct *cbs);

void PopdownDialogIMG(Widget w, Widget dialog, caddr_t call_data);
void ImgQuit(Widget w, caddr_t client_data, caddr_t call_data);

void ToggleMeFileIMG (Widget widget, Widget *wa, XmAnyCallbackStruct *call_data);
void HelpIMG(Widget w, Widget *WA, XmAnyCallbackStruct *call_data);

// defined in callbacks.C
extern void PopupSelector(Widget w, Widget *wa, caddr_t call_data);

#endif
```

```
// ERDAS/IMAGINE Converter
////////////////////////////////////////////////////////////////////
//   File:          callbacksIMG.C
//   Author:        Kirk A. Moeller
//   Description:
//                  Implements all the callbacks that respond
//                  to user actions in the Erdas Imagine dialog.
////////////////////////////////////////////////////////////////////

#include "callbacksIMG.h"
#include "img2dxW.h"
#include "img.h"
#include "cursor.h"
#include "errorbox.h"
#include "globals.h"
#include "callutils.h"

/* globals */
CursorClass img_gCursor;

/** True if all layers have same width & height, allow multiple selection
 ** of layers. */
Boolean    MultiLayers;
/* img converter globals */
Convert *img;
Widget  minxW,minyW,maxxW,maxyW;

// This procedure registers the callbacks for the widgets in
// the Erdas Imagine dialog box.

void AddIMGCallbacks(Widget w[], Widget mainw[], Widget imgpb)
{
  Widget tmpw1;
  Atom   WM_DELETE_WINDOW;

    XtAddCallback(imgpb, XmNactivateCallback,
                  (XtCallbackProc) PopupDialogIMG, w);
        XtAddCallback(w[WI_OPENBOX_IMG], XmNcancelCallback,
                  (XtCallbackProc) PopdownDialogIMG, w[WI_IMGFORM]);
        tmpw1 = XmSelectionBoxGetChild(w[WI_OPENBOX_IMG],
                   XmDIALOG_FILE_LIST);
        XtAddCallback(tmpw1, XmNbrowseSelectionCallback,
                  (XtCallbackProc) OpenFileIMG, w);
        XtAddCallback(w[WI_LAYERSSL_IMG], XmNbrowseSelectionCallback,
                  (XtCallbackProc) OpenLayerIMG, w);
        tmpw1 = XmSelectionBoxGetChild(w[WI_SAVEBOX_IMG],
                   XmDIALOG_TEXT);
        XtAddCallback(tmpw1, XmNactivateCallback,
                  (XtCallbackProc) IMGOkCallBack, w);
        tmpw1 = XmSelectionBoxGetChild(w[WI_OPENBOX_IMG],
                   XmDIALOG_TEXT);
        XtAddCallback(tmpw1, XmNactivateCallback,
                  (XtCallbackProc) OpenFileIMG, w);
    XtAddCallback(w[WI_OPENBOX_IMG], XmNokCallback,
                  (XtCallbackProc) IMGOkCallBack, w);
    XtAddCallback(w[WI_OPENBOX_IMG], XmNapplyCallback,
                  (XtCallbackProc) IMGApplyCallBack, w);
        XtAddCallback(w[WI_FILETB_IMG], XmNvalueChangedCallback,
                  (XtCallbackProc) ToggleMeFileIMG, w);
        XtAddCallback(w[WI_OPENBOX_IMG], XmNhelpCallback,
                  (XtCallbackProc) HelpIMG, w);
        XtAddCallback(w[WI_CLIPPB_IMG], XmNactivateCallback,
                  (XtCallbackProc) PopupSelector, mainw);
        XtAddCallback(w[WI_DEFAULTPB_IMG], XmNactivateCallback,
                  (XtCallbackProc) SetDefaultValuesIMG, w);

 WM_DELETE_WINDOW = XmInternAtom(XtDisplay(w[WI_IMGDLGSHELL]),
                  "WM_DELETE_WINDOW", False);
 XmAddWMProtocolCallback(w[WI_IMGDLGSHELL],WM_DELETE_WINDOW,
        (XtCallbackProc)PopdownDialogIMG,w[WI_IMGFORM]);
}

// Set the values of min/max edit boxes.

void SetSelectorValuesIMG(int lx, int ly, int hx, int hy)
{
 XmTextSetString(minxW,itoa(lx));
 XmTextSetString(minyW,itoa(ly));
 XmTextSetString(maxxW,itoa(hx));
 XmTextSetString(maxyW,itoa(hy));
}

// This procedure gets called if the user presses the default
// button.  The default min/max values will be placed in the
// min/max edit boxes.

void SetDefaultValuesIMG(Widget w, Widget *wa, caddr_t call_data)
{
 XmTextSetString(wa[WI_MINXTXT_IMG],"0");
 XmTextSetString(wa[WI_MINYTXT_IMG],"0");
```

```
 XmTextSetString(wa[WI_MAXXTXT_IMG],itoa( getWidth() ));
 XmTextSetString(wa[WI_MAXYTXT_IMG],itoa( getHeight() ));
}

void clearMinMax_IMG(Widget *wa)
{
 XmTextSetString(wa[WI_MINXTXT_IMG],"");
 XmTextSetString(wa[WI_MINYTXT_IMG],"");
 XmTextSetString(wa[WI_MAXXTXT_IMG],"");
 XmTextSetString(wa[WI_MAXYTXT_IMG],"");
}

void sensitizeMinMax(Widget *wa)
{
 XtSetSensitive(wa[WI_MINXTXT_IMG],True);
 XtSetSensitive(wa[WI_MINYTXT_IMG],True);
 XtSetSensitive(wa[WI_MAXXTXT_IMG],True);
 XtSetSensitive(wa[WI_MAXYTXT_IMG],True);
 XtSetSensitive(wa[WI_CLIPPB_IMG],True);
 XtSetSensitive(wa[WI_DEFAULTPB_IMG],True);
}

void deSensitizeMinMax(Widget* wa)
{
 XtSetSensitive(wa[WI_MINXTXT_IMG],False);
 XtSetSensitive(wa[WI_MINYTXT_IMG],False);
 XtSetSensitive(wa[WI_MAXXTXT_IMG],False);
 XtSetSensitive(wa[WI_MAXYTXT_IMG],False);
 XtSetSensitive(wa[WI_CLIPPB_IMG],False);
 XtSetSensitive(wa[WI_DEFAULTPB_IMG],False);
}

void PopupDialogIMG(Widget w, Widget *wa, caddr_t call_data)
{
 int n = 0;
 Arg args[5];
 XmString xms = (XmString) NULL;

 setGisType(IMG);
 minxW   = wa[WI_MINXTXT_IMG];
 minyW   = wa[WI_MINYTXT_IMG];
 maxxW   = wa[WI_MAXXTXT_IMG];
 maxyW   = wa[WI_MAXYTXT_IMG];

 if(!SPECIFIC)
 {
   img_gCursor.SetAppShellWidget(w);
   img_gCursor.DontEnter();
 }


// Set the file filters to the appropriate values.
 if(DIR_FLAG)
 {
   xms = XmStringCreate(DEF_DIR, XmSTRING_DEFAULT_CHARSET);
   XtVaSetValues(wa[WI_OPENBOX_IMG], XmNdirectory, xms, NULL);
   XtVaSetValues(wa[WI_SAVEBOX_IMG], XmNdirectory, xms, NULL);
   if (xms) XmStringFree(xms);
 }

// If either of these are true we want to output to stdout
// and adjust the dialog box accordingly
 if (PERSISTENT || FILTER)
 {
  XtUnmanageChild(wa[WI_RFORM_IMG]);
  XmToggleButtonSetState(wa[WI_FILETB_IMG],False,False);
 }
 XtManageChild(wa[WI_IMGFORM]);

 /** Do some initialization **/
 clearMinMax_IMG(wa);
 deSensitizeMinMax(wa);
 MultiLayers = True;
 XmListDeleteAllItems(wa[WI_LAYERSSL_IMG]);

XtVaSetValues(wa[WI_LAYERSSL_IMG],XmNselectionPolicy,XmMULTIPLE_SELECT,NULL);

 fileState(False);
}

// This callback procedure will take the open the selected file
// and call function which will retrieve a list of layers contained
// in the imagine file.  This list of layers is then displayed in
// the listbox in the imagine converter dialog box.

void OpenFileIMG(Widget w, Widget *wa, XmListCallbackStruct *list)
{
 CursorClass cursor;
 char      *filename;
 XmString   xms;
```

```
 int       num_layers;
 int       i = 0;
 Arg       args[10];

 cursor.SetShellWidget(wa[WI_IMGDLGSHELL]);
 cursor.Wait();
 XFlush(XtDisplay(w));


 // RESET SOME STUFF IN CASE THIS IS NOT THE FIRST CONVERSION
 fileState(False);
 clearMinMax_IMG(wa);

 XmListDeleteAllItems(wa[WI_LAYERSSL_IMG]);
 deSensitizeMinMax(wa);

//*** GET INPUT FILE NAME ****
   XmStringGetLtoR(list->item, XmSTRING_DEFAULT_CHARSET, &filename);

 if (img)
   delete img;

 // instantiate the conveter class and get the layers.
 img = new Convert(filename);
 MultiLayers = (img->ReadLayers()) ? True : False;
 num_layers = img->getnumlayers();

 if (MultiLayers && (num_layers > 0))
 {
  sensitizeMinMax(wa);
  setWidth(img->getwidth());
  setHeight(img->getheight());
  SetDefaultValuesIMG(w,wa,NULL);

XtVaSetValues(wa[WI_LAYERSSL_IMG],XmNselectionPolicy,XmMULTIPLE_SELECT,NULL);
   fileState(True);
 }
 else

XtVaSetValues(wa[WI_LAYERSSL_IMG],XmNselectionPolicy,XmBROWSE_SELECT,NULL);

 // Place the list of layers in the list box.
 for(i = 0; i < num_layers; i++)
 {
  xms = (XmString) NULL;
  xms = XmStringCreateLocalized(img->layer_name[i]);
  XmListAddItemUnselected(wa[WI_LAYERSSL_IMG],xms,0);
  if (xms) XmStringFree(xms);
 }

 cursor.Normal();
}

// This callback procedure gets called when the user selects one
// of the layers.  The layer information in the imagine file will
// be accessed and information such as image size will be retrieved.

void OpenLayerIMG(Widget w, Widget *wa, XmListCallbackStruct *list)
{
 CursorClass cursor;
 int errval;
 ErrorBox eb(w);
 int item_pos = 0;

 img->DestroyOldData();

 cursor.SetShellWidget(wa[WI_IMGDLGSHELL]);
 cursor.Wait();
 XFlush(XtDisplay(w));

 item_pos = list->item_position - 1;

 errval = img->ValidPixels(item_pos);
 if (errval == WRONGPIXTYPE)
 {
   eb.Error("Sorry, Unsupported pixel type detected in Layer!");
   // RESET SOME STUFF
   fileState(False);
   clearMinMax_IMG(wa);

   XmListDeselectAllItems(wa[WI_LAYERSSL_IMG]);
   deSensitizeMinMax(wa);
   cursor.Normal();
   return;
 }

 // Turn on various controls, now that a layer has been selected.
 sensitizeMinMax(wa);

 setWidth(img->getwidth(item_pos));
```

```
 setHeight(img->getheight(item_pos));
 SetDefaultValuesIMG(w,wa,NULL);


 fileState(True);
 cursor.Normal();
}

// This callbacks procedure calls the functions which will write the
// raster data and color table information(if present) out to a file
// or stdout in a format understandable to Data Explorer.  The data
// was retrieved in the OpenLayer callbacks above.

void IMGOkCallBack(Widget w, Widget *wa, XmFileSelectionBoxCallbackStruct *cbs)
{
 CursorClass cursor;
 Widget     tmpw;
 char       *outfile;
 int n = 0;
 int n2 = 0;
 Boolean isFile, isBinary;
 int MinMax[4];
 int *pos_list;
 int pos_count;
 int i;

 // Initialization
 pos_list = new int[10];
 outfile = new char[200];
 outfile[0] = '\0';

 ErrorBox eb(w);
 cursor.SetShellWidget(wa[WI_IMGDLGSHELL]);
 cursor.Wait();
 XFlush(XtDisplay(w));

 // Make sure both the file and layer is open before allowing execution to
 // continue.
 tmpw = XmSelectionBoxGetChild(wa[WI_OPENBOX_IMG], XmDIALOG_LIST);
 XtVaGetValues(tmpw, XmNselectedItemCount, &n, NULL);

 XmListGetSelectedPos(wa[WI_LAYERSSL_IMG],&pos_list,&pos_count);
 pos_list[pos_count] = -1;

 if(!n || !pos_count)
{
 eb.Error("Error: NO INPUT SELECTED!");
 cursor.Normal();
 return;
}
isFile   = XmToggleButtonGetState(wa[WI_FILETB_IMG]);
isBinary = XmToggleButtonGetState(wa[WI_BINARYTB_IMG]);

if (!isFile)
 strcpy(outfile,"stdout");
else if (!validFile(outfile,wa[WI_SAVEBOX_IMG],w))
{
   eb.Error("NO OUTPUT FILE SELECTED!");
   cursor.Normal();
   return;
}
else if (FileExists(outfile) && (!overwrite(w)))
{
   cursor.Normal();
   return;
}

MinMax[0] = atoi(XmTextGetString(wa[WI_MINXTXT_IMG]));
MinMax[1] = atoi(XmTextGetString(wa[WI_MINYTXT_IMG]));
MinMax[2] = atoi(XmTextGetString(wa[WI_MAXXTXT_IMG]));
MinMax[3] = atoi(XmTextGetString(wa[WI_MAXYTXT_IMG]));


img->OpenOutput(outfile);
i = 0;
while (pos_list[i] != -1)
{
 /** layers start at 0, list box starts at 1. **/
 pos_list[i] -= 1;

 img->ReadData(pos_list[i]);
 // This is the line that does the creation of the output.
 img->WriteData((isBinary == True),MinMax);
 i++;
}
if (i == 1)
 img->WriteEnd();
else
{
 img->WriteDxGroup(pos_list);
```

```
 }

 delete [] pos_list;
 cursor.Normal();
}

void IMGApplyCallBack(Widget w, Widget* wa, XmFileSelectionBoxCallbackStruct *cbs)
{
}

// Kill the dialog
void PopdownDialogIMG(Widget w, Widget dialog, caddr_t call_data)
{
 if (dialog)
 {
    if(SPECIFIC)
      ImgQuit(w, NULL, NULL);
    else
    {
      XtUnmanageChild(dialog);
      img_gCursor.SetAppShellWidget(w);
      img_gCursor.Normal();
    }
 }
 else
      XtUnmanageChild(XtParent(w));
}

void ImgQuit(Widget w, caddr_t client_data, caddr_t call_data)
{
 XCloseDisplay(XtDisplay(w));
 exit(0);
}

// Either show or hide to file save box depending on file toggle state.
void ToggleMeFileIMG(Widget widget, Widget *wa, XmAnyCallbackStruct *call_data)
{
 Boolean isFile;

 isFile = XmToggleButtonGetState(wa[WI_FILETB_IMG]);

 if(!isFile)
    XtUnmanageChild(wa[WI_RFORM_IMG]);
 else
    XtManageChild(wa[WI_RFORM_IMG]);
}

// This procedure creates and displays the help dialog for the
// erdas imagine converter.

void HelpIMG(Widget w, Widget *WA, XmAnyCallbackStruct *call_data)
{

  char buf[3000];
  int position;
  HelpBox h(w, 400);

  buf[0] = '\0';
  strcat(buf,"Left File Selection Box: used to choose input file.\n");
  strcat(buf,"   Ok      -- Start the conversion and writes out result.\n");
  strcat(buf,"   Filter  -- Used to filter files in directory.\n");
  strcat(buf,"   Quit    -- Quit the current converter session.\n");
  strcat(buf,"   Help    -- Popup this help message.\n\n");

  strcat(buf,"Conversion Options:\n");
   strcat(buf,"    min/max edit boxes    -- Display the size in pixels of the image. (These can be
modified.)\n");
  strcat(buf,"   Default Button         -- Resets the min/max values to there original values.\n");
   strcat(buf,"   Clip Button            -- Displays a dialog box that allows adjustment of the image
size.\n");
   strcat(buf,"   Layers List Box        -- Displays a list of layers present in the currently opened
file.\n\n");

  strcat(buf,"Output Options:\n");
  strcat(buf,"   Binary Output Toggle -- Allows choice between Binary and ASCII output.\n");
   strcat(buf,"   File Output Toggle     -- Allows choice between file or standard output for the
conversion.\n\n");

  strcat(buf,"Right file Selection Box: Used to choose output file.\n");
  strcat(buf,"   Only visible if file output is selected\n");
  strcat(buf,"   Filter -- used to filter files in directory.\n\n");

  strcat(buf,"Notes:\n");
  strcat(buf,"   1.  If the layers in the imagine file are NOT all the same width and height than:\n");
   strcat(buf,"         -- The min/max boxes, default buttuon, and clip button will not operate until a
layer is selected.\n");
  strcat(buf,"         -- Only one layer can be converted at a time.\n");
  strcat(buf,"   2.  It is recommended that Binary Output be used, as the ASCII file tends to be much
larger.\n");
```

```
    h.Help(buf);
}
```

```
/*************************************************************************
** File:       FileOutCk.h
** Author:     Kirk A. Moeller
** Description:
**      This function checks if the user gave an output file and if so
**      adds a .dx extension if there not already present.
**      If no file was selected the function returns 0, otherwise it
**      will return 1;
*************************************************************************/

#ifndef CALLUTILS_H
#define CALLUTILS_H

#include "XHeaders.h"
enum {Dem, LAN, Dlg, MOD, Arc, IMG, none};

Boolean fileOpen();
void fileState(Boolean newstate);

Boolean gisType();
void setGisType(int newtype);

int getType();
int ChangeType(int newtype = none);

int getWidth();
void setWidth(int newWidth);

int getHeight();
void setHeight(int newHeight);

int validFile(char* outfile,Widget filebox, Widget w);
int FileExists(char *file);

int  overwrite(Widget parent);
void overwriteOk(Widget w, int* answer, caddr_t call_data);
void overwriteCancel(Widget w, int* answer, caddr_t call_data);

char* itoa(int i);

#endif
```

```
/*****************************************************************************
** File:       callutils.C
** Author:     Kirk A. Moeller
** Description:
**        This file contain a couple of utility functions used by
**        various callbacks in the converters.
*****************************************************************************/

#include "callutils.h"
#include <string.h>
#include <fstream.h>

/** The variables that these functions modify and return
 ** were originally global.  These functions seemed the best way
 ** of limiting the scope of the variables.
**/

/*********************************************/
static Boolean isFileOpen = False;

Boolean fileOpen()
{
 return isFileOpen;
}

void fileState(Boolean newstate)
{
 isFileOpen = newstate;
}

/*********************************************/
static Boolean GISTYPE = none;

Boolean gisType()
{
 return (GISTYPE);
}

void setGisType(int newtype)
{
 GISTYPE = newtype;
}

/*********************************************/
static int WIDTH  = 0;
static int HEIGHT = 0;

int getWidth()
{
 return WIDTH;
}

void setWidth(int newWidth)
{
 WIDTH = (newWidth < 0) ? 0 : newWidth;
}

int getHeight()
{
 return HEIGHT;
}

void setHeight(int newHeight)
{
 HEIGHT = (newHeight < 0) ? 0 : newHeight;
}

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


/** This function checks if the user gave an output file and if so
**  adds a .dx extension if there not already present.
**  If no file was selected the function returns 0, otherwise it
**  will return 1;
**/
int validFile(char* outfile,Widget filebox, Widget w)
{
 int  curpos;
 char *p, *file;
 Widget select;

 select = XmSelectionBoxGetChild(filebox,XmDIALOG_TEXT);
 file = XmTextGetString(select);

 p = strrchr(file,'/');
 if (p == NULL) return 0;

 p++;
 if ( (*p == '\0') || (*p == ' ') )
  return 0;
```

```
 else if ( !(strstr(p,".dx")) )
 {
    curpos = XmTextGetLastPosition(select);
    XmTextInsert(select,curpos,".dx");
    XFlush(XtDisplay(w));
    file = XmTextGetString(select);
 }
 strcpy(outfile,file);
 return 1;
}

int FileExists(char *file)
{
 int exists;

 ifstream fin(file);

 exists = (fin) ? 1 : 0;

 fin.close();
 return exists;
}


/*************************************************
** OVERWRITE FUNCTIONS **
*************************************************/
extern XtAppContext appContext;

