XGRASS User's Guide

Kurt Buehler
Robert Beasley
Meredith McCarthy
Andrew Burnett

United States Army
Construction Engineering Research Laboratory
PO Box 9005
Champaign, Illinois 61826-9005

Contents

List of Tables ix
List of Figures xi

CHAPTER 1 Introduction 1-1

Organization of the XGRASS Software 1-1

Configuration of the XGRASS Software 1-2

Window Manager Configuration 1-2

CHAPTER 2 The XGRASS Executive Menu System 2-1

XGRASS Start-up 2-1

Databases, Locations, and Mapsets 2-1
Sessions 2-2
Adding New Sessions, Databases, Locations, or Mapsets 2-3
Command-Line Options 2-4

The XGRASS Session 2-5

The Specification File 2-5
Mnemonic and Accelerators 2-6
XGRASS Menu System Functions 2-6
The Default XGRASS Menu 2-7
Operating the Default Menu System 2-7
Selecting GRASS Commands from the Default Menu System 2-7

Using the GRASS Shell 2-8
Command Histories 2-8
Getting Help 2-9
XGRASS Context Help 2-9
Command-specific Help 2-9
GRASS Topics 2-10
GRASS On-Line Glossary 2-10
GRASS User's Manual 2-11

XGRASS Shutdown 2-11

CHAPTER 3 The **xclip** Command- Line Interfacing Program 3-1

The **xclip** Interface 3-1

Title 3-2
Description 3-2
Parameters 3-3
Other Options Available 3-3
Current Command 3-3
Editor 3-3
Execute 3-3
Reset 3-4
Cancel 3-4
Help 3-4

The **xclip** Input Script Format 3-4

General Interface Description 3-4
program 3-6
title 3-7
commandstring 3-7
description 3-7
help 3-7
helpfile 3-7
helpwidgetref 3-8
errorcodes 3-8
capture 3-8
Main Interface Description 3-8
parameter 3-8
flag 3-9
description 3-9
type 3-9
optional 3-10
Optional Keywords 3-10

default 3-10
help 3-11
helpfile 3-11
precludes 3-11
requires 3-11
multiple 3-11
input 3-12
Dialogs 3-12
Command-Line Options 3-12

CHAPTER 4 Other Related Programs 4-1

xgdisplay 4-1
Graphical Element Buttons 4-2
Drawing Graphical Elements 4-3
Selecting Graphical Elements 4-3
Moving/Resizing Graphical Elements 4-3
Reshaping Graphical Elements 4-4
Pulldown Menu 4-4
Drawing Area 4-8
Popup Menu 4-8
Information Panel 4-8
Open Dialog 4-9
Save Dialog 4-9
Modify Colors Dialog 4-9
Grid Attributes Dialog 4-10
Legend Attributes Dialog 4-11
Scale Attributes Dialog 4-13
Label Attributes Dialog 4-14
Graphic Tool Box 4-14
Modify the Region Dialog 4-14
Color Editing Dialog 4-17
Font Dialog 4-18
Query Raster Dialog 4-19
Delete Vector and Delete Sites Dialog 4-19
xgbrowser 4-19
Command-Line Usage 4-20
xgregion 4-22
Command-Line Usage 4-22
xgaccess 4-22

xgmapsets 4-23
xgmapcalc 4-23
xgreclass 4-24
Command-Line Usage 4-25
xgbuffer 4-26
xgsupport 4-28
Command-Line Usage 4-31

CHAPTER 5 The **xgen** Application Generator Program 5-1

Introduction 5-1
Input Script 5-3
Program Operation 5-4
Hello, World 5-5
Abstract Example 5-5

Functional Reference 5-7
Resources 5-7
Common Procedural Resources 5-8
Command Strings 5-9
The `interactive` Resource 5-10
Global Resources 5-10
Environments 5-10
Shells 5-11
The `decorations` and `functions` resources 5-12
Menu Shells 5-12
Commandboard Shells 5-14
Interface Objects 5-14
The `updateobject` resource 5-15
Label 5-16
Separator 5-17
Button 5-17
Message 5-17
Textentry 5-17
Multiline 5-18
List 5-18
Toggle 5-19
Slider 5-19
Table 5-19
Pulldown 5-20

xgen Code Examples 5-20
Variables 5-21
Object Values 5-21
Dynamic Menus 5-21
Updating Object Values 5-22

Appendix A **xgrass** Default Menu Specification A-1

List of Tables

Table 1 XGRASS menu system functions. 2-6
Table 2 Common Procedural Resources. 5-8
Table 3 Global Resources 5-11
Table 4 Environment Resources 5-11
Table 5 Shell Resources 5-11
Table 6 Values for the decorations Resource 5-13
Table 7 Values for the functions Resource 5-13
Table 8 Menu Shell-Specific Resources 5-14
Table 9 Commandboard Shell-Specific Resources 5-14
Table 10 Object Resources 5-15
Table 11 Object Procedural Resources 5-16