/** Used by callbacks to popup a overwrite warning box
**  and then wait for the user response.
**/
int overwrite(Widget parent)
{
 static Widget dialog;
 static int answer;
 XmString xms  = (XmString) NULL,
        xms1 = (XmString) NULL,
        xms2 = (XmString) NULL;
 Widget tmpw;
 int n;
 Arg args[10];

 if (!dialog)
 {
  n = 0;
  XtSetArg(args[n], XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL); n++;

  xms  = XmStringCreate("Warning!",XmSTRING_DEFAULT_CHARSET);
  xms1  = XmStringCreate("Output File Exists!",XmSTRING_DEFAULT_CHARSET);
  xms2  = XmStringCreate("Overwrite?",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(args[n], XmNdialogTitle, xms); n++;
  XtSetArg(args[n], XmNmessageString, xms1); n++;
  XtSetArg(args[n], XmNokLabelString, xms2); n++;

  dialog = XmCreateWarningDialog(parent, "overwrite_dxf", args, n);
  tmpw = XmMessageBoxGetChild(dialog,XmDIALOG_HELP_BUTTON);
  XtUnmanageChild(tmpw);

  XtAddCallback(dialog,XmNokCallback,(XtCallbackProc) overwriteOk,&answer);
  XtAddCallback(dialog,XmNcancelCallback,(XtCallbackProc) overwriteCancel,&answer);

  if (xms) XmStringFree(xms);
  if (xms1) XmStringFree(xms1);
  if (xms2) XmStringFree(xms2);
 }
 answer = -1;

 XtManageChild(dialog);

 while (answer == -1)
  XtAppProcessEvent(appContext, XtIMAll);

 XtUnmanageChild(dialog);
 XSync (XtDisplay(dialog),0);
 XmUpdateDisplay (parent);

 return answer;
}
void overwriteOk(Widget w, int* answer, caddr_t call_data)
{
 *answer = 1;
}

void overwriteCancel(Widget w, int* answer, caddr_t call_data)
{
 *answer = 0;
}
```

```
/*********************************************************/

// converter an integer to an ascii string.
char* itoa(int i)
{
  static char buf[256];
  sprintf(buf, "%d", i);
  return buf;
}
```

```
//**********************************************************************
// FILE:          utils.h
// AUTHOR:        (Unknown)
// DESCRIPTION:
//   Prototypes for a couple of utility functions for getting
//   shell widgets.
//**********************************************************************

#ifndef UTILS_H
#define UTILS_H

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

Boolean getApplicationShell(Widget& w, Widget& p);

Boolean getShell(Widget& w, Widget& p);

#endif
```

```
//************************************************************
// FILE:        utils.C
// AUTHOR:      (Unknown)
// DESRIPTION:
//   A couple of utility function for getting shell widgets
//************************************************************

#include "utils.h"

Boolean getApplicationShell(Widget& w, Widget &p)
{
        p = w;
        while(!XtIsApplicationShell(p) && p != NULL)
        {
                p = XtParent(p);
        }

        if (p)
                return True;
        return False;
}

Boolean getShell(Widget& w, Widget &p)
{
        p = w;

        while(!XtIsShell(p) && p != NULL)
        {
                p = XtParent(p);
        }

        if(p)
                return True;
        return False;
}
```

```
//////////////////////////////////////////////////////////////////////
//   File:          helpbox.h
//   Author:        David Thompson(??)
//   Description:
//     This is the header file for the class which implements a
//     help dialog box.
//////////////////////////////////////////////////////////////////////

#ifndef HELPBOX_H
#define HELPBOX_H

#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include "cursor.h"

class HelpBox
{
 private:
            Widget w;
            int charwidth;
            friend void DisplayMoreHelp(Widget w, char* msg,
                                        XmAnyCallbackStruct *call_data);
            friend void RemoveHelp(Widget w, caddr_t client_data,
                                        XmAnyCallbackStruct *call_data);
 public:
            HelpBox() { w = NULL; }
            HelpBox(Widget& parent, int width = 30);
            ~HelpBox();
            void SetMoreHelp(char *);
            void Help(char *);
};

#endif
```

```
//////////////////////////////////////////////////////////////////
//  File:       helpbox.C
//  Author:     David Thompson(??)
//  Description:
//    A class which implements a help dialog box.
//////////////////////////////////////////////////////////////////

#include <string.h>
#include <iostream.h>
#include <Xm/MessageB.h>
#include "utils.h"
#include "helpbox.h"
#include "g_helpbox.h"


HelpBox::HelpBox(Widget& parent, int width)
{
        Widget appshell;
        charwidth = width;
        int n = 0;
        Arg args[3];
        XmString xms = (XmString) NULL;

        if(!getShell(parent, appshell))
        {
                cout << "Couldn't get application shell for error box." << endl;
                appshell = parent;
        }

//      hlp_cursor.SetWidget(appshell);

//      XtSetArg(args[n], XmNdialogStyle,
XmDIALOG_PRIMARY_APPLICATION_MODAL); n++;
        XtSetArg(args[n], XmNdialogStyle, XmDIALOG_MODELESS); n++;
        xms = XmStringCreate("Help", XmSTRING_DEFAULT_CHARSET);
        XtSetArg(args[n], XmNdialogTitle, xms); n++;
        w = XmCreateMessageDialog(parent, "HelpBox", args, n);
        if(xms) XmStringFree(xms);

        XtAddCallback(w, XmNokCallback, (XtCallbackProc) RemoveHelp, NULL);

        Widget tmpw;
        tmpw = XmMessageBoxGetChild(w, XmDIALOG_CANCEL_BUTTON);
        XtUnmanageChild(tmpw);
        tmpw = XmMessageBoxGetChild(w, XmDIALOG_HELP_BUTTON);
        XtUnmanageChild(tmpw);

}

void RemoveHelp(Widget w, caddr_t client_data, XmAnyCallbackStruct *call_data)
{
        if(w)
        {
                XtDestroyWidget(w);
        }
//      hlp_cursor.Normal();
}


HelpBox::~HelpBox()
{
// Nothing to destroy.  Taken care of in RemoveHelp callback.
}

void HelpBox::SetMoreHelp(char *hlpmsg)
{
        Widget tmpw;
        char *tmp;

        tmp = new char[strlen(hlpmsg) + 1];
        strcpy(tmp, hlpmsg);

        tmpw = XmMessageBoxGetChild(w, XmDIALOG_HELP_BUTTON);
        XtManageChild(tmpw);

        XtAddCallback(w, XmNhelpCallback, (XtCallbackProc) DisplayMoreHelp, tmp);
}

void DisplayMoreHelp(Widget w, char* msg, XmAnyCallbackStruct *call_data)
{
        HelpBox h(w);
        h.Help(msg);
        delete msg;
}

void HelpBox::Help(char *err)
{
//      hlp_cursor.DontEnter();

        XmString xms = (XmString) NULL;
```

```
        enum Space { now, notnow };
        Space space = notnow;
        char *temp, *temp1, *temp2 = err;
        int count = 0;
        int len = strlen(err);

        temp = new char[len+1];
        temp1 = temp;

        while(*temp2 != '\0')
        {
                if(space == now || count > charwidth)
                        space = now;

                if(*temp2 == '\n' || (space == now &&
                                (*temp2 == ' ' || *temp2 == '\t')))
                {
                        *temp1 = '\0';
                        xms = XmStringConcat(xms, XmStringCreate(temp,

XmSTRING_DEFAULT_CHARSET));
                        xms = XmStringConcat(xms, XmStringSeparatorCreate());
                        temp1 = temp;
                        count = 0;
                        space = notnow;
                }
                else
                {
                        *temp1 = *temp2;
                        count++; temp1++;
                }

                temp2++;
        }
        *temp1 = '\0';

        xms = XmStringConcat(xms, XmStringCreate(temp,
                                XmSTRING_DEFAULT_CHARSET));

        XtVaSetValues(w, XmNmessageString, xms, NULL);

        if(xms) XmStringFree(xms);

        XtManageChild(w);
}
```

```
//////////////////////////////////////////////////////////////////////
//  File:         errobox.h
//  Author:
//  Description:
//                A class implementing an X-Windows error box.
//////////////////////////////////////////////////////////////////////

#ifndef ERRORBOX_H
#define ERRORBOX_H
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include "helpbox.h"
#include "cursor.h"

class ErrorBox
{
 private:
        Widget w;
        int charwidth;
        friend void DisplayHelp(Widget w, char* msg,
                                        XmAnyCallbackStruct *call_data);
        friend void RemoveError(Widget w, caddr_t client_data,
                                        XmAnyCallbackStruct *call_data);
 public:
        ErrorBox() { w = NULL; }
        ErrorBox(Widget& parent, int width = 30);
        ~ErrorBox();
        void SetHelp(char *);
        void Error(char *);
};

#endif
```

```
//////////////////////////////////////////////////////////////////////
//  File:       errorbox.C
//  Author:     David Thompson(??)
//  Description:
//              Implementation of a class that creates an
//              error message box.
//////////////////////////////////////////////////////////////////////

#include <string.h>
#include <iostream.h>
#include <Xm/MessageB.h>
#include "utils.h"
#include "errorbox.h"
#include "g_errorbox.h"

ErrorBox::ErrorBox(Widget& parent, int width)
{
        Widget appshell;
        charwidth = width;
        int n = 0;
        Arg args[3];
        XmString xms = (XmString) NULL;

        if(!getShell(parent, appshell))
        {
                cout << "Couldn't get application shell for error box." << endl;
                appshell = parent;
        }
        err_cursor.SetWidget(appshell);


        XtSetArg(args[n], XmNdialogStyle,
XmDIALOG_PRIMARY_APPLICATION_MODAL); n++;
        xms = XmStringCreate("Error", XmSTRING_DEFAULT_CHARSET);
        XtSetArg(args[n], XmNdialogTitle, xms); n++;
        w = XmCreateErrorDialog(appshell, "ErrorBox", args, n);
        if(xms) XmStringFree(xms);

        XtAddCallback(w, XmNokCallback, (XtCallbackProc) RemoveError, NULL);

        Widget tmpw;
        tmpw = XmMessageBoxGetChild(w, XmDIALOG_CANCEL_BUTTON);
        XtUnmanageChild(tmpw);
        tmpw = XmMessageBoxGetChild(w, XmDIALOG_HELP_BUTTON);
        XtUnmanageChild(tmpw);
}

void RemoveError(Widget w, caddr_t client_data, XmAnyCallbackStruct *call_data)
{
        if(w)
        {
                XtDestroyWidget(w);
        }
        err_cursor.Normal();
}


ErrorBox::~ErrorBox()
{
// Nothing to destroy.  Taken care of in RemoveError callback.
}

void ErrorBox::SetHelp(char *hlpmsg)
{
        Widget tmpw;
        char *tmp;

        tmp = new char[strlen(hlpmsg) + 1];
        strcpy(tmp, hlpmsg);

        tmpw = XmMessageBoxGetChild(w, XmDIALOG_HELP_BUTTON);
        XtManageChild(tmpw);

        XtAddCallback(w, XmNhelpCallback, (XtCallbackProc) DisplayHelp, tmp);
}

void DisplayHelp(Widget w, char *msg, XmAnyCallbackStruct *call_data)
{
        HelpBox help(w);
        help.Help(msg);
        delete msg;
}

void ErrorBox::Error(char *err)
{
        err_cursor.Error();

        XmString xms = (XmString) NULL;

        enum Space { now, notnow };
```

```
        Space space = notnow;
        char *temp, *temp1, *temp2 = err;
        int count = 0;
        int len = strlen(err);

        temp = new char[len+1];
        temp1 = temp;

        while(*temp2 != '\0')
        {
                if(space == now || count > charwidth)
                        space = now;

                if(*temp2 == '\n' || (space == now &&
                                (*temp2 == ' ' || *temp2 == '\t')))
                {
                        *temp1 = '\0';
                        xms = XmStringConcat(xms, XmStringCreate(temp,
XmSTRING_DEFAULT_CHARSET));
                        xms = XmStringConcat(xms, XmStringSeparatorCreate());
                        temp1 = temp;
                        count = 0;
                        space = notnow;
                }
                else
                {
                        *temp1 = *temp2;
                        count++; temp1++;
                }

                temp2++;
        }
        *temp1 = '\0';

        xms = XmStringConcat(xms, XmStringCreate(temp,
                                XmSTRING_DEFAULT_CHARSET));

        XtVaSetValues(w, XmNmessageString, xms, NULL);

        if(xms) XmStringFree(xms);

        XtManageChild(w);
}
```

```
#ifndef CURSOR_H
/************************************************************************
 *
 *
 *  Written 3/7/94 by David L. Thompson
 *
 *  The classes should be somewhat self explanatory.
 *
 *  c 1994 David L. Thompson, University of Montana
 *
 ************************************************************************/

#define CURSOR_H
#include <X11/Intrinsic.h>

// The following class is set up to change the cursor from a standard cursor to the
//  wait cursor.  This can then be called at any point.  To set up pass in
//  the toplevel widget.
class CursorClass {
private:
        Cursor standard, wait, dne, error, stop, other;
        Widget widget;
        Display* dpy;
        void SetCursor(Cursor c);
public:
        CursorClass(void) { widget = NULL; standard = (Cursor) NULL,
                wait = (Cursor) NULL, dne = (Cursor) NULL;
                        error = (Cursor) NULL; stop = (Cursor) NULL;
                                other = (Cursor) NULL; }

        CursorClass(Widget& w);
        ~CursorClass(void);
        void SetWidget(Widget& w);
        void SetShellWidget(Widget& w);
        void SetAppShellWidget(Widget &w);
        void Wait(void) { SetCursor(wait); }
        void Normal(void) { SetCursor(standard); }
        void DontEnter(void) { SetCursor(dne); }
        void Stop(void) { SetCursor(stop); }
        void Error(void) { SetCursor(error); }
};
#endif
```

```
//*****************************************************************
// FILE:        cursor.C
// AUTHOR:      David Thompson
// DESCRIPTION:
//
//   Implementaion of a class for displaying a variety of
//   different cursors that can be used by applications.
//*****************************************************************

#include <X11/cursorfont.h>
#include <iostream.h>
#include "cursor.h"
#include "utils.h"

// Include the bitmaps for several special cursors.

#include "bitmaps/errC.bmp"
#include "bitmaps/errCM.bmp"
#include "bitmaps/dneC.bmp"
#include "bitmaps/dneCM.bmp"
#include "bitmaps/stopC.bmp"
#include "bitmaps/stopCM.bmp"


CursorClass::CursorClass(Widget& w)
{
        standard = (Cursor) NULL;
        wait = (Cursor) NULL;
        error = (Cursor) NULL;
        dne = (Cursor) NULL;
        stop = (Cursor) NULL;
        other = (Cursor) NULL;
        SetWidget(w);
}

CursorClass::~CursorClass()
{
/*  I think that I should free the cursors when the variable leaves scope,
        but this is tending to cause a core dump.  Don't know why?

  Ans: The reason this is a core dump, is that the variables I was declaring
        are global.  When this destructor is called, the display has already
        been destroyed.  The cursors and the display are already gone.

  Prob:  There is no way that I can find to see if the display is still
        active.  If I could then I can have it free the cursors.  This is
        only valid if I create local instances of the class.  If I always
        use the global instance, then the cursors are only created once, and
        they are freed when the display is closed.

        if(dpy)
        {
                if(standard)
                        XFreeCursor(dpy, standard);
                if(wait)
                        XFreeCursor(dpy, wait);
                if(error)
                        XFreeCursor(dpy, error);
                if(dne)
                        XFreeCursor(dpy, dne);
                if(stop)
                        XFreeCursor(dpy, stop);
                if(other)
                        XFreeCursor(dpy, other);
        }
*/
}

void CursorClass::SetShellWidget(Widget& w)
{
        Widget temp;
        if(getShell(w, temp))
                SetWidget(temp);
        else
                cout << "Unable to find parent shell for cursor class" << endl;
}

void CursorClass::SetAppShellWidget(Widget& w)
{
        Widget temp;
        if(getApplicationShell(w, temp))
                SetWidget(temp);
        else
                cout << "Unable to find application shell for cursor class" <<
                                endl;
}

void CursorClass::SetWidget(Widget& w)
{
  Pixmap pix = (Pixmap) NULL, clip = (Pixmap) NULL;
```

```
widget = w;
dpy = XtDisplay(widget);
Window win = XtWindow(widget);
Screen *s = XtScreen(widget);
XColor c1, c2;

c1.red = 0; c1.blue = 0; c1.green = 0;
c2.red = 65535; c2.blue = 65535; c2.green = 65535;

if(!error)
{
        pix = XCreateBitmapFromData(dpy, win, errC_bits,
                                                errC_width, errC_height);
        clip = XCreateBitmapFromData(dpy, win, errCM_bits,
                                                errCM_width, errCM_height);

        error = XCreatePixmapCursor(dpy, pix, clip, &c1, &c2,
                                                        errC_x_hot, errC_y_hot);
}
if(clip) XFreePixmap(dpy, clip);
if(pix) XFreePixmap(dpy, pix);

if(!stop)
{
        pix = XCreateBitmapFromData(dpy, win, stopC_bits,
                                                stopC_width, stopC_height);
        clip = XCreateBitmapFromData(dpy, win, stopCM_bits,
                                                stopCM_width, stopCM_height);

        stop = XCreatePixmapCursor(dpy, pix, clip, &c1, &c2,
                                                        stopC_x_hot, stopC_y_hot);
}
if(clip) XFreePixmap(dpy, clip);
if(pix) XFreePixmap(dpy, pix);

if(!dne)
{
        pix = XCreateBitmapFromData(dpy, win, dneC_bits,
                                                dneC_width, dneC_height);
        clip = XCreateBitmapFromData(dpy, win, dneCM_bits,
                                                dneCM_width, dneCM_height);

        dne = XCreatePixmapCursor(dpy, pix, clip, &c1, &c2,
                                                        dneC_x_hot, dneC_y_hot);
}
if(clip) XFreePixmap(dpy, clip);
if(pix) XFreePixmap(dpy, pix);

if(!standard)
{
        standard = XCreateFontCursor(dpy, XC_left_ptr);
}

if(!wait)
{
        wait = XCreateFontCursor(dpy, XC_watch);
}

}

void CursorClass::SetCursor(Cursor c)
{
        if(widget == NULL)
        {
                cout << "Widget for cursor call not set." << endl;
        }
        else
        {
                if(c)
                {
                        XDefineCursor(dpy, XtWindow(widget), c);
                        XFlush(dpy);
                }
                else
                        cout << "Cursor not set in class." << endl;
        }
}
```

```
// DEM, DLG, ERDAS, MODFLOW Converters
//////////////////////////////////////////////////////////////////////
//  File:       convW.h
//  Author:     Kirk A. Moeller
//  Description:
//          Header file with defines and a prototype for a
//          function which creates the main application gui
//          as well as dialogs for the above converters
//////////////////////////////////////////////////////////////////////

#ifndef CONVW_H
#define CONVW_H

#include "XHeaders.h"

#define WI_APPLICATIONSHELL 0
#define WI_FORM 1
#define WI_CONVLBL 2
#define WI_TLBL1 3
#define WI_TLBL2 4
#define WI_TLBL3 5
#define WI_TLBL4 6
#define WI_WORKAREA 7
#define WI_DEMPB 8
#define WI_DLGPB 9
#define WI_ERDASPB 10
#define WI_MODFLOWPB 11
#define WI_ARCPB 12
#define WI_SHAPEPB 13
#define WI_ORACLEPB 14
#define WI_DXFPB 15
#define WI_IMGPB 16
#define WI_HELPPB1 17
#define WI_CANCELPB1 18
#define WI_WORKAREA1 19
#define WI_INFOLBL1 20
#define WI_INFOLBL2 21
#define WI_INFOLBL3 22
#define WI_INFOLBL4 23
#define WI_INFOLBL5 24
#define WI_INFOLBL6 25
#define WI_INFOLBL7 26
#define WI_INFOLBL8 27
#define WI_INFOLBL9 28
#define WI_SEPARATOR 29
#define WI_FCONV1 30
#define WI_FILECONVPU1 31
#define WI_VSEP_DLG 32
#define WI_V2SEP_DLG 33
#define WI_LFORM_DLG 34
#define WI_OPENLBL1 35
#define WI_FILESELBOXOPEN1 36
#define WI_MFORM_DLG 37
#define WI_STATLBL1 38
#define WI_STATTXT 39
#define WI_FILESAVETB1 40
#define WI_BINARYTB1 41
#define WI_OUTLBL_DLG 42
#define WI_HSEP_DLG 43
#define WI_H2SEP_DLG 44
#define WI_RFORM_DLG 45
#define WI_CONVLBL2 46
#define WI_FILESELBOXCONV1 47
#define WI_SELECTORBOX 48
#define WI_SELECTORPU 49
#define WI_SELECTORLBL 50
#define WI_DRAWINGAREA 51
#define WI_HEIGHTLBL 52
#define WI_WIDTHLBL 53
#define WI_XLBL 54
#define WI_YLBL 55
#define WI_XSIZELBL 56
#define WI_YSIZELBL 57
#define WI_SEPARATOR1 58
#define WI_CANCELPB 59
#define WI_HELPPB 60
#define WI_BLANKLBL 61
#define WI_OKPB 62
#define WI_FCONV 63
#define WI_FILECONVPU 64
#define WI_SEPARATOR2 65
#define WI_SEPARATOR3 66
#define WI_LFORM_DEM 67
#define WI_OPENLBL 68
#define WI_FILESELBOXOPEN 69
#define WI_MFORM_DEM 70
#define WI_STATLBL 71
#define WI_DEFAULTPB 72
#define WI_LAYERSLBL 73
#define WI_LAYERSSL 74
```

```
#define WI_SELECTORPB 75
#define WI_H3SEP_DEM 76
#define WI_H4SEP_DEM 77
#define WI_OUTLBL_DEM 78
#define WI_FILESAVETB 79
#define WI_BINARYTB 80
#define WI_HSEP_DEM 81
#define WI_H2SEP_DEM 82
#define WI_MINXFORM_DEM 83
#define WI_MINXLBL 84
#define WI_MINXTXT 85
#define WI_MINYFORM_DEM 86
#define WI_MINYLBL 87
#define WI_MINYTXT 88
#define WI_MAXXFORM_DEM 89
#define WI_MAXXLBL 90
#define WI_MAXXTXT 91
#define WI_MAXYFORM_DEM 92
#define WI_MAXYLBL 93
#define WI_MAXYTXT 94
#define WI_RFORM_DEM 95
#define WI_CONVLBL1 96
#define WI_FILESELBOXCONV 97
#define WI_WARNINGDIALOG 98

Widget tu_applicationshell_widget(char    * name,
                        Widget    parent,
                        Widget ** warr_ret);

Widget tu_FConv1_widget(char    * name,
                Widget    parent,
                Widget widget_array[]);

Widget tu_SelectorBox_widget(char    * name,
                    Widget    parent,
                    Widget widget_array[]);

Widget tu_FConv_widget(char    * name,
                Widget    parent,
                Widget widget_array[]);

Widget tu_warningDialog_widget(char    * name,
                    Widget    parent,
                    Widget widget_array[]);

#endif
```

```
// DEM, DLG, ERDAS, MODFLOW
////////////////////////////////////////////////////////////////////////
//   File:          convW.C
//   Author:        David Thompson, Dick Thompson, Petr Votava
//                  Kirk A. Moeller, and others.
//   Desription:
//                  The functions in the file create the dialog
//                  boxes used by the above converters as well
//                  as the clip dialog box and the main
//                  application dialog box.
////////////////////////////////////////////////////////////////////////

#include "convW.h"
#include "g_conv.h"
#include "callbacks.h"
#include "callbacksDlg.h"



/*
 * get_constraint_widget:
 ************************************************************/
static Widget get_constraint_widget(Widget child, Widget parent)
{
  Widget w;

  w = child;
  while (XtParent(w) != parent)
    w = XtParent(w);
  return (w);
}
```

```
/*****************************************************************
 *
 *  Main C code for presentation component
 *
 *****************************************************************/

/*****************************************************************
 * tu_applicationshell_widget:
 *****************************************************************/
Widget tu_applicationshell_widget(char    * name,
                        Widget    parent,
                        Widget ** warr_ret)
{
  Arg args[20];
  XmString xms = (XmString) NULL;
  int n;
  Display * display;
  Widget widget_array[100];

  /***************** object of type : ApplicationShell *****************/
  n = 0;
  XtSetArg(args[n], XtNtitle, "Data Explorer (DX) Converters v. 3.2"); n++;
  /* screen specified ? */
  if (sScreen != NULL) {
    XtSetArg(args[n], XtNscreen, sScreen); n++;
  }


  /* Setup argc & argv attribute */

  display = sDisplay;
  widget_array[WI_APPLICATIONSHELL] =
     XtAppCreateShell(PROGRAM_NAME, PROGRAM_CLASS, applicationShellWidgetClass,
display, args, n);

  /***************** form : XmForm *****************/
  n = 0;
  XtSetArg(args[n], XmNmarginWidth, 10); n++;
  XtSetArg(args[n], XmNmarginHeight, 5); n++;
  XtSetArg(args[n], XmNnoResize, True); n++;
  XtSetArg(args[n], XmNresizePolicy, XmRESIZE_ANY); n++;
  XtSetArg(args[n], XmNrubberPositioning, False); n++;
  widget_array[WI_FORM] =
    XmCreateForm(widget_array[WI_APPLICATIONSHELL], "form", args, n);

  /***************** ConvLBL : XmLabel *****************/
  n = 0;
  xms = XmStringCreate("Dataset Import and Conversion",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(args[n], XmNlabelString, xms); n++;
  XtSetArg(args[n], XmNresizable, True); n++;
  widget_array[WI_CONVLBL] =
    XmCreateLabel(widget_array[WI_FORM], "ConvLBL", args, n);

  if (xms) XmStringFree(xms);

  /***************** tlbl1 : XmLabel *****************/
  n = 0;
  xms  = XmStringCreate("University of Montana",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(args[n], XmNlabelString, xms); n++;
  XtSetArg(args[n], XmNresizable, True); n++;
  widget_array[WI_TLBL1] =
    XmCreateLabel(widget_array[WI_FORM], "tlbl1", args, n);

  if (xms) XmStringFree(xms);

  /***************** tlbl2 : XmLabel *****************/
  n = 0;
       xms        =    XmStringCreate("Distributed    Applications    and    Systems    Lab
(DASL)",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(args[n], XmNlabelString, xms); n++;
  XtSetArg(args[n], XmNresizable, True); n++;
  widget_array[WI_TLBL2] =
    XmCreateLabel(widget_array[WI_FORM], "tlbl2", args, n);

  if (xms) XmStringFree(xms);

  /***************** tlbl3 : XmLabel *****************/
  n = 0;
  xms  = XmStringCreate("AND",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(args[n], XmNlabelString, xms); n++;
  XtSetArg(args[n], XmNresizable, True); n++;
  widget_array[WI_TLBL3] =
    XmCreateLabel(widget_array[WI_FORM], "tlbl3", args, n);

  if (xms) XmStringFree(xms);

  /***************** tlbl4 : XmLabel *****************/
  n = 0;
```

```
            xms = XmStringCreate("IBM Visualization Data
Explorer",XmSTRING_DEFAULT_CHARSET);
 XtSetArg(args[n], XmNlabelString, xms); n++;
 XtSetArg(args[n], XmNresizable, True); n++;
 widget_array[WI_TLBL4] =
  XmCreateLabel(widget_array[WI_FORM], "tlbl4", args, n);

 if (xms) XmStringFree(xms);

 /***************** workArea : XmWorkArea *****************/
 n = 0;
 XtSetArg(args[n], XmNnumColumns, 1); n++;
 XtSetArg(args[n], XmNborderWidth, 1); n++;
 XtSetArg(args[n], XmNspacing, 5); n++;
 XtSetArg(args[n], XmNresizable, True); n++;
 widget_array[WI_WORKAREA] =
  XmCreateWorkArea(widget_array[WI_FORM], "workArea", args, n);

 /***************** DEMPB : XmPushButton *****************/
 n = 0;
 xms = XmStringCreate("DEM",XmSTRING_DEFAULT_CHARSET);
 XtSetArg(args[n], XmNlabelString, xms); n++;
 XtSetArg(args[n], XmNheight, 26); n++;
 XtSetArg(args[n], XmNrecomputeSize, False); n++;
 XtSetArg(args[n], XmNshadowThickness, 2); n++;
 XtSetArg(args[n], XmNhighlightThickness, 2); n++;
 widget_array[WI_DEMPB] =
  XmCreatePushButton(widget_array[WI_WORKAREA], "DEMPB", args, n);

 if (xms) XmStringFree(xms);

 /***************** DLGPB : XmPushButton *****************/
 n = 0;
 xms = XmStringCreate("DLG",XmSTRING_DEFAULT_CHARSET);
 XtSetArg(args[n], XmNlabelString, xms); n++;
 XtSetArg(args[n], XmNheight, 26); n++;
 XtSetArg(args[n], XmNrecomputeSize, False); n++;
 XtSetArg(args[n], XmNshadowThickness, 2); n++;
 XtSetArg(args[n], XmNhighlightThickness, 2); n++;
 widget_array[WI_DLGPB] =
  XmCreatePushButton(widget_array[WI_WORKAREA], "DLGPB", args, n);

 if (xms) XmStringFree(xms);

 /***************** ERDASPB : XmPushButton *****************/
 n = 0;
 xms = XmStringCreate("ERDAS",XmSTRING_DEFAULT_CHARSET);
 XtSetArg(args[n], XmNlabelString, xms); n++;
 XtSetArg(args[n], XmNheight, 26); n++;
 XtSetArg(args[n], XmNrecomputeSize, False); n++;
 XtSetArg(args[n], XmNshadowThickness, 2); n++;
 XtSetArg(args[n], XmNhighlightThickness, 2); n++;
 widget_array[WI_ERDASPB] =
  XmCreatePushButton(widget_array[WI_WORKAREA], "ERDASPB", args, n);

 if (xms) XmStringFree(xms);

 /***************** MODFLOWPB : XmPushButton *****************/
 n = 0;
 xms = XmStringCreate("MODFLOW",XmSTRING_DEFAULT_CHARSET);
 XtSetArg(args[n], XmNlabelString, xms); n++;
 XtSetArg(args[n], XmNheight, 26); n++;
 XtSetArg(args[n], XmNrecomputeSize, False); n++;
 XtSetArg(args[n], XmNshadowThickness, 2); n++;
 XtSetArg(args[n], XmNhighlightThickness, 2); n++;
 widget_array[WI_MODFLOWPB] =
  XmCreatePushButton(widget_array[WI_WORKAREA], "MODFLOWPB", args, n);

 if (xms) XmStringFree(xms);

 /***************** ARCPB : XmPushButton *****************/
 n = 0;
 xms = XmStringCreate("ARC",XmSTRING_DEFAULT_CHARSET);
 XtSetArg(args[n], XmNlabelString, xms); n++;
 XtSetArg(args[n], XmNheight, 26); n++;
 XtSetArg(args[n], XmNrecomputeSize, False); n++;
 XtSetArg(args[n], XmNshadowThickness, 2); n++;
 XtSetArg(args[n], XmNhighlightThickness, 2); n++;
 widget_array[WI_ARCPB] =
  XmCreatePushButton(widget_array[WI_WORKAREA], "ARCPB", args, n);

 if (xms) XmStringFree(xms);

 /***************** SHAPEPB : XmPushButton *****************/
 n = 0;
 xms = XmStringCreate("ArcShape",XmSTRING_DEFAULT_CHARSET);
 XtSetArg(args[n], XmNlabelString, xms); n++;
 XtSetArg(args[n], XmNheight, 26); n++;
 XtSetArg(args[n], XmNrecomputeSize, False); n++;
 XtSetArg(args[n], XmNshadowThickness, 2); n++;
```

```
XtSetArg(args[n], XmNhighlightThickness, 2); n++;
widget_array[WI_SHAPEPB] =
 XmCreatePushButton(widget_array[WI_WORKAREA], "SHAPEPB", args, n);

if (xms) XmStringFree(xms);

/***************** OraclePB : XmPushButton *****************/
n = 0;
xms  = XmStringCreate("Oracle",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNheight, 26); n++;
XtSetArg(args[n], XmNrecomputeSize, False); n++;
XtSetArg(args[n], XmNshadowThickness, 2); n++;
XtSetArg(args[n], XmNhighlightThickness, 2); n++;
widget_array[WI_ORACLEPB] =
 XmCreatePushButton(widget_array[WI_WORKAREA], "OraclePB", args, n);

if (xms) XmStringFree(xms);

/***************** DXFPB : XmPushButton *****************/
n = 0;
xms  = XmStringCreate("DXF",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNheight, 26); n++;
XtSetArg(args[n], XmNrecomputeSize, False); n++;
XtSetArg(args[n], XmNshadowThickness, 2); n++;
XtSetArg(args[n], XmNhighlightThickness, 2); n++;
widget_array[WI_DXFPB] =
 XmCreatePushButton(widget_array[WI_WORKAREA], "DXFPB", args, n);

if (xms) XmStringFree(xms);

/***************** IMGPB : XmPushButton *****************/
n = 0;
xms  = XmStringCreate("ERDAS/IMG",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNheight, 26); n++;
XtSetArg(args[n], XmNrecomputeSize, False); n++;
XtSetArg(args[n], XmNshadowThickness, 2); n++;
XtSetArg(args[n], XmNhighlightThickness, 2); n++;
widget_array[WI_IMGPB] =
 XmCreatePushButton(widget_array[WI_WORKAREA], "IMGPB", args, n);

if (xms) XmStringFree(xms);

/***************** HelpPB1 : XmPushButton *****************/
n = 0;
xms  = XmStringCreate("Help",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNwidth, 90); n++;
XtSetArg(args[n], XmNshadowThickness, 2); n++;
XtSetArg(args[n], XmNhighlightThickness, 2); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_HELPPB1] =
 XmCreatePushButton(widget_array[WI_FORM], "HelpPB1", args, n);

if (xms) XmStringFree(xms);

/***************** CancelPB1 : XmPushButton *****************/
n = 0;
xms  = XmStringCreate("Quit",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNwidth, 90); n++;
XtSetArg(args[n], XmNshadowThickness, 2); n++;
XtSetArg(args[n], XmNhighlightThickness, 2); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_CANCELPB1] =
 XmCreatePushButton(widget_array[WI_FORM], "CancelPB1", args, n);

if (xms) XmStringFree(xms);

/***************** workArea1 : XmWorkArea *****************/
n = 0;
XtSetArg(args[n], XmNspacing, 5); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_WORKAREA1] =
 XmCreateWorkArea(widget_array[WI_FORM], "workArea1", args, n);

/***************** InfoLBL1 : XmLabel *****************/
n = 0;
xms  = XmStringCreate("USGS/DEM To DX",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNalignment, XmALIGNMENT_BEGINNING); n++;
XtSetArg(args[n], XmNheight, 26); n++;
XtSetArg(args[n], XmNrecomputeSize, False); n++;
widget_array[WI_INFOLBL1] =
 XmCreateLabel(widget_array[WI_WORKAREA1], "InfoLBL1", args, n);

if (xms) XmStringFree(xms);
```

```
/***************** InfoLBL2 : XmLabel *****************/
n = 0;
xms = XmStringCreate("USGS/DLG To DX",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNalignment, XmALIGNMENT_BEGINNING); n++;
XtSetArg(args[n], XmNheight, 26); n++;
XtSetArg(args[n], XmNrecomputeSize, False); n++;
widget_array[WI_INFOLBL2] =
 XmCreateLabel(widget_array[WI_WORKAREA1], "InfoLBL2", args, n);

if (xms) XmStringFree(xms);

/***************** InfoLBL3 : XmLabel *****************/
n = 0;
xms = XmStringCreate("ERDAS/LAN To DX",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNalignment, XmALIGNMENT_BEGINNING); n++;
XtSetArg(args[n], XmNheight, 26); n++;
XtSetArg(args[n], XmNrecomputeSize, False); n++;
widget_array[WI_INFOLBL3] =
 XmCreateLabel(widget_array[WI_WORKAREA1], "InfoLBL3", args, n);

if (xms) XmStringFree(xms);

/***************** InfoLBL4 : XmLabel *****************/
n = 0;
        xms = XmStringCreate("USGS/ModFlow (typical) To
DX",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNalignment, XmALIGNMENT_BEGINNING); n++;
XtSetArg(args[n], XmNheight, 26); n++;
XtSetArg(args[n], XmNrecomputeSize, False); n++;
widget_array[WI_INFOLBL4] =
 XmCreateLabel(widget_array[WI_WORKAREA1], "InfoLBL4", args, n);

if (xms) XmStringFree(xms);

/***************** InfoLBL5 : XmLabel *****************/
n = 0;
xms = XmStringCreate("ESRI ARC/INFO To DX",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNalignment, XmALIGNMENT_BEGINNING); n++;
XtSetArg(args[n], XmNheight, 26); n++;
XtSetArg(args[n], XmNrecomputeSize, False); n++;
widget_array[WI_INFOLBL5] =
 XmCreateLabel(widget_array[WI_WORKAREA1], "InfoLBL5", args, n);

if (xms) XmStringFree(xms);

/***************** InfoLBL6 : XmLabel *****************/
n = 0;
xms = XmStringCreate("ArcShape to DX [beta]",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNalignment, XmALIGNMENT_BEGINNING); n++;
XtSetArg(args[n], XmNheight, 26); n++;
XtSetArg(args[n], XmNrecomputeSize, False); n++;
widget_array[WI_INFOLBL6] =
 XmCreateLabel(widget_array[WI_WORKAREA1], "InfoLBL6", args, n);

if (xms) XmStringFree(xms);

/***************** InfoLBL7 : XmLabel *****************/
n = 0;
xms = XmStringCreate("Oracle to DX [beta]",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNalignment, XmALIGNMENT_BEGINNING); n++;
XtSetArg(args[n], XmNheight, 26); n++;
XtSetArg(args[n], XmNrecomputeSize, False); n++;
widget_array[WI_INFOLBL7] =
 XmCreateLabel(widget_array[WI_WORKAREA1], "InfoLBL7", args, n);

if (xms) XmStringFree(xms);

/***************** InfoLBL8 : XmLabel *****************/
n = 0;
xms = XmStringCreate("DXF to DX",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNalignment, XmALIGNMENT_BEGINNING); n++;
XtSetArg(args[n], XmNheight, 26); n++;
XtSetArg(args[n], XmNrecomputeSize, False); n++;
widget_array[WI_INFOLBL8] =
 XmCreateLabel(widget_array[WI_WORKAREA1], "InfoLBL8", args, n);

if (xms) XmStringFree(xms);

/***************** InfoLBL9 : XmLabel *****************/
n = 0;
xms = XmStringCreate("ERDAS/IMAGINE to DX [beta]",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNalignment, XmALIGNMENT_BEGINNING); n++;
```

```
XtSetArg(args[n], XmNheight, 26); n++;
XtSetArg(args[n], XmNrecomputeSize, False); n++;
widget_array[WI_INFOLBL9] =
  XmCreateLabel(widget_array[WI_WORKAREA1], "InfoLBL9", args, n);

if (xms) XmStringFree(xms);

/***************** separator : XmSeparator *****************/
n = 0;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_SEPARATOR] =
  XmCreateSeparator(widget_array[WI_FORM], "separator", args, n);


/***************** form : XmForm *****************/
n = 0;
XtSetArg(args[n], XmNdefaultButton, widget_array[WI_DEMPB]); n++;
XtSetValues(widget_array[WI_FORM], args, n);


/***************** ConvLBL : XmLabel *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNleftOffset, 140); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopOffset, 10); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNrightOffset, 140); n++;
XtSetValues(widget_array[WI_CONVLBL], args, n);

XtManageChild(widget_array[WI_CONVLBL]);

/***************** tlbl1 : XmLabel *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNleftOffset, 170); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_CONVLBL]); n++;
XtSetArg(args[n], XmNtopOffset, 20); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNrightOffset, 170); n++;
XtSetValues(widget_array[WI_TLBL1], args, n);

XtManageChild(widget_array[WI_TLBL1]);

/***************** tlbl2 : XmLabel *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNleftOffset, 85); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_TLBL1]); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNrightOffset, 85); n++;
XtSetValues(widget_array[WI_TLBL2], args, n);

XtManageChild(widget_array[WI_TLBL2]);

/***************** tlbl3 : XmLabel *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNleftOffset, 230); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_TLBL2]); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNrightOffset, 230); n++;
XtSetValues(widget_array[WI_TLBL3], args, n);

XtManageChild(widget_array[WI_TLBL3]);

/***************** tlbl4 : XmLabel *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNleftOffset, 140); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_TLBL3]); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNrightOffset, 140); n++;
XtSetValues(widget_array[WI_TLBL4], args, n);

XtManageChild(widget_array[WI_TLBL4]);

/***************** workArea : XmWorkArea *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
```

```
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_CONVLBL]); n++;
XtSetArg(args[n], XmNtopOffset, 135); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetValues(widget_array[WI_WORKAREA], args, n);

XtManageChild(widget_array[WI_DEMPB]);
XtManageChild(widget_array[WI_DLGPB]);
XtManageChild(widget_array[WI_ERDASPB]);
XtManageChild(widget_array[WI_MODFLOWPB]);
XtManageChild(widget_array[WI_ARCPB]);
XtManageChild(widget_array[WI_SHAPEPB]);
XtManageChild(widget_array[WI_ORACLEPB]);
XtManageChild(widget_array[WI_DXFPB]);
XtManageChild(widget_array[WI_IMGPB]);
XtManageChild(widget_array[WI_WORKAREA]);

/***************** HelpPBI : XmPushButton *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightOffset, 10); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetValues(widget_array[WI_HELPPBI], args, n);

XtManageChild(widget_array[WI_HELPPB1]);

/***************** CancelPBI : XmPushButton *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNleftOffset, 10); n++;
XtSetValues(widget_array[WI_CANCELPB1], args, n);

XtManageChild(widget_array[WI_CANCELPBI]);

/***************** workAreaI : XmWorkArea *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_WORKAREA]); n++;
XtSetArg(args[n], XmNleftOffset, 50); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_WORKAREA]); n++;
XtSetValues(widget_array[WI_WORKAREAI], args, n);

XtManageChild(widget_array[WI_INFOLBL1]);
XtManageChild(widget_array[WI_INFOLBL2]);
XtManageChild(widget_array[WI_INFOLBL3]);
XtManageChild(widget_array[WI_INFOLBL4]);
XtManageChild(widget_array[WI_INFOLBL5]);
XtManageChild(widget_array[WI_INFOLBL6]);
XtManageChild(widget_array[WI_INFOLBL7]);
XtManageChild(widget_array[WI_INFOLBL8]);
XtManageChild(widget_array[WI_INFOLBL9]);
XtManageChild(widget_array[WI_WORKAREAI]);

/***************** separator : XmSeparator *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNbottomWidget, widget_array[WI_CANCELPBI]); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_WORKAREA]); n++;
XtSetArg(args[n], XmNtopOffset, 40); n++;
XtSetValues(widget_array[WI_SEPARATOR], args, n);

XtManageChild(widget_array[WI_SEPARATOR]);
XtManageChild(widget_array[WI_FORM]);

/** Create dialogs **/
/** Create DEM Converter dialog box **/
tu_FConv_widget("FConv",widget_array[WI_FORM],widget_array);

/** Create the selector dialog box (i.e. the clipping box) **/
/** WI_SELECTORPB is defined in the above function.      **/
tu_SelectorBox_widget("SelectorBox",widget_array[WI_SELECTORPB],widget_array);

/** Create DLG Converter dialog box **/
tu_FConv1_widget("FConvI",widget_array[WI_FORM],widget_array);

/** Create the overwrite warning dialog box **/
tu_warningDialog_widget("WarningBox",widget_array[WI_FORM],widget_array);
```

```
 /*
  *  Allocate memory for the widget array to return
  */
  *warr_ret = (Widget *) malloc(sizeof(Widget)*100);
  (void) memcpy((char *)*warr_ret,
            (char *)widget_array,
        sizeof(Widget)*100);

 AddCallbacks(*warr_ret);
 AddCallbacksDLG(*warr_ret);

 /*
  *  Return the first created widget.
  */
 return widget_array[WI_APPLICATIONSHELL];
}


/*************************************************************
 * tu_FConv1_widget:
 *************************************************************/
Widget tu_FConv1_widget(char    * name,
                Widget    parent,
                Widget widget_array[])
{
 Arg args[22];
 Widget tmpw1;
 XmString xms = (XmString) NULL;
 XmString xms1 = (XmString) NULL;
 int n;

 /***************** object of type : XmDialogShell *****************/
 n = 0;
 XtSetArg(args[n], XtNallowShellResize, True); n++;
 XtSetArg(args[n], XtNtitle, "DLG Converter"); n++;
 XtSetArg(args[n], XmNdeleteResponse, XmDO_NOTHING); n++;
 widget_array[WI_FCONV1] =
  XmCreateDialogShell(parent, name, args, n);

 /***************** FileConvPU1 : XmForm *****************/
 n = 0;
 XtSetArg(args[n], XmNrubberPositioning, False); n++;
 XtSetArg(args[n], XmNresizePolicy, XmRESIZE_ANY); n++;
 XtSetArg(args[n], XmNautoUnmanage, False); n++;
 XtSetArg(args[n], XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL); n++;
 XtSetArg(args[n], XmNnoResize, True); n++;
 widget_array[WI_FILECONVPU1] =
  XmCreateForm(widget_array[WI_FCONV1], "FileConvPU1", args, n);

 /***************** vSEP_DLG : XmSeparator *****************/
 n = 0;
 XtSetArg(args[n], XmNorientation, XmVERTICAL); n++;
 XtSetArg(args[n], XmNresizable, True); n++;
 widget_array[WI_VSEP_DLG] =
  XmCreateSeparator(widget_array[WI_FILECONVPU1], "vSEP_DLG", args, n);

 /***************** v2SEP_DLG : XmSeparator *****************/
 n = 0;
 XtSetArg(args[n], XmNorientation, XmVERTICAL); n++;
 XtSetArg(args[n], XmNresizable, True); n++;
 widget_array[WI_V2SEP_DLG] =
  XmCreateSeparator(widget_array[WI_FILECONVPU1], "v2SEP_DLG", args, n);

 /***************** LForm_DLG : XmForm *****************/
 n = 0;
 XtSetArg(args[n], XmNrubberPositioning, False); n++;
 XtSetArg(args[n], XmNresizePolicy, XmRESIZE_ANY); n++;
 XtSetArg(args[n], XmNresizable, True); n++;
 widget_array[WI_LFORM_DLG] =
  XmCreateForm(widget_array[WI_FILECONVPU1], "LForm_DLG", args, n);

 /***************** OpenLBL1 : XmLabel *****************/
 n = 0;
 xms  = XmStringCreate("Select File to Convert",XmSTRING_DEFAULT_CHARSET);
 XtSetArg(args[n], XmNlabelString, xms); n++;
 XtSetArg(args[n], XmNresizable, True); n++;
 widget_array[WI_OPENLBL1] =
  XmCreateLabel(widget_array[WI_LFORM_DLG], "OpenLBL1", args, n);

 if (xms) XmStringFree(xms);

 /***************** fileSelBoxOpen1 : XmFileSelectionBox *****************/
 n = 0;
 XtSetArg(args[n], XmNresizePolicy, XmRESIZE_NONE); n++;
 XtSetArg(args[n], XmNwidth, 320); n++;
 xms  = XmStringCreate("*.dlg",XmSTRING_DEFAULT_CHARSET);
 XtSetArg(args[n], XmNdirMask, xms); n++;
 xms1  = XmStringCreate("Quit",XmSTRING_DEFAULT_CHARSET);
 XtSetArg(args[n], XmNcancelLabelString, xms1); n++;
 XtSetArg(args[n], XmNresizable, False); n++;
```

```
widget_array[WI_FILESELBOXOPEN1] =
 XmCreateFileSelectionBox(widget_array[WI_LFORM_DLG], "fileSelBoxOpen1", args, n);

if (xms) XmStringFree(xms);
if (xms1) XmStringFree(xms1);

/***************** MForm_DLG : XmForm *****************/
n = 0;
XtSetArg(args[n], XmNrubberPositioning, False); n++;
XtSetArg(args[n], XmNresizePolicy, XmRESIZE_ANY); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_MFORM_DLG] =
 XmCreateForm(widget_array[WI_FILECONVPU1], "MForm_DLG", args, n);

/***************** StatLBL1 : XmLabel *****************/
n = 0;
xms  = XmStringCreate("Statistics",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_STATLBL1] =
 XmCreateLabel(widget_array[WI_MFORM_DLG], "StatLBL1", args, n);

if (xms) XmStringFree(xms);

/***************** StatTXT : XmText *****************/
n = 0;
XtSetArg(args[n], XmNwidth, 200); n++;
XtSetArg(args[n], XmNheight, 270); n++;
XtSetArg(args[n], XmNeditMode, XmMULTI_LINE_EDIT); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_STATTXT] =
 XmCreateText(widget_array[WI_MFORM_DLG], "StatTXT", args, n);

/***************** fileSaveTB1 : XmToggleButton *****************/
n = 0;
xms  = XmStringCreate("File Output",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNshadowThickness, 1); n++;
XtSetArg(args[n], XmNset, True); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_FILESAVETB1] =
 XmCreateToggleButton(widget_array[WI_MFORM_DLG], "fileSaveTB1", args, n);

if (xms) XmStringFree(xms);

/***************** binaryTB1 : XmToggleButton *****************/
n = 0;
xms  = XmStringCreate("Binary Output",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNshadowThickness, 1); n++;
XtSetArg(args[n], XmNset, True); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_BINARYTB1] =
 XmCreateToggleButton(widget_array[WI_MFORM_DLG], "binaryTB1", args, n);

if (xms) XmStringFree(xms);

/***************** outLBL_DLG : XmLabel *****************/
n = 0;
XtSetArg(args[n], XmNalignment, XmALIGNMENT_BEGINNING); n++;
xms  = XmStringCreate("Output Options:",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_OUTLBL_DLG] =
 XmCreateLabel(widget_array[WI_MFORM_DLG], "outLBL_DLG", args, n);

if (xms) XmStringFree(xms);

/***************** hSEP_DLG : XmSeparator *****************/
n = 0;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_HSEP_DLG] =
 XmCreateSeparator(widget_array[WI_MFORM_DLG], "hSEP_DLG", args, n);

/***************** h2SEP_DLG : XmSeparator *****************/
n = 0;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_H2SEP_DLG] =
 XmCreateSeparator(widget_array[WI_MFORM_DLG], "h2SEP_DLG", args, n);

/***************** RForm_DLG : XmForm *****************/
n = 0;
XtSetArg(args[n], XmNrubberPositioning, False); n++;
XtSetArg(args[n], XmNresizePolicy, XmRESIZE_ANY); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_RFORM_DLG] =
 XmCreateForm(widget_array[WI_FILECONVPU1], "RForm_DLG", args, n);

/***************** ConvLBL2 : XmLabel *****************/
n = 0;
```

```
xms  = XmStringCreate("Select File to Save As",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNalignment, XmALIGNMENT_BEGINNING); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_CONVLBL2] =
  XmCreateLabel(widget_array[WI_RFORM_DLG], "ConvLBL2", args, n);

if (xms) XmStringFree(xms);

/****************** fileSelBoxConv1 : XmFileSelectionBox *****************/
n = 0;
XtSetArg(args[n], XmNwidth, 320); n++;
XtSetArg(args[n], XmNresizePolicy, XmRESIZE_NONE); n++;
xms  = XmStringCreate("*.dx",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNdirMask, xms); n++;
XtSetArg(args[n], XmNresizable, False); n++;
widget_array[WI_FILESELBOXCONV1] =
  XmCreateFileSelectionBox(widget_array[WI_RFORM_DLG], "fileSelBoxConv1", args, n);

if (xms) XmStringFree(xms);


/****************** vSEP_DLG : XmSeparator *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_LFORM_DLG]); n++;
XtSetValues(widget_array[WI_VSEP_DLG], args, n);

XtManageChild(widget_array[WI_VSEP_DLG]);

/****************** v2SEP_DLG : XmSeparator *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_MFORM_DLG]); n++;
XtSetValues(widget_array[WI_V2SEP_DLG], args, n);

XtManageChild(widget_array[WI_V2SEP_DLG]);

/****************** LForm_DLG : XmForm *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetValues(widget_array[WI_LFORM_DLG], args, n);


/****************** OpenLBL1 : XmLabel *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopOffset, 10); n++;
XtSetValues(widget_array[WI_OPENLBL1], args, n);

XtManageChild(widget_array[WI_OPENLBL1]);

/****************** fileSelBoxOpen1 : XmFileSelectionBox *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_OPENLBL1]); n++;
XtSetValues(widget_array[WI_FILESELBOXOPEN1], args, n);

XtManageChild(widget_array[WI_FILESELBOXOPEN1]);
XtManageChild(widget_array[WI_LFORM_DLG]);

/****************** MForm_DLG : XmForm *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_VSEP_DLG]); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetValues(widget_array[WI_MFORM_DLG], args, n);


/****************** StatLBL1 : XmLabel *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
```

```
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopOffset, 10); n++;
XtSetValues(widget_array[WI_STATLBL1], args, n);

XtManageChild(widget_array[WI_STATLBL1]);

/***************** StatTXT : XmText *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_STATLBL1]); n++;
XtSetArg(args[n], XmNtopOffset, 20); n++;
XtSetValues(widget_array[WI_STATTXT], args, n);

XtManageChild(widget_array[WI_STATTXT]);

/***************** fileSaveTB1 : XmToggleButton *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_BINARYTB1]); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_BINARYTB1]); n++;
XtSetArg(args[n], XmNrightWidget, widget_array[WI_BINARYTB1]); n++;
XtSetValues(widget_array[WI_FILESAVETB1], args, n);

XtManageChild(widget_array[WI_FILESAVETB1]);

/***************** binaryTB1 : XmToggleButton *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_OUTLBL_DLG]); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_OUTLBL_DLG]); n++;
XtSetArg(args[n], XmNleftOffset, 30); n++;
XtSetValues(widget_array[WI_BINARYTB1], args, n);

XtManageChild(widget_array[WI_BINARYTB1]);

/***************** outLBL_DLG : XmLabel *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_HSEP_DLG]); n++;
XtSetValues(widget_array[WI_OUTLBL_DLG], args, n);

XtManageChild(widget_array[WI_OUTLBL_DLG]);

/***************** hSEP_DLG : XmSeparator *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_STATLBL1]); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNrightWidget, widget_array[WI_STATLBL1]); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_STATTXT]); n++;
XtSetArg(args[n], XmNtopOffset, 15); n++;
XtSetValues(widget_array[WI_HSEP_DLG], args, n);

XtManageChild(widget_array[WI_HSEP_DLG]);

/***************** h2SEP_DLG : XmSeparator *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_FILESAVETB1]); n++;
XtSetValues(widget_array[WI_H2SEP_DLG], args, n);

XtManageChild(widget_array[WI_H2SEP_DLG]);
XtManageChild(widget_array[WI_MFORM_DLG]);

/***************** RForm_DLG : XmForm *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_V2SEP_DLG]); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetValues(widget_array[WI_RFORM_DLG], args, n);
```

```
 /***************** ConvLBL2 : XmLabel *****************/
 n = 0;
 XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
 XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
 XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
 XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
 XtSetArg(args[n], XmNtopOffset, 10); n++;
 XtSetValues(widget_array[WI_CONVLBL2], args, n);

 XtManageChild(widget_array[WI_CONVLBL2]);

 /***************** fileSelBoxConv1 : XmFileSelectionBox *****************/
 n = 0;
        tmpw1     =     XmSelectionBoxGetChild(widget_array[WI_FILESELBOXCONV1],
XmDIALOG_OK_BUTTON);
 XtUnmanageChild(tmpw1);
        tmpw1     =     XmSelectionBoxGetChild(widget_array[WI_FILESELBOXCONV1],
XmDIALOG_CANCEL_BUTTON);
 XtUnmanageChild(tmpw1);
        tmpw1     =     XmSelectionBoxGetChild(widget_array[WI_FILESELBOXCONV1],
XmDIALOG_HELP_BUTTON);
 XtUnmanageChild(tmpw1);
 XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
 XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
 XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
 XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
 XtSetArg(args[n], XmNtopWidget, widget_array[WI_CONVLBL2]); n++;
 XtSetValues(widget_array[WI_FILESELBOXCONV1], args, n);

 XtManageChild(widget_array[WI_FILESELBOXCONV1]);
 XtManageChild(widget_array[WI_RFORM_DLG]);

 /*
 */
 /*
  *  Return the first created widget.
  */
 return widget_array[WI_FCONV1];
}


/*************************************************************
 * tu_SelectorBox_widget:
 *************************************************************/
```

```
Widget tu_SelectorBox_widget(char    * name,
                    Widget    parent,
                    Widget widget_array[])
{
 Arg args[20];
 XmString xms = (XmString) NULL;
 int n;

 /***************** object of type : XmDialogShell *****************/
 widget_array[WI_SELECTORBOX] =
  XmCreateDialogShell(parent, name, NULL, 0);

 /***************** SelectorPU : XmForm *****************/
 n = 0;
 XtSetArg(args[n], XmNrubberPositioning, False); n++;
 XtSetArg(args[n], XmNresizePolicy, XmRESIZE_ANY); n++;
 XtSetArg(args[n], XmNautoUnmanage, False); n++;
 XtSetArg(args[n], XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL); n++;
 xms  = XmStringCreate("Selector",XmSTRING_DEFAULT_CHARSET);
 XtSetArg(args[n], XmNdialogTitle, xms); n++;
 widget_array[WI_SELECTORPU] =
   XmCreateForm(widget_array[WI_SELECTORBOX], "SelectorPU", args, n);

 if (xms) XmStringFree(xms);

 /***************** selectorLBL : XmLabel *****************/
 n = 0;
 xms  = XmStringCreate("Selector",XmSTRING_DEFAULT_CHARSET);
 XtSetArg(args[n], XmNlabelString, xms); n++;
 XtSetArg(args[n], XmNresizable, True); n++;
 widget_array[WI_SELECTORLBL] =
   XmCreateLabel(widget_array[WI_SELECTORPU], "selectorLBL", args, n);

 if (xms) XmStringFree(xms);

 /***************** drawingArea : XmDrawingArea *****************/
 n = 0;
 XtSetArg(args[n], XmNborderWidth, 2); n++;
 XtSetArg(args[n], XmNwidth, 50); n++;
 XtSetArg(args[n], XmNheight, 50); n++;
 XtSetArg(args[n], XmNresizable, True); n++;
 widget_array[WI_DRAWINGAREA] =
   XmCreateDrawingArea(widget_array[WI_SELECTORPU], "drawingArea", args, n);

 /***************** heightLBL : XmLabel *****************/
```

```
n = 0;
XtSetArg(args[n], XmNalignment, XmALIGNMENT_BEGINNING); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_HEIGHTLBL] =
 XmCreateLabel(widget_array[WI_SELECTORPU], "heightLBL", args, n);

/***************** widthLBL : XmLabel *****************/
n = 0;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_WIDTHLBL] =
 XmCreateLabel(widget_array[WI_SELECTORPU], "widthLBL", args, n);

/***************** XLBL : XmLabel *****************/
n = 0;
xms = XmStringCreate("X: ",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_XLBL] =
 XmCreateLabel(widget_array[WI_SELECTORPU], "XLBL", args, n);

if (xms) XmStringFree(xms);

/***************** YLBL : XmLabel *****************/
n = 0;
xms = XmStringCreate("Y: ",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_YLBL] =
 XmCreateLabel(widget_array[WI_SELECTORPU], "YLBL", args, n);

if (xms) XmStringFree(xms);

/***************** XSizeLBL : XmLabel *****************/
n = 0;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_XSIZELBL] =
 XmCreateLabel(widget_array[WI_SELECTORPU], "XSizeLBL", args, n);

/***************** YSizeLBL : XmLabel *****************/
n = 0;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_YSIZELBL] =
 XmCreateLabel(widget_array[WI_SELECTORPU], "YSizeLBL", args, n);

/***************** separator : XmSeparator *****************/
n = 0;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_SEPARATOR1] =
 XmCreateSeparator(widget_array[WI_SELECTORPU], "separator", args, n);

/***************** CancelPB : XmPushButton *****************/
n = 0;
xms = XmStringCreate("Cancel",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_CANCELPB] =
 XmCreatePushButton(widget_array[WI_SELECTORPU], "CancelPB", args, n);

if (xms) XmStringFree(xms);

/***************** HelpPB : XmPushButton *****************/
n = 0;
xms = XmStringCreate("Help",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNresizable, False); n++;
widget_array[WI_HELPPB] =
 XmCreatePushButton(widget_array[WI_SELECTORPU], "HelpPB", args, n);

if (xms) XmStringFree(xms);

/***************** BlankLBL : XmLabel *****************/
n = 0;
xms = XmStringCreate(" ",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_BLANKLBL] =
 XmCreateLabel(widget_array[WI_SELECTORPU], "BlankLBL", args, n);

if (xms) XmStringFree(xms);

/***************** OKPB : XmPushButton *****************/
n = 0;
XtSetArg(args[n], XmNx, 196); n++;
XtSetArg(args[n], XmNy, 47); n++;
xms = XmStringCreate("OK",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNwidth, 40); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_OKPB] =
 XmCreatePushButton(widget_array[WI_SELECTORPU], "OKPB", args, n);
```

```
if (xms) XmStringFree(xms);


/***************** SelectorPU : XmForm *****************/
n = 0;
XtSetArg(args[n], XmNdefaultButton, widget_array[WI_OKPB]); n++;
XtSetArg(args[n], XmNcancelButton, widget_array[WI_CANCELPB]); n++;
XtSetValues(widget_array[WI_SELECTORPU], args, n);


/***************** selectorLBL : XmLabel *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNleftOffset, 5); n++;
XtSetArg(args[n], XmNtopOffset, 5); n++;
XtSetValues(widget_array[WI_SELECTORLBL], args, n);

XtManageChild(widget_array[WI_SELECTORLBL]);

/***************** drawingArea : XmDrawingArea *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_SELECTORLBL]); n++;
XtSetArg(args[n], XmNleftOffset, 5); n++;
XtSetValues(widget_array[WI_DRAWINGAREA], args, n);

XtManageChild(widget_array[WI_DRAWINGAREA]);

/***************** heightLBL : XmLabel *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_DRAWINGAREA]); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_DRAWINGAREA]); n++;
XtSetArg(args[n], XmNbottomWidget, widget_array[WI_DRAWINGAREA]); n++;
XtSetValues(widget_array[WI_HEIGHTLBL], args, n);

XtManageChild(widget_array[WI_HEIGHTLBL]);

/***************** widthLBL : XmLabel *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_DRAWINGAREA]); n++;
XtSetArg(args[n], XmNrightWidget, widget_array[WI_DRAWINGAREA]); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_DRAWINGAREA]); n++;
XtSetValues(widget_array[WI_WIDTHLBL], args, n);

XtManageChild(widget_array[WI_WIDTHLBL]);

/***************** XLBL : XmLabel *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopOffset, 5); n++;
XtSetArg(args[n], XmNleftOffset, 50); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_WIDTHLBL]); n++;
XtSetValues(widget_array[WI_XLBL], args, n);

XtManageChild(widget_array[WI_XLBL]);

/***************** YLBL : XmLabel *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_XLBL]); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_XLBL]); n++;
XtSetValues(widget_array[WI_YLBL], args, n);

XtManageChild(widget_array[WI_YLBL]);

/***************** XSizeLBL : XmLabel *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_XLBL]); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
```

```
XtSetArg(args[n], XmNtopAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_XLBL]); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNbottomWidget, widget_array[WI_XLBL]); n++;
XtSetValues(widget_array[WI_XSIZELBL], args, n);

XtManageChild(widget_array[WI_XSIZELBL]);

/****************** YSizeLBL : XmLabel ******************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_YLBL]); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_YLBL]); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNbottomWidget, widget_array[WI_YLBL]); n++;
XtSetValues(widget_array[WI_YSIZELBL], args, n);

XtManageChild(widget_array[WI_YSIZELBL]);

/****************** separator : XmSeparator ******************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_YLBL]); n++;
XtSetArg(args[n], XmNbottomWidget, widget_array[WI_OKPB]); n++;
XtSetValues(widget_array[WI_SEPARATOR1], args, n);

XtManageChild(widget_array[WI_SEPARATOR1]);

/****************** CancelPB : XmPushButton ******************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNbottomWidget, widget_array[WI_OKPB]); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_OKPB]); n++;
XtSetValues(widget_array[WI_CANCELPB], args, n);

XtManageChild(widget_array[WI_CANCELPB]);

/****************** HelpPB : XmPushButton ******************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_CANCELPB]); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_OKPB]); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNbottomWidget, widget_array[WI_OKPB]); n++;
XtSetValues(widget_array[WI_HELPPB], args, n);

XtManageChild(widget_array[WI_HELPPB]);

/****************** BlankLBL : XmLabel ******************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_HELPPB]); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNrightWidget, widget_array[WI_OKPB]); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_OKPB]); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNbottomWidget, widget_array[WI_OKPB]); n++;
XtSetValues(widget_array[WI_BLANKLBL], args, n);

XtManageChild(widget_array[WI_BLANKLBL]);

/****************** OKPB : XmPushButton ******************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetValues(widget_array[WI_OKPB], args, n);

XtManageChild(widget_array[WI_OKPB]);

/*
*/
/*
 *  Return the first created widget.
 */
return widget_array[WI_SELECTORBOX];
}
```

```
/*************************************************************
 * tu_FConv_widget:
 *************************************************************/
Widget tu_FConv_widget(char    * name,
                Widget    parent,
                Widget widget_array[])
{
 Arg args[22];
 Arg pargs[22];
 Widget tmpw;
 Widget tmpw1;
 XmString xms = (XmString) NULL;
 int n;
 int pn;

 /***************** object of type : XmDialogShell *****************/
 n = 0;
 XtSetArg(args[n], XtNallowShellResize, True); n++;
 XtSetArg(args[n], XtNtitle, "DEM Converter"); n++;
 XtSetArg(args[n], XmNdeleteResponse, XmDO_NOTHING); n++;
 widget_array[WI_FCONV] =
  XmCreateDialogShell(parent, name, args, n);

 /***************** FileConvPU : XmForm *****************/
 n = 0;
 XtSetArg(args[n], XmNrubberPositioning, False); n++;
 XtSetArg(args[n], XmNresizePolicy, XmRESIZE_ANY); n++;
 XtSetArg(args[n], XmNautoUnmanage, False); n++;
 XtSetArg(args[n], XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL); n++;
 xms  = XmStringCreate("DEM Converter",XmSTRING_DEFAULT_CHARSET);
 XtSetArg(args[n], XmNdialogTitle, xms); n++;
 XtSetArg(args[n], XmNnoResize, True); n++;
 widget_array[WI_FILECONVPU] =
   XmCreateForm(widget_array[WI_FCONV], "FileConvPU", args, n);

 if (xms) XmStringFree(xms);

 /***************** separator : XmSeparator *****************/
 n = 0;
 XtSetArg(args[n], XmNorientation, XmVERTICAL); n++;
 XtSetArg(args[n], XmNresizable, True); n++;
 widget_array[WI_SEPARATOR2] =
   XmCreateSeparator(widget_array[WI_FILECONVPU], "separator", args, n);

 /***************** separator1 : XmSeparator *****************/
 n = 0;
 XtSetArg(args[n], XmNorientation, XmVERTICAL); n++;
 XtSetArg(args[n], XmNresizable, True); n++;
 widget_array[WI_SEPARATOR3] =
  XmCreateSeparator(widget_array[WI_FILECONVPU], "separator1", args, n);

 /***************** LForm_DEM : XmForm *****************/
 n = 0;
 XtSetArg(args[n], XmNrubberPositioning, False); n++;
 XtSetArg(args[n], XmNresizePolicy, XmRESIZE_ANY); n++;
 XtSetArg(args[n], XmNresizable, True); n++;
 widget_array[WI_LFORM_DEM] =
  XmCreateForm(widget_array[WI_FILECONVPU], "LForm_DEM", args, n);

 /***************** OpenLBL : XmLabel *****************/
 n = 0;
 xms  = XmStringCreate("Select File to Convert",XmSTRING_DEFAULT_CHARSET);
 XtSetArg(args[n], XmNlabelString, xms); n++;
 XtSetArg(args[n], XmNresizable, True); n++;
 widget_array[WI_OPENLBL] =
  XmCreateLabel(widget_array[WI_LFORM_DEM], "OpenLBL", args, n);

 if (xms) XmStringFree(xms);

 /***************** fileSelBoxOpen : XmFileSelectionBox *****************/
 n = 0;
 XtSetArg(args[n], XmNresizePolicy, XmRESIZE_NONE); n++;
 XtSetArg(args[n], XmNwidth, 320); n++;
 xms  = XmStringCreate("Quit",XmSTRING_DEFAULT_CHARSET);
 XtSetArg(args[n], XmNcancelLabelString, xms); n++;
 XtSetArg(args[n], XmNresizable, False); n++;
 widget_array[WI_FILESELBOXOPEN] =
  XmCreateFileSelectionBox(widget_array[WI_LFORM_DEM], "fileSelBoxOpen", args, n);

 if (xms) XmStringFree(xms);

 /***************** MForm_DEM : XmForm *****************/
 n = 0;
 XtSetArg(args[n], XmNrubberPositioning, False); n++;
 XtSetArg(args[n], XmNresizePolicy, XmRESIZE_ANY); n++;
 XtSetArg(args[n], XmNresizable, True); n++;
 widget_array[WI_MFORM_DEM] =
  XmCreateForm(widget_array[WI_FILECONVPU], "MForm_DEM", args, n);

 /***************** StatLBL : XmLabel *****************/
```

```
n = 0;
xms  = XmStringCreate("Statistics",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_STATLBL] =
  XmCreateLabel(widget_array[WI_MFORM_DEM], "StatLBL", args, n);

if (xms) XmStringFree(xms);

/***************** DefaultPB : XmPushButton *****************/
n = 0;
xms  = XmStringCreate("Default",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNwidth, 70); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_DEFAULTPB] =
  XmCreatePushButton(widget_array[WI_MFORM_DEM], "DefaultPB", args, n);

if (xms) XmStringFree(xms);

/***************** layersLBL : XmLabel *****************/
n = 0;
xms  = XmStringCreate("Layers",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_LAYERSLBL] =
  XmCreateLabel(widget_array[WI_MFORM_DEM], "layersLBL", args, n);

if (xms) XmStringFree(xms);

/***************** layersSL : XmScrolledList *****************/
n = 0;
pn = 0;
XtSetArg(args[n], XmNselectionPolicy, XmMULTIPLE_SELECT); n++;
XtSetArg(args[n], XmNwidth, 100); n++;
XtSetArg(args[n], XmNvisibleItemCount, 6); n++;
XtSetArg(pargs[pn], XmNresizable, True); pn++;
widget_array[WI_LAYERSSL] =
  XmCreateScrolledList(widget_array[WI_MFORM_DEM], "layersSL", args, n);
        tmpw    =    get_constraint_widget(widget_array[WI_LAYERSSL],
widget_array[WI_MFORM_DEM]);
 if (tmpw)
  XtSetValues(tmpw, pargs, pn);

 /***************** selectorPB : XmPushButton *****************/
```

```
n = 0;
xms  = XmStringCreate("Clip",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_SELECTORPB] =
  XmCreatePushButton(widget_array[WI_MFORM_DEM], "selectorPB", args, n);

if (xms) XmStringFree(xms);

/***************** h3SEP_DEM : XmSeparator *****************/
n = 0;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_H3SEP_DEM] =
  XmCreateSeparator(widget_array[WI_MFORM_DEM], "h3SEP_DEM", args, n);

/***************** h4SEP_DEM : XmSeparator *****************/
n = 0;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_H4SEP_DEM] =
  XmCreateSeparator(widget_array[WI_MFORM_DEM], "h4SEP_DEM", args, n);

/***************** outLBL_DEM : XmLabel *****************/
n = 0;
XtSetArg(args[n], XmNalignment, XmALIGNMENT_BEGINNING); n++;
xms  = XmStringCreate("Output Options:",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_OUTLBL_DEM] =
  XmCreateLabel(widget_array[WI_MFORM_DEM], "outLBL_DEM", args, n);

if (xms) XmStringFree(xms);

/***************** fileSaveTB : XmToggleButton *****************/
n = 0;
XtSetArg(args[n], XmNshadowThickness, 1); n++;
xms  = XmStringCreate("File Output",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNset, True); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_FILESAVETB] =
  XmCreateToggleButton(widget_array[WI_MFORM_DEM], "fileSaveTB", args, n);

if (xms) XmStringFree(xms);

/***************** binaryTB : XmToggleButton *****************/
```

```
n = 0;
xms  = XmStringCreate("Binary Output",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNshadowThickness, 1); n++;
XtSetArg(args[n], XmNset, True); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_BINARYTB] =
 XmCreateToggleButton(widget_array[WI_MFORM_DEM], "binaryTB", args, n);

if (xms) XmStringFree(xms);

/***************** hSEP_DEM : XmSeparator *****************/
n = 0;
XtSetArg(args[n], XmNseparatorType, XmDOUBLE_LINE); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_HSEP_DEM] =
 XmCreateSeparator(widget_array[WI_MFORM_DEM], "hSEP_DEM", args, n);

/***************** h2SEP_DEM : XmSeparator *****************/
n = 0;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_H2SEP_DEM] =
 XmCreateSeparator(widget_array[WI_MFORM_DEM], "h2SEP_DEM", args, n);

/***************** minxForm_DEM : XmForm *****************/
n = 0;
XtSetArg(args[n], XmNrubberPositioning, False); n++;
XtSetArg(args[n], XmNresizePolicy, XmRESIZE_ANY); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_MINXFORM_DEM] =
 XmCreateForm(widget_array[WI_MFORM_DEM], "minxForm_DEM", args, n);

/***************** minXLBL : XmLabel *****************/
n = 0;
xms  = XmStringCreate("Min X:",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNmarginTop, 5); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_MINXLBL] =
 XmCreateLabel(widget_array[WI_MINXFORM_DEM], "minXLBL", args, n);

if (xms) XmStringFree(xms);

/***************** minXTXT : XmTextField *****************/
n = 0;
XtSetArg(args[n], XmNwidth, 70); n++;
XtSetArg(args[n], XmNresizable, False); n++;
widget_array[WI_MINXTXT] =
 XmCreateTextField(widget_array[WI_MINXFORM_DEM], "minXTXT", args, n);

/***************** minyForm_DEM : XmForm *****************/
n = 0;
XtSetArg(args[n], XmNrubberPositioning, False); n++;
XtSetArg(args[n], XmNresizePolicy, XmRESIZE_ANY); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_MINYFORM_DEM] =
 XmCreateForm(widget_array[WI_MFORM_DEM], "minyForm_DEM", args, n);

/***************** minYLBL : XmLabel *****************/
n = 0;
xms  = XmStringCreate("Min Y:",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_MINYLBL] =
 XmCreateLabel(widget_array[WI_MINYFORM_DEM], "minYLBL", args, n);

if (xms) XmStringFree(xms);

/***************** minYTXT : XmTextField *****************/
n = 0;
XtSetArg(args[n], XmNwidth, 70); n++;
XtSetArg(args[n], XmNresizable, False); n++;
widget_array[WI_MINYTXT] =
 XmCreateTextField(widget_array[WI_MINYFORM_DEM], "minYTXT", args, n);

/***************** maxxForm_DEM : XmForm *****************/
n = 0;
XtSetArg(args[n], XmNrubberPositioning, False); n++;
XtSetArg(args[n], XmNresizePolicy, XmRESIZE_ANY); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_MAXXFORM_DEM] =
 XmCreateForm(widget_array[WI_MFORM_DEM], "maxxForm_DEM", args, n);

/***************** maxXLBL : XmLabel *****************/
n = 0;
xms  = XmStringCreate("Max X:",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_MAXXLBL] =
 XmCreateLabel(widget_array[WI_MAXXFORM_DEM], "maxXLBL", args, n);
```

```
if (xms) XmStringFree(xms);

/***************** maxXTXT : XmTextField *****************/
n = 0;
XtSetArg(args[n], XmNwidth, 70); n++;
XtSetArg(args[n], XmNresizable, False); n++;
widget_array[WI_MAXXTXT] =
 XmCreateTextField(widget_array[WI_MAXXFORM_DEM], "maxXTXT", args, n);

/***************** maxyForm_DEM : XmForm *****************/
n = 0;
XtSetArg(args[n], XmNrubberPositioning, False); n++;
XtSetArg(args[n], XmNresizePolicy, XmRESIZE_ANY); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_MAXYFORM_DEM] =
 XmCreateForm(widget_array[WI_MFORM_DEM], "maxyForm_DEM", args, n);

/***************** maxYLBL : XmLabel *****************/
n = 0;
xms  = XmStringCreate("Max Y:",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_MAXYLBL] =
 XmCreateLabel(widget_array[WI_MAXYFORM_DEM], "maxYLBL", args, n);

if (xms) XmStringFree(xms);

/***************** maxYTXT : XmTextField *****************/
n = 0;
XtSetArg(args[n], XmNwidth, 70); n++;
XtSetArg(args[n], XmNresizable, False); n++;
widget_array[WI_MAXYTXT] =
 XmCreateTextField(widget_array[WI_MAXYFORM_DEM], "maxYTXT", args, n);

/***************** RForm_DEM : XmForm *****************/
n = 0;
XtSetArg(args[n], XmNrubberPositioning, False); n++;
XtSetArg(args[n], XmNresizePolicy, XmRESIZE_ANY); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_RFORM_DEM] =
 XmCreateForm(widget_array[WI_FILECONVPU], "RForm_DEM", args, n);

/***************** ConvLBL1 : XmLabel *****************/
n = 0;
xms  = XmStringCreate("Select File to Save As",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNalignment, XmALIGNMENT_BEGINNING); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_CONVLBL1] =
 XmCreateLabel(widget_array[WI_RFORM_DEM], "ConvLBL1", args, n);

if (xms) XmStringFree(xms);

/***************** fileSelBoxConv : XmFileSelectionBox *****************/
n = 0;
XtSetArg(args[n], XmNresizePolicy, XmRESIZE_NONE); n++;
XtSetArg(args[n], XmNwidth, 320); n++;
xms  = XmStringCreate("*.dx",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNdirMask, xms); n++;
XtSetArg(args[n], XmNresizable, False); n++;
widget_array[WI_FILESELBOXCONV] =
 XmCreateFileSelectionBox(widget_array[WI_RFORM_DEM], "fileSelBoxConv", args, n);

if (xms) XmStringFree(xms);


/***************** separator : XmSeparator *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_LFORM_DEM]); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetValues(widget_array[WI_SEPARATOR2], args, n);

XtManageChild(widget_array[WI_SEPARATOR2]);

/***************** separator1 : XmSeparator *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_MFORM_DEM]); n++;
XtSetValues(widget_array[WI_SEPARATOR3], args, n);

XtManageChild(widget_array[WI_SEPARATOR3]);

/***************** LForm_DEM : XmForm *****************/
n = 0;
```

```
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetValues(widget_array[WI_LFORM_DEM], args, n);


/***************** OpenLBL : XmLabel *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopOffset, 10); n++;
XtSetValues(widget_array[WI_OPENLBL], args, n);

XtManageChild(widget_array[WI_OPENLBL]);

/***************** fileSelBoxOpen : XmFileSelectionBox *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_OPENLBL]); n++;
XtSetValues(widget_array[WI_FILESELBOXOPEN], args, n);

XtManageChild(widget_array[WI_FILESELBOXOPEN]);
XtManageChild(widget_array[WI_LFORM_DEM]);

/***************** MForm_DEM : XmForm *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_SEPARATOR2]); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetValues(widget_array[WI_MFORM_DEM], args, n);


/***************** StatLBL : XmLabel *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopOffset, 10); n++;
XtSetValues(widget_array[WI_STATLBL], args, n);

XtManageChild(widget_array[WI_STATLBL]);

/***************** DefaultPB : XmPushButton *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNrightWidget, widget_array[WI_MAXYFORM_DEM]); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_MAXYFORM_DEM]); n++;
XtSetArg(args[n], XmNtopOffset, 2); n++;
XtSetValues(widget_array[WI_DEFAULTPB], args, n);

XtManageChild(widget_array[WI_DEFAULTPB]);

/***************** layersLBL : XmLabel *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_H2SEP_DEM]); n++;
XtSetArg(args[n], XmNtopOffset, 5); n++;
XtSetValues(widget_array[WI_LAYERSLBL], args, n);

XtManageChild(widget_array[WI_LAYERSLBL]);

/***************** layersSL : XmScrolledList *****************/
pn = 0;
XtSetArg(pargs[pn], XmNleftAttachment, XmATTACH_FORM); pn++;
XtSetArg(pargs[pn], XmNrightAttachment, XmATTACH_FORM); pn++;
XtSetArg(pargs[pn], XmNtopAttachment, XmATTACH_WIDGET); pn++;
XtSetArg(pargs[pn], XmNtopWidget, widget_array[WI_LAYERSLBL]); pn++;
XtSetArg(pargs[pn], XmNbottomAttachment, XmATTACH_NONE); pn++;
XtSetArg(pargs[pn], XmNleftOffset, 15); pn++;
XtSetArg(pargs[pn], XmNrightOffset, 15); pn++;
                tmpw        =        get_constraint_widget(widget_array[WI_LAYERSSL],
widget_array[WI_MFORM_DEM]);
 if (tmpw)
   XtSetValues(tmpw, pargs, pn);
```

```
XtManageChild(widget_array[WI_LAYERSSL]);

/***************** selectorPB : XmPushButton *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNrightWidget, widget_array[WI_DEFAULTPB]); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_DEFAULTPB]); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_DEFAULTPB]); n++;
XtSetArg(args[n], XmNtopOffset, 1); n++;
XtSetValues(widget_array[WI_SELECTORPB], args, n);

XtManageChild(widget_array[WI_SELECTORPB]);

/***************** h3SEP_DEM : XmSeparator *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
        tmpw1       =       get_constraint_widget(widget_array[WI_LAYERSSL],
widget_array[WI_MFORM_DEM]);
XtSetArg(args[n], XmNtopWidget, tmpw1); n++;
XtSetArg(args[n], XmNtopOffset, 10); n++;
XtSetValues(widget_array[WI_H3SEP_DEM], args, n);

XtManageChild(widget_array[WI_H3SEP_DEM]);

/***************** h4SEP_DEM : XmSeparator *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_FILESAVETB]); n++;
XtSetValues(widget_array[WI_H4SEP_DEM], args, n);

XtManageChild(widget_array[WI_H4SEP_DEM]);

/***************** outLBL_DEM : XmLabel *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_H3SEP_DEM]); n++;
XtSetArg(args[n], XmNtopOffset, 10); n++;
XtSetArg(args[n], XmNleftOffset, 5); n++;
XtSetValues(widget_array[WI_OUTLBL_DEM], args, n);

XtManageChild(widget_array[WI_OUTLBL_DEM]);

/***************** fileSaveTB : XmToggleButton *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_BINARYTB]); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNrightWidget, widget_array[WI_BINARYTB]); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_BINARYTB]); n++;
XtSetValues(widget_array[WI_FILESAVETB], args, n);

XtManageChild(widget_array[WI_FILESAVETB]);

/***************** binaryTB : XmToggleButton *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_OUTLBL_DEM]); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNleftOffset, 15); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_OUTLBL_DEM]); n++;
XtSetValues(widget_array[WI_BINARYTB], args, n);

XtManageChild(widget_array[WI_BINARYTB]);

/***************** hSEP_DEM : XmSeparator *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_STATLBL]); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetValues(widget_array[WI_HSEP_DEM], args, n);

XtManageChild(widget_array[WI_HSEP_DEM]);
```

```
/***************** h2SEP_DEM : XmSeparator *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_SELECTORPB]); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopOffset, 5); n++;
XtSetValues(widget_array[WI_H2SEP_DEM], args, n);

XtManageChild(widget_array[WI_H2SEP_DEM]);

/***************** minxForm_DEM : XmForm *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_HSEP_DEM]); n++;
XtSetArg(args[n], XmNtopOffset, 15); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNleftOffset, 5); n++;
XtSetArg(args[n], XmNrightOffset, 5); n++;
XtSetValues(widget_array[WI_MINXFORM_DEM], args, n);


/***************** minXLBL : XmLabel *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetValues(widget_array[WI_MINXLBL], args, n);

XtManageChild(widget_array[WI_MINXLBL]);

/***************** minXTXT : XmTextField *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_MINXLBL]); n++;
XtSetValues(widget_array[WI_MINXTXT], args, n);

XtManageChild(widget_array[WI_MINXTXT]);
XtManageChild(widget_array[WI_MINXFORM_DEM]);

/***************** minyForm_DEM : XmForm *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_MINXFORM_DEM]); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_MINXFORM_DEM]); n++;
XtSetArg(args[n], XmNrightWidget, widget_array[WI_MINXFORM_DEM]); n++;
XtSetValues(widget_array[WI_MINYFORM_DEM], args, n);


/***************** minYLBL : XmLabel *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetValues(widget_array[WI_MINYLBL], args, n);

XtManageChild(widget_array[WI_MINYLBL]);

/***************** minYTXT : XmTextField *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_MINYLBL]); n++;
XtSetValues(widget_array[WI_MINYTXT], args, n);

XtManageChild(widget_array[WI_MINYTXT]);
XtManageChild(widget_array[WI_MINYFORM_DEM]);

/***************** maxxForm_DEM : XmForm *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_MINYFORM_DEM]); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_MINYFORM_DEM]); n++;
XtSetArg(args[n], XmNrightWidget, widget_array[WI_MINYFORM_DEM]); n++;
```

```
XtSetValues(widget_array[WI_MAXXFORM_DEM], args, n);


/***************** maxXLBL : XmLabel *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetValues(widget_array[WI_MAXXLBL], args, n);

XtManageChild(widget_array[WI_MAXXLBL]);

/***************** maxXTXT : XmTextField *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_MAXXLBL]); n++;
XtSetValues(widget_array[WI_MAXXTXT], args, n);

XtManageChild(widget_array[WI_MAXXTXT]);
XtManageChild(widget_array[WI_MAXXFORM_DEM]);

/***************** maxyForm_DEM : XmForm *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_MAXXFORM_DEM]); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_MAXXFORM_DEM]); n++;
XtSetArg(args[n], XmNrightWidget, widget_array[WI_MAXXFORM_DEM]); n++;
XtSetValues(widget_array[WI_MAXYFORM_DEM], args, n);


/***************** maxYLBL : XmLabel *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetValues(widget_array[WI_MAXYLBL], args, n);

XtManageChild(widget_array[WI_MAXYLBL]);

/***************** maxYTXT : XmTextField *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_MAXYLBL]); n++;
XtSetValues(widget_array[WI_MAXYTXT], args, n);

XtManageChild(widget_array[WI_MAXYTXT]);
XtManageChild(widget_array[WI_MAXYFORM_DEM]);
XtManageChild(widget_array[WI_MFORM_DEM]);

/***************** RForm_DEM : XmForm *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_SEPARATOR3]); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetValues(widget_array[WI_RFORM_DEM], args, n);


/***************** ConvLBL1 : XmLabel *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopOffset, 10); n++;
XtSetValues(widget_array[WI_CONVLBL1], args, n);

XtManageChild(widget_array[WI_CONVLBL1]);

/***************** fileSelBoxConv : XmFileSelectionBox *****************/
n = 0;
        tmpw1     =     XmSelectionBoxGetChild(widget_array[WI_FILESELBOXCONV],
XmDIALOG_OK_BUTTON);
XtUnmanageChild(tmpw1);
        tmpw1     =     XmSelectionBoxGetChild(widget_array[WI_FILESELBOXCONV],
XmDIALOG_CANCEL_BUTTON);
XtUnmanageChild(tmpw1);
        tmpw1     =     XmSelectionBoxGetChild(widget_array[WI_FILESELBOXCONV],
```

```
XmDIALOG_HELP_BUTTON);
 XtUnmanageChild(tmpw1);
 XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
 XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
 XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
 XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
 XtSetArg(args[n], XmNtopWidget, widget_array[WI_CONVLBL1]); n++;
 XtSetValues(widget_array[WI_FILESELBOXCONV], args, n);

 XtManageChild(widget_array[WI_FILESELBOXCONV]);
 XtManageChild(widget_array[WI_RFORM_DEM]);

 /*
 */
 /*
  *  Return the first created widget.
  */
 return widget_array[WI_FCONV];
}


/************************************************************
 * tu_warningDialog_widget:
 ************************************************************/
Widget tu_warningDialog_widget(char    * name,
                    Widget    parent,
                    Widget widget_array[])
{
 Arg args[15];
 Widget tmpw1;
 XmString xms = (XmString) NULL;
 XmString xms1 = (XmString) NULL;
 XmString xms2 = (XmString) NULL;
 int n;

 /***************** object of type : XmWarningDialog ****************/
 n = 0;
 XtSetArg(args[n], XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL); n++;
 xms  = XmStringCreate("Warning!",XmSTRING_DEFAULT_CHARSET);
 XtSetArg(args[n], XmNdialogTitle, xms); n++;
 xms1  = XmStringCreate("Output File Exists!",XmSTRING_DEFAULT_CHARSET);
 XtSetArg(args[n], XmNmessageString, xms1); n++;
 xms2  = XmStringCreate("Overwrite?",XmSTRING_DEFAULT_CHARSET);
 XtSetArg(args[n], XmNokLabelString, xms2); n++;
 widget_array[WI_WARNINGDIALOG] =
  XmCreateWarningDialog(parent, name, args, n);

 if (xms) XmStringFree(xms);
 if (xms1) XmStringFree(xms1);
 if (xms2) XmStringFree(xms2);

          tmpw1      =      XmMessageBoxGetChild(widget_array[WI_WARNINGDIALOG],
XmDIALOG_HELP_BUTTON);
 XtUnmanageChild(tmpw1);

 /*
 */
 /*
  *  Return the first created widget.
  */
 return widget_array[WI_WARNINGDIALOG];
}


/************************************************************
*/
```

```
// SHAPE Converter
//////////////////////////////////////////////////////////////////////
//  File:       shp2dxW.h
//  Author:     Kirk A. Moeller
//  Description:
//              Header file for shape converter dialog
//              creation function.
//////////////////////////////////////////////////////////////////////

#ifndef SHP2DXW_H
#define SHP2DXW_H

#include "XHeaders.h"

#define WI_FCONV_SHAPE 0
#define WI_FILECONVPU_SHAPE 1
#define WI_RFORM_SHAPE 2
#define WI_FILESELBOXCONV_SHAPE 3
#define WI_CONVLBL_SHAPE 4
#define WI_LFORM_SHAPE 5
#define WI_FILESELBOXOPEN_SHAPE 6
#define WI_OPNLBL_SHAPE 7
#define WI_MFORM_SHAPE 8
#define WI_STATLBL_SHAPE 9
#define WI_ATTRIBLBL_SHAPE 10
#define WI_ATTRIBLST_SHAPE 11
#define WI_TOPSEP_SHAPE 12
#define WI_ORDERDLGPB_SHAPE 13
#define WI_LINESRB_SHAPE 14
#define WI_TB_SHAPE 15
#define WI_TB1_SHAPE 16
#define WI_LINESRBLBL_SHAPE 17
#define WI_MID2SEP_SHAPE 18
#define WI_FILETB_SHAPE 19
#define WI_BINARYTB_SHAPE 20
#define WI_OUTLBL_SHAPE 21
#define WI_BOTSEP_SHAPE 22
#define WI_CREATESERIESTB_SHAPE 23
#define WI_MIDSEP_SHAPE 24
#define WI_LSEP_SHAPE 25
#define WI_RSEP_SHAPE 26


Widget tu_FConv_Shape_widget(char    * name,
                             Widget    parent,
                             Widget ** warr_ret);

#endif
```

```
// SHAPE Converter
////////////////////////////////////////////////////////////////////////
//  File:           shp2dxW.C
//  Author:         Kirk A. Moeller
//  Description:
//                  Creates the dialog box for the shape
//                  converter.
//
// File is generated by TeleUSE
// Version : TeleUSE v2.1.5 / AIX 3.2
////////////////////////////////////////////////////////////////////////

#include "shp2dxW.h"



/*
 * get_constraint_widget:
 ****************************************************************/
static Widget get_constraint_widget(Widget child, Widget parent)
{
  Widget w;

  w = child;
  while (XtParent(w) != parent)
    w = XtParent(w);
  return (w);
}
```

```
/*****************************************************************
 *
 *  Main C code for presentation component
 *
 *****************************************************************/

/*************************************************************
 * tu_FConv_Shape_widget:
 *************************************************************/
Widget tu_FConv_Shape_widget(char    * name,
                    Widget    parent,
                    Widget ** warr_ret)
{
  Arg args[22];
  Arg pargs[22];
  Widget tmpw;
  Widget tmpw1;
  Widget widget_array[27];
  XmString xms = (XmString) NULL;
  XmString xms1 = (XmString) NULL;
  int n;
  int pn;

  /***************** object of type : XmDialogShell *****************/
  n = 0;
  XtSetArg(args[n], XtNallowShellResize, True); n++;
  XtSetArg(args[n], XmNdeleteResponse, XmDO_NOTHING); n++;
  widget_array[WI_FCONV_SHAPE] =
   XmCreateDialogShell(parent, name, args, n);

  /***************** FileConvPu_Shape : XmForm *****************/
  n = 0;
  XtSetArg(args[n], XmNnoResize, True); n++;
  XtSetArg(args[n], XmNautoUnmanage, False); n++;
  XtSetArg(args[n], XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL); n++;
  widget_array[WI_FILECONVPU_SHAPE] =
    XmCreateForm(widget_array[WI_FCONV_SHAPE], "FileConvPu_Shape", args, n);

  /***************** RForm_Shape : XmForm *****************/
  n = 0;
  XtSetArg(args[n], XmNautoUnmanage, False); n++;
  XtSetArg(args[n], XmNrubberPositioning, False); n++;
  XtSetArg(args[n], XmNresizePolicy, XmRESIZE_ANY); n++;
  XtSetArg(args[n], XmNresizable, True); n++;
  widget_array[WI_RFORM_SHAPE] =
    XmCreateForm(widget_array[WI_FILECONVPU_SHAPE], "RForm_Shape", args, n);

  /***************** fileSelBoxConv_Shape : XmFileSelectionBox *****************/
  n = 0;
  XtSetArg(args[n], XmNresizePolicy, XmRESIZE_NONE); n++;
  XtSetArg(args[n], XmNwidth, 320); n++;
  xms  = XmStringCreate("*.dx",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(args[n], XmNdirMask, xms); n++;
  XtSetArg(args[n], XmNresizable, False); n++;
  widget_array[WI_FILESELBOXCONV_SHAPE] =
    XmCreateFileSelectionBox(widget_array[WI_RFORM_SHAPE], "fileSelBoxConv_Shape", args,
n);

  if (xms) XmStringFree(xms);

  /***************** convLBL_Shape : XmLabel *****************/
  n = 0;
  xms  = XmStringCreate("Select File to Save As",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(args[n], XmNlabelString, xms); n++;
  XtSetArg(args[n], XmNalignment, XmALIGNMENT_BEGINNING); n++;
  XtSetArg(args[n], XmNresizable, True); n++;
  widget_array[WI_CONVLBL_SHAPE] =
    XmCreateLabel(widget_array[WI_RFORM_SHAPE], "convLBL_Shape", args, n);

  if (xms) XmStringFree(xms);

  /***************** LForm_Shape : XmForm *****************/
  n = 0;
  XtSetArg(args[n], XmNrubberPositioning, False); n++;
  XtSetArg(args[n], XmNresizePolicy, XmRESIZE_ANY); n++;
  XtSetArg(args[n], XmNresizable, True); n++;
  widget_array[WI_LFORM_SHAPE] =
    XmCreateForm(widget_array[WI_FILECONVPU_SHAPE], "LForm_Shape", args, n);

  /***************** fileSelBoxOpen_Shape : XmFileSelectionBox *****************/
  n = 0;
  XtSetArg(args[n], XmNresizePolicy, XmRESIZE_NONE); n++;
  XtSetArg(args[n], XmNwidth, 320); n++;
  xms  = XmStringCreate("*.shp",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(args[n], XmNdirMask, xms); n++;
  xms1  = XmStringCreate("Quit",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(args[n], XmNcancelLabelString, xms1); n++;
  XtSetArg(args[n], XmNresizable, False); n++;
```

```
 widget_array[WI_FILESELBOXOPEN_SHAPE] =
  XmCreateFileSelectionBox(widget_array[WI_LFORM_SHAPE], "fileSelBoxOpen_Shape", args,
n);

 if (xms) XmStringFree(xms);
 if (xms1) XmStringFree(xms1);

 /***************** OpnLBL_Shape : XmLabel *****************/
 n = 0;
 xms  = XmStringCreate("Select File to Convert",XmSTRING_DEFAULT_CHARSET);
 XtSetArg(args[n], XmNlabelString, xms); n++;
 XtSetArg(args[n], XmNalignment, XmALIGNMENT_BEGINNING); n++;
 XtSetArg(args[n], XmNresizable, True); n++;
 widget_array[WI_OPNLBL_SHAPE] =
  XmCreateLabel(widget_array[WI_LFORM_SHAPE], "OpnLBL_Shape", args, n);

 if (xms) XmStringFree(xms);

 /***************** MForm_Shape : XmForm *****************/
 n = 0;
 XtSetArg(args[n], XmNrubberPositioning, False); n++;
 XtSetArg(args[n], XmNresizePolicy, XmRESIZE_ANY); n++;
 XtSetArg(args[n], XmNresizable, True); n++;
 widget_array[WI_MFORM_SHAPE] =
  XmCreateForm(widget_array[WI_FILECONVPU_SHAPE], "MForm_Shape", args, n);

 /***************** StatLBL_Shape : XmLabel *****************/
 n = 0;
 xms  = XmStringCreate("Options",XmSTRING_DEFAULT_CHARSET);
 XtSetArg(args[n], XmNlabelString, xms); n++;
 XtSetArg(args[n], XmNresizable, True); n++;
 widget_array[WI_STATLBL_SHAPE] =
  XmCreateLabel(widget_array[WI_MFORM_SHAPE], "StatLBL_Shape", args, n);

 if (xms) XmStringFree(xms);

 /***************** AttribLBL_Shape : XmLabel *****************/
 n = 0;
 xms  = XmStringCreate("Attributes:",XmSTRING_DEFAULT_CHARSET);
 XtSetArg(args[n], XmNlabelString, xms); n++;
 XtSetArg(args[n], XmNalignment, XmALIGNMENT_BEGINNING); n++;
 XtSetArg(args[n], XmNresizable, True); n++;
 widget_array[WI_ATTRIBLBL_SHAPE] =
  XmCreateLabel(widget_array[WI_MFORM_SHAPE], "AttribLBL_Shape", args, n);

 if (xms) XmStringFree(xms);

 /***************** AttribLST_Shape : XmScrolledList *****************/
 n = 0;
 pn = 0;
 XtSetArg(args[n], XmNselectionPolicy, XmMULTIPLE_SELECT); n++;
 XtSetArg(args[n], XmNvisibleItemCount, 6); n++;
 XtSetArg(args[n], XmNwidth, 150); n++;
 XtSetArg(pargs[pn], XmNresizable, True); pn++;
 widget_array[WI_ATTRIBLST_SHAPE] =
  XmCreateScrolledList(widget_array[WI_MFORM_SHAPE], "AttribLST_Shape", args, n);
        tmpw      =      get_constraint_widget(widget_array[WI_ATTRIBLST_SHAPE],
widget_array[WI_MFORM_SHAPE]);
 if (tmpw)
  XtSetValues(tmpw, pargs, pn);

 /***************** topSEP_Shape : XmSeparator *****************/
 n = 0;
 XtSetArg(args[n], XmNseparatorType, XmDOUBLE_LINE); n++;
 XtSetArg(args[n], XmNresizable, True); n++;
 widget_array[WI_TOPSEP_SHAPE] =
  XmCreateSeparator(widget_array[WI_MFORM_SHAPE], "topSEP_Shape", args, n);

 /***************** OrderdlgPB_Shape : XmPushButton *****************/
 n = 0;
 xms  = XmStringCreate("Attribute Dialog ...",XmSTRING_DEFAULT_CHARSET);
 XtSetArg(args[n], XmNlabelString, xms); n++;
 XtSetArg(args[n], XmNresizable, True); n++;
 widget_array[WI_ORDERDLGPB_SHAPE] =
  XmCreatePushButton(widget_array[WI_MFORM_SHAPE], "OrderdlgPB_Shape", args, n);

 if (xms) XmStringFree(xms);

 /***************** linesRB_Shape : XmRadioBox *****************/
 n = 0;
 XtSetArg(args[n], XmNresizable, True); n++;
 widget_array[WI_LINESRB_SHAPE] =
  XmCreateRadioBox(widget_array[WI_MFORM_SHAPE], "linesRB_Shape", args, n);

 /***************** TB_Shape : XmToggleButton *****************/
 n = 0;
 xms  = XmStringCreate("Lines",XmSTRING_DEFAULT_CHARSET);
 XtSetArg(args[n], XmNlabelString, xms); n++;
 XtSetArg(args[n], XmNset, True); n++;
 widget_array[WI_TB_SHAPE] =
```

```
 XmCreateToggleButton(widget_array[WI_LINESRB_SHAPE], "TB_Shape", args, n);

if (xms) XmStringFree(xms);

/***************** TB1_Shape : XmToggleButton *****************/
n = 0;
xms  = XmStringCreate("Polylines",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNrecomputeSize, False); n++;
XtSetArg(args[n], XmNheight, 20); n++;
widget_array[WI_TB1_SHAPE] =
 XmCreateToggleButton(widget_array[WI_LINESRB_SHAPE], "TB1_Shape", args, n);

if (xms) XmStringFree(xms);

/***************** linesRBlbl_Shape : XmLabel *****************/
n = 0;
xms  = XmStringCreate("DX structure:",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNalignment, XmALIGNMENT_BEGINNING); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_LINESRBLBL_SHAPE] =
 XmCreateLabel(widget_array[WI_MFORM_SHAPE], "linesRBlbl_Shape", args, n);

if (xms) XmStringFree(xms);

/***************** mid2SEP_Shape : XmSeparator *****************/
n = 0;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_MID2SEP_SHAPE] =
 XmCreateSeparator(widget_array[WI_MFORM_SHAPE], "mid2SEP_Shape", args, n);

/***************** fileTB_Shape : XmToggleButton *****************/
n = 0;
xms  = XmStringCreate("File Output",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNset, True); n++;
XtSetArg(args[n], XmNalignment, XmALIGNMENT_BEGINNING); n++;
XtSetArg(args[n], XmNshadowThickness, 1); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_FILETB_SHAPE] =
 XmCreateToggleButton(widget_array[WI_MFORM_SHAPE], "fileTB_Shape", args, n);

if (xms) XmStringFree(xms);

/***************** binaryTB_Shape : XmToggleButton *****************/
n = 0;
xms  = XmStringCreate("Binary Output",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNset, True); n++;
XtSetArg(args[n], XmNalignment, XmALIGNMENT_BEGINNING); n++;
XtSetArg(args[n], XmNshadowThickness, 1); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_BINARYTB_SHAPE] =
 XmCreateToggleButton(widget_array[WI_MFORM_SHAPE], "binaryTB_Shape", args, n);

if (xms) XmStringFree(xms);

/***************** outlbl_Shape : XmLabel *****************/
n = 0;
xms  = XmStringCreate("Output Options:",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNalignment, XmALIGNMENT_BEGINNING); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_OUTLBL_SHAPE] =
 XmCreateLabel(widget_array[WI_MFORM_SHAPE], "outlbl_Shape", args, n);

if (xms) XmStringFree(xms);

/***************** botSEP_Shape : XmSeparator *****************/
n = 0;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_BOTSEP_SHAPE] =
 XmCreateSeparator(widget_array[WI_MFORM_SHAPE], "botSEP_Shape", args, n);

/***************** CreateSeriesTB_Shape : XmToggleButton *****************/
n = 0;
XtSetArg(args[n], XmNshadowThickness, 1); n++;
xms  = XmStringCreate("Create Time Series",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_CREATESERIESTB_SHAPE] =
 XmCreateToggleButton(widget_array[WI_MFORM_SHAPE], "CreateSeriesTB_Shape", args,
n);

if (xms) XmStringFree(xms);

/***************** midSEP_Shape : XmSeparator *****************/
n = 0;
XtSetArg(args[n], XmNresizable, True); n++;
```

```
widget_array[WI_MIDSEP_SHAPE] =
  XmCreateSeparator(widget_array[WI_MFORM_SHAPE], "midSEP_Shape", args, n);

/***************** lSEP_Shape : XmSeparator *****************/
n = 0;
XtSetArg(args[n], XmNorientation, XmVERTICAL); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_LSEP_SHAPE] =
  XmCreateSeparator(widget_array[WI_FILECONVPU_SHAPE], "lSEP_Shape", args, n);

/***************** rSEP_Shape : XmSeparator *****************/
n = 0;
XtSetArg(args[n], XmNorientation, XmVERTICAL); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_RSEP_SHAPE] =
  XmCreateSeparator(widget_array[WI_FILECONVPU_SHAPE], "rSEP_Shape", args, n);


/***************** RForm_Shape : XmForm *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_RSEP_SHAPE]); n++;
XtSetValues(widget_array[WI_RFORM_SHAPE], args, n);


/***************** fileSelBoxConv_Shape : XmFileSelectionBox *****************/
n = 0;
    tmpw1   =   XmSelectionBoxGetChild(widget_array[WI_FILESELBOXCONV_SHAPE],
XmDIALOG_OK_BUTTON);
XtUnmanageChild(tmpw1);
    tmpw1   =   XmSelectionBoxGetChild(widget_array[WI_FILESELBOXCONV_SHAPE],
XmDIALOG_CANCEL_BUTTON);
XtUnmanageChild(tmpw1);
    tmpw1   =   XmSelectionBoxGetChild(widget_array[WI_FILESELBOXCONV_SHAPE],
XmDIALOG_HELP_BUTTON);
XtUnmanageChild(tmpw1);
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_CONVLBL_SHAPE]); n++;
XtSetValues(widget_array[WI_FILESELBOXCONV_SHAPE], args, n);
XtManageChild(widget_array[WI_FILESELBOXCONV_SHAPE]);

/***************** convLBL_Shape : XmLabel *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopOffset, 10); n++;
XtSetValues(widget_array[WI_CONVLBL_SHAPE], args, n);

XtManageChild(widget_array[WI_CONVLBL_SHAPE]);
XtManageChild(widget_array[WI_RFORM_SHAPE]);

/***************** LForm_Shape : XmForm *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetValues(widget_array[WI_LFORM_SHAPE], args, n);


/***************** fileSelBoxOpen_Shape : XmFileSelectionBox *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_OPNLBL_SHAPE]); n++;
XtSetValues(widget_array[WI_FILESELBOXOPEN_SHAPE], args, n);

XtManageChild(widget_array[WI_FILESELBOXOPEN_SHAPE]);

/***************** OpnLBL_Shape : XmLabel *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopOffset, 10); n++;
XtSetValues(widget_array[WI_OPNLBL_SHAPE], args, n);

XtManageChild(widget_array[WI_OPNLBL_SHAPE]);
XtManageChild(widget_array[WI_LFORM_SHAPE]);
```

```
/***************** MForm_Shape : XmForm *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_LSEP_SHAPE]); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetValues(widget_array[WI_MFORM_SHAPE], args, n);


/***************** StatLBL_Shape : XmLabel *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopOffset, 10); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetValues(widget_array[WI_STATLBL_SHAPE], args, n);

XtManageChild(widget_array[WI_STATLBL_SHAPE]);

/***************** AttribLBL_Shape : XmLabel *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_TOPSEP_SHAPE]); n++;
XtSetArg(args[n], XmNtopOffset, 20); n++;
XtSetValues(widget_array[WI_ATTRIBLBL_SHAPE], args, n);

XtManageChild(widget_array[WI_ATTRIBLBL_SHAPE]);

/***************** AttribLST_Shape : XmScrolledList *****************/
pn = 0;
XtSetArg(pargs[pn], XmNleftAttachment, XmATTACH_FORM); pn++;
XtSetArg(pargs[pn], XmNrightAttachment, XmATTACH_FORM); pn++;
XtSetArg(pargs[pn], XmNtopAttachment, XmATTACH_WIDGET); pn++;
XtSetArg(pargs[pn], XmNtopWidget, widget_array[WI_ATTRIBLBL_SHAPE]); pn++;
XtSetArg(pargs[pn], XmNbottomAttachment, XmATTACH_NONE); pn++;
XtSetArg(pargs[pn], XmNtopOffset, 3); pn++;
XtSetArg(pargs[pn], XmNleftOffset, 5); pn++;
XtSetArg(pargs[pn], XmNrightOffset, 5); pn++;
        tmpw    =    get_constraint_widget(widget_array[WI_ATTRIBLST_SHAPE],
widget_array[WI_MFORM_SHAPE]);
if (tmpw)
  XtSetValues(tmpw, pargs, pn);

XtManageChild(widget_array[WI_ATTRIBLST_SHAPE]);

/***************** topSEP_Shape : XmSeparator *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_STATLBL_SHAPE]); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetValues(widget_array[WI_TOPSEP_SHAPE], args, n);

XtManageChild(widget_array[WI_TOPSEP_SHAPE]);

/***************** OrderdlgPB_Shape : XmPushButton *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_CREATESERIESTB_SHAPE]); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_CREATESERIESTB_SHAPE]); n++;
XtSetArg(args[n], XmNrightWidget, widget_array[WI_CREATESERIESTB_SHAPE]); n++;
XtSetValues(widget_array[WI_ORDERDLGPB_SHAPE], args, n);

XtManageChild(widget_array[WI_ORDERDLGPB_SHAPE]);

/***************** linesRB_Shape : XmRadioBox *****************/
n = 0;
XtSetSensitive(widget_array[WI_LINESRB_SHAPE], False);
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNleftOffset, 25); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_LINESRBLBL_SHAPE]); n++;
XtSetValues(widget_array[WI_LINESRB_SHAPE], args, n);

XtManageChild(widget_array[WI_TB_SHAPE]);
XtManageChild(widget_array[WI_TB1_SHAPE]);
XtManageChild(widget_array[WI_LINESRB_SHAPE]);

/***************** linesRBlbl_Shape : XmLabel *****************/
```

```
n = 0;
XtSetSensitive(widget_array[WI_LINESRBLBL_SHAPE], False);
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_MIDSEP_SHAPE]); n++;
XtSetValues(widget_array[WI_LINESRBLBL_SHAPE], args, n);

XtManageChild(widget_array[WI_LINESRBLBL_SHAPE]);

/***************** mid2SEP_Shape : XmSeparator *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_LINESRB_SHAPE]); n++;
XtSetValues(widget_array[WI_MID2SEP_SHAPE], args, n);

XtManageChild(widget_array[WI_MID2SEP_SHAPE]);

/***************** fileTB_Shape : XmToggleButton *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_BINARYTB_SHAPE]); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_BINARYTB_SHAPE]); n++;
XtSetArg(args[n], XmNrightWidget, widget_array[WI_BINARYTB_SHAPE]); n++;
XtSetValues(widget_array[WI_FILETB_SHAPE], args, n);

XtManageChild(widget_array[WI_FILETB_SHAPE]);

/***************** binaryTB_Shape : XmToggleButton *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_OUTLBL_SHAPE]); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNleftOffset, 15); n++;
XtSetValues(widget_array[WI_BINARYTB_SHAPE], args, n);

XtManageChild(widget_array[WI_BINARYTB_SHAPE]);

/***************** outlbl_Shape : XmLabel *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_MID2SEP_SHAPE]); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopOffset, 10); n++;
XtSetValues(widget_array[WI_OUTLBL_SHAPE], args, n);

XtManageChild(widget_array[WI_OUTLBL_SHAPE]);

/***************** botSEP_Shape : XmSeparator *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_FILETB_SHAPE]); n++;
XtSetValues(widget_array[WI_BOTSEP_SHAPE], args, n);

XtManageChild(widget_array[WI_BOTSEP_SHAPE]);

/***************** CreateSeriesTB_Shape : XmToggleButton *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
        tmpw1       =       get_constraint_widget(widget_array[WI_ATTRIBLST_SHAPE],
widget_array[WI_MFORM_SHAPE]);
XtSetArg(args[n], XmNtopWidget, tmpw1); n++;
XtSetArg(args[n], XmNtopOffset, 5); n++;
XtSetValues(widget_array[WI_CREATESERIESTB_SHAPE], args, n);

XtManageChild(widget_array[WI_CREATESERIESTB_SHAPE]);

/***************** midSEP_Shape : XmSeparator *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
```

```
XtSetArg(args[n], XmNtopWidget, widget_array[WI_ORDERDLGPB_SHAPE]); n++;
XtSetArg(args[n], XmNtopOffset, 10); n++;
XtSetValues(widget_array[WI_MIDSEP_SHAPE], args, n);

XtManageChild(widget_array[WI_MIDSEP_SHAPE]);
XtManageChild(widget_array[WI_MFORM_SHAPE]);

/***************** lSEP_Shape : XmSeparator *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_LFORM_SHAPE]); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetValues(widget_array[WI_LSEP_SHAPE], args, n);

XtManageChild(widget_array[WI_LSEP_SHAPE]);

/***************** rSEP_Shape : XmSeparator *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_MFORM_SHAPE]); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetValues(widget_array[WI_RSEP_SHAPE], args, n);

XtManageChild(widget_array[WI_RSEP_SHAPE]);

/*
 *  Allocate memory for the widget array to return
 */
  *warr_ret = (Widget *) malloc(sizeof(Widget)*27);
  (void) memcpy((char *)*warr_ret,
            (char *)widget_array,
        sizeof(Widget)*27);

/*
 *  Return the first created widget.
 */
 return widget_array[WI_FCONV_SHAPE];
}

/*************************************************************
*/
```

```
// ********************************************************************
// FILE:        seriesW.h
// AUTHOR:      Kirk A. Moeller
// DESCRIPTION:
//              prototype series order dialog creation function
// ********************************************************************
#ifndef SERIES_H
#define SERIES_H

#include "XHeaders.h"

#define WI_SERIESDLG 0
#define WI_ATTRIBLST 1
#define WI_ATTRIBLSTLBL 2
#define WI_INSERTRBLBL 3
#define WI_INSERTRB 4
#define WI_BEGINTB 5
#define WI_ENDTB 6
#define WI_AFTERTB 7
#define WI_SERIESORDERLST 8
#define WI_ATTRIBLBL 9
#define WI_TIMEFORM_SERIES 10
#define WI_TIMETAGPB 11
#define WI_TIMETAGTXT 12
#define WI_TIMETAGLBL 13
#define WI_ARROWFORM_SERIES 14
#define WI_ARROWSLBL2 15
#define WI_RIGHTARROW 16
#define WI_LEFTARROW 17
#define WI_ARROWSLBL 18
#define WI_HSEP_SERIES 19
#define WI_CONTROLS 20
#define WI_OKPB_SERIES 21
#define WI_CANCELPB_SERIES 22
#define WI_HELPPB_SERIES 23
#define WI_TIMEDEFPB 24
#define WI_H3SEP_SERIES 25
#define WI_H2SEP_SERIES 26

Widget tu_seriesdlg_widget(char    * name,
                    Widget    parent, Widget ** warr_ret);
#endif
```

```
// ERDAS/IMAGINE Converter
////////////////////////////////////////////////////////////////////////
//  File:           img2dxW.h
//  Author:         Kirk A. Moeller
//  Description:
//                  Header file for imagine dialog creation
//                  function.
////////////////////////////////////////////////////////////////////////

#ifndef IMG2DXW_H
#define IMG2DXW_H

#include "XHeaders.h"

#define WI_IMGDLGSHELL 0
#define WI_IMGFORM 1
#define WI_LFORM_IMG 2
#define WI_OPENLBL_IMG 3
#define WI_OPENBOX_IMG 4
#define WI_VSEP_IMG 5
#define WI_MFORM_IMG 6
#define WI_STATLBL_IMG 7
#define WI_HSEP_IMG 8
#define WI_MINXFORM_IMG 9
#define WI_MINXLBL_IMG 10
#define WI_MINXTXT_IMG 11
#define WI_MINYFORM_IMG 12
#define WI_MINYLBL_IMG 13
#define WI_MINYTXT_IMG 14
#define WI_MAXXFORM_IMG 15
#define WI_MAXXLBL_IMG 16
#define WI_MAXXTXT_IMG 17
#define WI_MAXYFORM_IMG 18
#define WI_MAXYLBL_IMG 19
#define WI_MAXYTXT_IMG 20
#define WI_DEFAULTPB_IMG 21
#define WI_CLIPPB_IMG 22
#define WI_H2SEP_IMG 23
#define WI_LAYERSLBL_IMG 24
#define WI_LAYERSSL_IMG 25
#define WI_H3SEP_IMG 26
#define WI_OUTLBL_IMG 27
#define WI_BINARYTB_IMG 28
#define WI_FILETB_IMG 29
#define WI_H4SEP_IMG 30
#define WI_V2SEP_IMG 31
#define WI_RFORM_IMG 32
#define WI_SAVELBL_IMG 33
#define WI_SAVEBOX_IMG 34


Widget tu_IMGDlgShell_widget(char    * name,
                             Widget    parent,
                             Widget ** warr_ret);


#endif
```

```
// ERDAS/IMAGINE Converter
////////////////////////////////////////////////////////////////////////
//   File:          img2dxW.C
//   Author:        Kirk A. Moeller and TeleUSE
//   Description:
//                  Creates the dialog for the imagine converter
//                  Generated by teleuse from .pcd file
////////////////////////////////////////////////////////////////////////

#include "img2dxW.h"



/*
 * get_constraint_widget:
 ************************************************************/
static Widget get_constraint_widget(Widget child, Widget parent)
{
  Widget w;

  w = child;
  while (XtParent(w) != parent)
    w = XtParent(w);
  return (w);
}
```

```
/*****************************************************************
 *
 *  Main C code for presentation component
 *
 *****************************************************************/

/************************************************************
 * tu_IMGDlgShell_widget:
 ************************************************************/
Widget tu_IMGDlgShell_widget(char    * name,
                    Widget    parent,
                    Widget ** warr_ret)
{
  Arg args[23];
  Arg pargs[23];
  Widget tmpw;
  Widget tmpw1;
  Widget widget_array[35];
  XmString xms = (XmString) NULL;
  XmString xms1 = (XmString) NULL;
  int n;
  int pn;

  /***************** object of type : XmDialogShell *****************/
  n = 0;
  XtSetArg(args[n], XtNallowShellResize, True); n++;
  XtSetArg(args[n], XmNdeleteResponse, XmDO_NOTHING); n++;
  widget_array[WI_IMGDLGSHELL] =
   XmCreateDialogShell(parent, name, args, n);

  /***************** IMGForm : XmForm *****************/
  n = 0;
  XtSetArg(args[n], XmNrubberPositioning, False); n++;
  XtSetArg(args[n], XmNresizePolicy, XmRESIZE_ANY); n++;
  XtSetArg(args[n], XmNautoUnmanage, False); n++;
  XtSetArg(args[n], XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL); n++;
  XtSetArg(args[n], XmNnoResize, True); n++;
  widget_array[WI_IMGFORM] =
    XmCreateForm(widget_array[WI_IMGDLGSHELL], "IMGForm", args, n);

  /***************** LForm_IMG : XmForm *****************/
  n = 0;
  XtSetArg(args[n], XmNrubberPositioning, False); n++;
  XtSetArg(args[n], XmNresizePolicy, XmRESIZE_ANY); n++;
  XtSetArg(args[n], XmNresizable, True); n++;
  widget_array[WI_LFORM_IMG] =
   XmCreateForm(widget_array[WI_IMGFORM], "LForm_IMG", args, n);

  /***************** openLBL_IMG : XmLabel *****************/
  n = 0;
  xms  = XmStringCreate("Select File to Convert",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(args[n], XmNlabelString, xms); n++;
  XtSetArg(args[n], XmNresizable, True); n++;
  widget_array[WI_OPENLBL_IMG] =
   XmCreateLabel(widget_array[WI_LFORM_IMG], "openLBL_IMG", args, n);

  if (xms) XmStringFree(xms);

  /***************** openBox_IMG : XmFileSelectionBox *****************/
  n = 0;
  XtSetArg(args[n], XmNwidth, 325); n++;
  XtSetArg(args[n], XmNresizePolicy, XmRESIZE_NONE); n++;
  xms  = XmStringCreate("*.img",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(args[n], XmNdirMask, xms); n++;
  xms1  = XmStringCreate("Quit",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(args[n], XmNcancelLabelString, xms1); n++;
  XtSetArg(args[n], XmNresizable, False); n++;
  widget_array[WI_OPENBOX_IMG] =
   XmCreateFileSelectionBox(widget_array[WI_LFORM_IMG], "openBox_IMG", args, n);

  if (xms) XmStringFree(xms);
  if (xms1) XmStringFree(xms1);

  /***************** vSEP_IMG : XmSeparator *****************/
  n = 0;
  XtSetArg(args[n], XmNorientation, XmVERTICAL); n++;
  XtSetArg(args[n], XmNresizable, True); n++;
  widget_array[WI_VSEP_IMG] =
   XmCreateSeparator(widget_array[WI_IMGFORM], "vSEP_IMG", args, n);

  /***************** MForm_IMG : XmForm *****************/
  n = 0;
  XtSetArg(args[n], XmNrubberPositioning, False); n++;
  XtSetArg(args[n], XmNresizePolicy, XmRESIZE_ANY); n++;
  XtSetArg(args[n], XmNresizable, True); n++;
  widget_array[WI_MFORM_IMG] =
   XmCreateForm(widget_array[WI_IMGFORM], "MForm_IMG", args, n);
```

```
/***************** statLBL_IMG : XmLabel *****************/
n = 0;
xms  = XmStringCreate("Statistics",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_STATLBL_IMG] =
  XmCreateLabel(widget_array[WI_MFORM_IMG], "statLBL_IMG", args, n);

if (xms) XmStringFree(xms);

/***************** hSEP_IMG : XmSeparator *****************/
n = 0;
XtSetArg(args[n], XmNseparatorType, XmDOUBLE_LINE); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_HSEP_IMG] =
  XmCreateSeparator(widget_array[WI_MFORM_IMG], "hSEP_IMG", args, n);

/***************** minxForm_IMG : XmForm *****************/
n = 0;
XtSetArg(args[n], XmNrubberPositioning, False); n++;
XtSetArg(args[n], XmNresizePolicy, XmRESIZE_ANY); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_MINXFORM_IMG] =
  XmCreateForm(widget_array[WI_MFORM_IMG], "minxForm_IMG", args, n);

/***************** minxLBL_IMG : XmLabel *****************/
n = 0;
xms  = XmStringCreate("Min X: ",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_MINXLBL_IMG] =
  XmCreateLabel(widget_array[WI_MINXFORM_IMG], "minxLBL_IMG", args, n);

if (xms) XmStringFree(xms);

/***************** minxtxt_IMG : XmTextField *****************/
n = 0;
XtSetArg(args[n], XmNwidth, 70); n++;
XtSetArg(args[n], XmNresizable, False); n++;
widget_array[WI_MINXTXT_IMG] =
  XmCreateTextField(widget_array[WI_MINXFORM_IMG], "minxtxt_IMG", args, n);

/***************** minyForm_IMG : XmForm *****************/
n = 0;
XtSetArg(args[n], XmNrubberPositioning, False); n++;
XtSetArg(args[n], XmNresizePolicy, XmRESIZE_ANY); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_MINYFORM_IMG] =
  XmCreateForm(widget_array[WI_MFORM_IMG], "minyForm_IMG", args, n);

/***************** minyLBL_IMG : XmLabel *****************/
n = 0;
xms  = XmStringCreate("Min Y: ",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_MINYLBL_IMG] =
  XmCreateLabel(widget_array[WI_MINYFORM_IMG], "minyLBL_IMG", args, n);

if (xms) XmStringFree(xms);

/***************** minytxt_IMG : XmTextField *****************/
n = 0;
XtSetArg(args[n], XmNwidth, 70); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_MINYTXT_IMG] =
  XmCreateTextField(widget_array[WI_MINYFORM_IMG], "minytxt_IMG", args, n);

/***************** maxxForm_IMG : XmForm *****************/
n = 0;
XtSetArg(args[n], XmNrubberPositioning, False); n++;
XtSetArg(args[n], XmNresizePolicy, XmRESIZE_ANY); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_MAXXFORM_IMG] =
  XmCreateForm(widget_array[WI_MFORM_IMG], "maxxForm_IMG", args, n);

/***************** maxxLBL_IMG : XmLabel *****************/
n = 0;
xms  = XmStringCreate("Max X: ",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_MAXXLBL_IMG] =
  XmCreateLabel(widget_array[WI_MAXXFORM_IMG], "maxxLBL_IMG", args, n);

if (xms) XmStringFree(xms);

/***************** maxxtxt_IMG : XmTextField *****************/
n = 0;
XtSetArg(args[n], XmNwidth, 70); n++;
XtSetArg(args[n], XmNresizable, False); n++;
widget_array[WI_MAXXTXT_IMG] =
```

```
 XmCreateTextField(widget_array[WI_MAXXFORM_IMG], "maxxtxt_IMG", args, n);

/***************** maxyForm_IMG : XmForm *****************/
n = 0;
XtSetArg(args[n], XmNrubberPositioning, False); n++;
XtSetArg(args[n], XmNresizePolicy, XmRESIZE_ANY); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_MAXYFORM_IMG] =
 XmCreateForm(widget_array[WI_MFORM_IMG], "maxyForm_IMG", args, n);

/***************** maxyLBL_IMG : XmLabel *****************/
n = 0;
xms  = XmStringCreate("Max Y: ",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_MAXYLBL_IMG] =
 XmCreateLabel(widget_array[WI_MAXYFORM_IMG], "maxyLBL_IMG", args, n);

if (xms) XmStringFree(xms);

/***************** maxytxt_IMG : XmTextField *****************/
n = 0;
XtSetArg(args[n], XmNwidth, 70); n++;
XtSetArg(args[n], XmNresizable, False); n++;
widget_array[WI_MAXYTXT_IMG] =
 XmCreateTextField(widget_array[WI_MAXYFORM_IMG], "maxytxt_IMG", args, n);

/***************** defaultPB_IMG : XmPushButton *****************/
n = 0;
xms  = XmStringCreate("Default",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNwidth, 70); n++;
XtSetArg(args[n], XmNresizable, False); n++;
widget_array[WI_DEFAULTPB_IMG] =
 XmCreatePushButton(widget_array[WI_MFORM_IMG], "defaultPB_IMG", args, n);

if (xms) XmStringFree(xms);

/***************** clipPB_IMG : XmPushButton *****************/
n = 0;
xms  = XmStringCreate("Clip",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_CLIPPB_IMG] =
 XmCreatePushButton(widget_array[WI_MFORM_IMG], "clipPB_IMG", args, n);

if (xms) XmStringFree(xms);

/***************** h2SEP_IMG : XmSeparator *****************/
n = 0;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_H2SEP_IMG] =
 XmCreateSeparator(widget_array[WI_MFORM_IMG], "h2SEP_IMG", args, n);

/***************** layersLBL_IMG : XmLabel *****************/
n = 0;
xms  = XmStringCreate("Layers",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_LAYERSLBL_IMG] =
 XmCreateLabel(widget_array[WI_MFORM_IMG], "layersLBL_IMG", args, n);

if (xms) XmStringFree(xms);

/***************** layersSL_IMG : XmScrolledList *****************/
n = 0;
pn = 0;
XtSetArg(args[n], XmNvisibleItemCount, 6); n++;
XtSetArg(pargs[pn], XmNresizable, True); pn++;
widget_array[WI_LAYERSSL_IMG] =
 XmCreateScrolledList(widget_array[WI_MFORM_IMG], "layersSL_IMG", args, n);
        tmpw    =    get_constraint_widget(widget_array[WI_LAYERSSL_IMG],
widget_array[WI_MFORM_IMG]);
 if (tmpw)
  XtSetValues(tmpw, pargs, pn);

/***************** h3SEP_IMG : XmSeparator *****************/
n = 0;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_H3SEP_IMG] =
 XmCreateSeparator(widget_array[WI_MFORM_IMG], "h3SEP_IMG", args, n);

/***************** outLBL_IMG : XmLabel *****************/
n = 0;
xms  = XmStringCreate("Output Options:",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_OUTLBL_IMG] =
 XmCreateLabel(widget_array[WI_MFORM_IMG], "outLBL_IMG", args, n);

if (xms) XmStringFree(xms);
```

```
/***************** binaryTB_IMG : XmToggleButton *****************/
n = 0;
xms  = XmStringCreate("Binary Output",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNset, True); n++;
XtSetArg(args[n], XmNshadowThickness, 1); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_BINARYTB_IMG] =
  XmCreateToggleButton(widget_array[WI_MFORM_IMG], "binaryTB_IMG", args, n);

if (xms) XmStringFree(xms);

/***************** fileTB_IMG : XmToggleButton *****************/
n = 0;
XtSetArg(args[n], XmNset, True); n++;
xms  = XmStringCreate("File Output",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNshadowThickness, 1); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_FILETB_IMG] =
  XmCreateToggleButton(widget_array[WI_MFORM_IMG], "fileTB_IMG", args, n);

if (xms) XmStringFree(xms);

/***************** h4SEP_IMG : XmSeparator *****************/
n = 0;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_H4SEP_IMG] =
  XmCreateSeparator(widget_array[WI_MFORM_IMG], "h4SEP_IMG", args, n);

/***************** v2SEP_IMG : XmSeparator *****************/
n = 0;
XtSetArg(args[n], XmNorientation, XmVERTICAL); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_V2SEP_IMG] =
  XmCreateSeparator(widget_array[WI_IMGFORM], "v2SEP_IMG", args, n);

/***************** RForm_IMG : XmForm *****************/
n = 0;
XtSetArg(args[n], XmNrubberPositioning, False); n++;
XtSetArg(args[n], XmNresizePolicy, XmRESIZE_ANY); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_RFORM_IMG] =
  XmCreateForm(widget_array[WI_IMGFORM], "RForm_IMG", args, n);

/***************** saveLBL_IMG : XmLabel *****************/
n = 0;
xms  = XmStringCreate("Select File to Save As",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNlabelString, xms); n++;
XtSetArg(args[n], XmNresizable, True); n++;
widget_array[WI_SAVELBL_IMG] =
  XmCreateLabel(widget_array[WI_RFORM_IMG], "saveLBL_IMG", args, n);

if (xms) XmStringFree(xms);

/***************** saveBox_IMG : XmFileSelectionBox *****************/
n = 0;
XtSetArg(args[n], XmNwidth, 325); n++;
XtSetArg(args[n], XmNautoUnmanage, False); n++;
XtSetArg(args[n], XmNresizePolicy, XmRESIZE_NONE); n++;
xms  = XmStringCreate("*.dx",XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[n], XmNdirMask, xms); n++;
XtSetArg(args[n], XmNresizable, False); n++;
widget_array[WI_SAVEBOX_IMG] =
  XmCreateFileSelectionBox(widget_array[WI_RFORM_IMG], "saveBox_IMG", args, n);

if (xms) XmStringFree(xms);


/***************** LForm_IMG : XmForm *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetValues(widget_array[WI_LFORM_IMG], args, n);


/***************** openLBL_IMG : XmLabel *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopOffset, 10); n++;
XtSetValues(widget_array[WI_OPENLBL_IMG], args, n);

XtManageChild(widget_array[WI_OPENLBL_IMG]);

/***************** openBox_IMG : XmFileSelectionBox *****************/
```

```
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_OPENLBL_IMG]); n++;
XtSetValues(widget_array[WI_OPENBOX_IMG], args, n);

XtManageChild(widget_array[WI_OPENBOX_IMG]);
XtManageChild(widget_array[WI_LFORM_IMG]);

/***************** vSEP_IMG : XmSeparator *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_LFORM_IMG]); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetValues(widget_array[WI_VSEP_IMG], args, n);

XtManageChild(widget_array[WI_VSEP_IMG]);

/***************** MForm_IMG : XmForm *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_VSEP_IMG]); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetValues(widget_array[WI_MFORM_IMG], args, n);


/***************** statLBL_IMG : XmLabel *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopOffset, 10); n++;
XtSetValues(widget_array[WI_STATLBL_IMG], args, n);

XtManageChild(widget_array[WI_STATLBL_IMG]);

/***************** hSEP_IMG : XmSeparator *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_STATLBL_IMG]); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetValues(widget_array[WI_HSEP_IMG], args, n);

XtManageChild(widget_array[WI_HSEP_IMG]);

/***************** minxForm_IMG : XmForm *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNleftOffset, 5); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightOffset, 5); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_HSEP_IMG]); n++;
XtSetArg(args[n], XmNtopOffset, 15); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetValues(widget_array[WI_MINXFORM_IMG], args, n);


/***************** minxLBL_IMG : XmLabel *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetValues(widget_array[WI_MINXLBL_IMG], args, n);

XtManageChild(widget_array[WI_MINXLBL_IMG]);

/***************** minxtxt_IMG : XmTextField *****************/
n = 0;
XtSetSensitive(widget_array[WI_MINXTXT_IMG], False);
XtSetArg(args[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_MINXLBL_IMG]); n++;
XtSetValues(widget_array[WI_MINXTXT_IMG], args, n);

XtManageChild(widget_array[WI_MINXTXT_IMG]);
XtManageChild(widget_array[WI_MINXFORM_IMG]);
```

```
/***************** minyForm_IMG : XmForm *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_MINXFORM_IMG]); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNrightWidget, widget_array[WI_MINXFORM_IMG]); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_MINXFORM_IMG]); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetValues(widget_array[WI_MINYFORM_IMG], args, n);


/***************** minyLBL_IMG : XmLabel *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetValues(widget_array[WI_MINYLBL_IMG], args, n);

XtManageChild(widget_array[WI_MINYLBL_IMG]);

/***************** minytxt_IMG : XmTextField *****************/
n = 0;
XtSetSensitive(widget_array[WI_MINYTXT_IMG], False);
XtSetArg(args[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_MINYLBL_IMG]); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetValues(widget_array[WI_MINYTXT_IMG], args, n);

XtManageChild(widget_array[WI_MINYTXT_IMG]);
XtManageChild(widget_array[WI_MINYFORM_IMG]);

/***************** maxxForm_IMG : XmForm *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_MINYFORM_IMG]); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNrightWidget, widget_array[WI_MINYFORM_IMG]); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_MINYFORM_IMG]); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetValues(widget_array[WI_MAXXFORM_IMG], args, n);

/***************** maxxLBL_IMG : XmLabel *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetValues(widget_array[WI_MAXXLBL_IMG], args, n);

XtManageChild(widget_array[WI_MAXXLBL_IMG]);

/***************** maxxtxt_IMG : XmTextField *****************/
n = 0;
XtSetSensitive(widget_array[WI_MAXXTXT_IMG], False);
XtSetArg(args[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_MAXXLBL_IMG]); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetValues(widget_array[WI_MAXXTXT_IMG], args, n);

XtManageChild(widget_array[WI_MAXXTXT_IMG]);
XtManageChild(widget_array[WI_MAXXFORM_IMG]);

/***************** maxyForm_IMG : XmForm *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_MAXXFORM_IMG]); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNrightWidget, widget_array[WI_MAXXFORM_IMG]); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_MAXXFORM_IMG]); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetValues(widget_array[WI_MAXYFORM_IMG], args, n);


/***************** maxyLBL_IMG : XmLabel *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetValues(widget_array[WI_MAXYLBL_IMG], args, n);

XtManageChild(widget_array[WI_MAXYLBL_IMG]);
```

```
/***************** maxytxt_IMG : XmTextField *****************/
n = 0;
XtSetSensitive(widget_array[WI_MAXYTXT_IMG], False);
XtSetArg(args[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_MAXYLBL_IMG]); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetValues(widget_array[WI_MAXYTXT_IMG], args, n);

XtManageChild(widget_array[WI_MAXYTXT_IMG]);
XtManageChild(widget_array[WI_MAXYFORM_IMG]);

/***************** defaultPB_IMG : XmPushButton *****************/
n = 0;
XtSetSensitive(widget_array[WI_DEFAULTPB_IMG], False);
XtSetArg(args[n], XmNleftAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNrightWidget, widget_array[WI_MAXYFORM_IMG]); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_MAXYFORM_IMG]); n++;
XtSetArg(args[n], XmNtopOffset, 3); n++;
XtSetValues(widget_array[WI_DEFAULTPB_IMG], args, n);

XtManageChild(widget_array[WI_DEFAULTPB_IMG]);

/***************** clipPB_IMG : XmPushButton *****************/
n = 0;
XtSetSensitive(widget_array[WI_CLIPPB_IMG], False);
XtSetArg(args[n], XmNleftAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_DEFAULTPB_IMG]); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNrightWidget, widget_array[WI_DEFAULTPB_IMG]); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_DEFAULTPB_IMG]); n++;
XtSetArg(args[n], XmNtopOffset, 1); n++;
XtSetValues(widget_array[WI_CLIPPB_IMG], args, n);

XtManageChild(widget_array[WI_CLIPPB_IMG]);

/***************** h2SEP_IMG : XmSeparator *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_CLIPPB_IMG]); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopOffset, 5); n++;
XtSetValues(widget_array[WI_H2SEP_IMG], args, n);

XtManageChild(widget_array[WI_H2SEP_IMG]);

/***************** layersLBL_IMG : XmLabel *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_H2SEP_IMG]); n++;
XtSetArg(args[n], XmNtopOffset, 5); n++;
XtSetValues(widget_array[WI_LAYERSLBL_IMG], args, n);

XtManageChild(widget_array[WI_LAYERSLBL_IMG]);

/***************** layersSL_IMG : XmScrolledList *****************/
pn = 0;
XtSetArg(pargs[pn], XmNleftAttachment, XmATTACH_FORM); pn++;
XtSetArg(pargs[pn], XmNrightAttachment, XmATTACH_FORM); pn++;
XtSetArg(pargs[pn], XmNtopAttachment, XmATTACH_WIDGET); pn++;
XtSetArg(pargs[pn], XmNbottomAttachment, XmATTACH_NONE); pn++;
XtSetArg(pargs[pn], XmNleftOffset, 15); pn++;
XtSetArg(pargs[pn], XmNrightOffset, 15); pn++;
XtSetArg(pargs[pn], XmNtopWidget, widget_array[WI_LAYERSLBL_IMG]); pn++;
        tmpw = get_constraint_widget(widget_array[WI_LAYERSSL_IMG],
widget_array[WI_MFORM_IMG]);
if (tmpw)
  XtSetValues(tmpw, pargs, pn);

XtManageChild(widget_array[WI_LAYERSSL_IMG]);

/***************** h3SEP_IMG : XmSeparator *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
        tmpw1 = get_constraint_widget(widget_array[WI_LAYERSSL_IMG],
widget_array[WI_MFORM_IMG]);
XtSetArg(args[n], XmNtopWidget, tmpw1); n++;
```

```
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopOffset, 10); n++;
XtSetValues(widget_array[WI_H3SEP_IMG], args, n);

XtManageChild(widget_array[WI_H3SEP_IMG]);

/***************** outLBL_IMG : XmLabel *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_H3SEP_IMG]); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopOffset, 5); n++;
XtSetValues(widget_array[WI_OUTLBL_IMG], args, n);

XtManageChild(widget_array[WI_OUTLBL_IMG]);

/***************** binaryTB_IMG : XmToggleButton *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_OUTLBL_IMG]); n++;
XtSetArg(args[n], XmNleftOffset, 15); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_OUTLBL_IMG]); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopOffset, 5); n++;
XtSetValues(widget_array[WI_BINARYTB_IMG], args, n);

XtManageChild(widget_array[WI_BINARYTB_IMG]);

/***************** fileTB_IMG : XmToggleButton *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_BINARYTB_IMG]); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_OPPOSITE_WIDGET); n++;
XtSetArg(args[n], XmNrightWidget, widget_array[WI_BINARYTB_IMG]); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_BINARYTB_IMG]); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetValues(widget_array[WI_FILETB_IMG], args, n);

XtManageChild(widget_array[WI_FILETB_IMG]);

/***************** h4SEP_IMG : XmSeparator *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNtopWidget, widget_array[WI_FILETB_IMG]); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopOffset, 5); n++;
XtSetValues(widget_array[WI_H4SEP_IMG], args, n);

XtManageChild(widget_array[WI_H4SEP_IMG]);
XtManageChild(widget_array[WI_MFORM_IMG]);

/***************** v2SEP_IMG : XmSeparator *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_MFORM_IMG]); n++;
XtSetValues(widget_array[WI_V2SEP_IMG], args, n);

XtManageChild(widget_array[WI_V2SEP_IMG]);

/***************** RForm_IMG : XmForm *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNleftWidget, widget_array[WI_V2SEP_IMG]); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetValues(widget_array[WI_RFORM_IMG], args, n);


/***************** saveLBL_IMG : XmLabel *****************/
n = 0;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_NONE); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNtopOffset, 10); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_NONE); n++;
XtSetValues(widget_array[WI_SAVELBL_IMG], args, n);

XtManageChild(widget_array[WI_SAVELBL_IMG]);
```

```
 /***************** saveBox_IMG : XmFileSelectionBox *****************/
 n = 0;
            tmpw1       =       XmSelectionBoxGetChild(widget_array[WI_SAVEBOX_IMG],
XmDIALOG_OK_BUTTON);
 XtUnmanageChild(tmpw1);
            tmpw1       =       XmSelectionBoxGetChild(widget_array[WI_SAVEBOX_IMG],
XmDIALOG_CANCEL_BUTTON);
 XtUnmanageChild(tmpw1);
            tmpw1       =       XmSelectionBoxGetChild(widget_array[WI_SAVEBOX_IMG],
XmDIALOG_HELP_BUTTON);
 XtUnmanageChild(tmpw1);
 XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
 XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
 XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
 XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
 XtSetArg(args[n], XmNtopWidget, widget_array[WI_SAVELBL_IMG]); n++;
 XtSetValues(widget_array[WI_SAVEBOX_IMG], args, n);

 XtManageChild(widget_array[WI_SAVEBOX_IMG]);
 XtManageChild(widget_array[WI_RFORM_IMG]);

 /*
  * Allocate memory for the widget array to return
  */
  *warr_ret = (Widget *) malloc(sizeof(Widget)*35);
  (void) memcpy((char *)*warr_ret,
            (char *)widget_array,
        sizeof(Widget)*35);

 /*
  *  Return the first created widget.
  */
 return widget_array[WI_IMGDLGSHELL];
}



/*************************************************************
*/
```

BIBLIOGRAPHY

[1] Bloomer, John. Power Programming with RPC. O'Reilly & Associates, 1992.

[2] Conger, James L. Windows API Bible: The Definitive Programmer's Reference. Waite Group Press, 1992.

[3] Eddon, Guy. RPC for NT: Building Remote Procedure Calls for Windows NT Networks. R&D Publications, Inc., 1994.

[4] Petzold, Charles. Programming Windows 3.1. 3rd ed. Microsoft Press, 1992.

[5] Pohl, Ira. Object-Oriented Programming Using C++. The Benjamin/Cummings Publishing Company, Inc., 1993.

[6] Shirley, John., and Ward Rosenberry. Microsoft RPC Programming Guide. O'Reilly & Associates, Inc., 1995.

[7] Yao, Paul. Borland C++ 4.0 Programming for Windows. Random House Electronic Publishing., 1994.

[8] Young, Douglas A. Object-Oriented Programming with C++ and OSF/Motif. 2nd ed. Prentice Hall PTR, 1995.

[9] Borland International, Inc., "Borland C++ 5.01 Object Windows Library On-Line Reference Guide", Borland International, 1997 (on-line reference guide and help system included as part of the Borland C++ 5.01 system distribution for Windows 95/NT).

[10] Young, Douglas A., The X Windows System: Programming and Applications with Xt OSF/MOTIF Edition. Prentice Hall, 1990.

[11] Ferguson, Paula M. and Dan Heller. Motif Programming Manual. O'Reilly & Associates, 1991-1994.

[12] Ferguson, Paula M. Motif Reference Manual. O'Reilly & Associates, inc. 1993.

[13] Prata, Stephan. C++ Primer Plus. 2nd Edition. Waite Group Press, 1995.