List of Figures

Figure 1 The XGRASS start-up dialog window. 2-2
Figure 2 The “Add Items...” sub-dialog. 2-3
Figure 3 The basic syntax of the XGRASS menu specification file. 2-5
Figure 4 XGRASS default menu system, 2-7
Figure 5 The XGRASS “Save/Exit Session Dialog Window.” 2-11
Figure 6 The **d.mon xclip** interface. 3-2
Figure 7 The **d.mon xclip** script. 3-5
Figure 8 The **xgdisplay** graphical interface with major functional areas designated. 4-2
Figure 9 The Save Dialog. 4-9
Figure 10 The Modify Colors Dialog. 4-10
Figure 11 The Grid Attributes Dialog. 4-11
Figure 12 Legend Attributes Dialog. 4-12
Figure 13 The Bar Scale Attributes Dialog and examples of bar scale styles. 4-13
Figure 14 The Graphic Tool Box dialog. 4-15
Figure 15 The Modify Region Dialog, along with the Grid Options sub-dialog. 4-16
Figure 16 The color editing dialog. 4-18
Figure 17 The Font Dialog. 4-18

Figure 18 The Query Raster Dialog. 4-19
Figure 19 The Delete Vector Dialog. 4-20
Figure 20 Example **xgbrowser** showing a single panel. 4-21
Figure 21 The **xgaccess** interface. 4-22
Figure 22 The **xgmapsets** interface. 4-23
Figure 23 The **xgmapcalc** interface. 4-24
Figure 24 The **xgreclass** interface. 4-26
Figure 25 The **xgbuffer** interface. 4-27
Figure 26 The **xgsupport** main panel, and associated sub-dialogs. 4-29
Figure 27 The **xgcats** dialog, invoked from the **xgsupport** main panel. 4-30
Figure 28 The **xghistory** dialog, invoked from the **xgsupport** main panel. 4-30
Figure 29 An application hierarchy. 5-2
Figure 30 The basic organization of the input script. 5-4
Figure 31 The Hello, World input script. 5-5
Figure 32 The Hello, World interface. 5-6
Figure 33 An abstract example. 5-6
Figure 34 A script based on the example from Figure 33. 5-7
Figure 35 A partial script detailing the pulldown syntax. 5-21
Figure 36 An example of the use of variables. 5-21
Figure 37 An example of retreiving object values. 5-22
Figure 38 A dynamic menu example. 5-22
Figure 39 An example of object updating and the table object. 5-23

CHAPTER 1

Introduction

This chapter describes the overall organization of XGRASS, an X Window System and OSF/Motif based graphical user interface to GRASS 4.1. Overall user configuration issues will also be discussed.

Organization of the XGRASS Software

XGRASS is composed of many individual components which interact together. The term "XGRASS" refers to the collection of all the components which make up the XGRASS system. These components are:

1. An executive menu system (described in "The XGRASS Executive Menu System" on page 2-1),
2. The X Command Line Interface Program (described in "The xclip Command-Line Interfacing Program" on page 3-1),
3. Other related programs (described in "Other Related Programs" on page 4-1), and
4. xgen (described in "The xgen Application Generator Program" on page 5-1).

All of the functionality of XGRASS is available through a central executive menu system. Once GRASS 4.1 is installed and the user implements any needed configuration (described at the end of this chapter), this menu is activated by typing

```
xgrass
```

It is no longer necessary to access GRASS using the **`grass4.1`** command. XGRASS provides a new interface to GRASS, and the "replacement" for the GRASS shell is the **`xgrass`** menu. The contents of this menu may be reconfigured in any way you desire. Customizing your XGRASS system is no harder than customizing your window manager; in fact, the two techniques are very similar.

Essentially, any UNIX™ commands may be added to the XGRASS menu system. You may find that you want to include some commonly used programs which are not in any way related to GRASS. For example, a system administrator may wish to have a personalized XGRASS menu bar with an additional menu called "SysAdmin." This menu might contain commands which list the current users, check printer status, or kill a runaway process.

The XGRASS menu accesses existing GRASS commands in a variety of ways. Many interfaces to the GRASS commands were created using a facility called **`xclip`** (X Command Line Interface Program). **`xclip`** creates a consistent user-interface for any command-line oriented program. It operates by reading an ASCII text file called a *script*, which defines the functionality of an interface to a command-line program. **`xclip`** scripts are easy to write and result in easy to use interfaces that allow a user to see the options which are available as well as use file selectors to choose the file arguments. For example, an **`xclip`** script could be written for the UNIX **`tar`** command, thus removing the need to remember the many arguments of this rather cryptic utility. The XGRASS menu system supports the addition of **`xclip`** scripts as menu items.

XGRASS includes the **`xgen`** program, which allows the easy creation of simple user interfaces. **`xgen`** is located in the contributed section of the GRASS source tree, and has its own compilation procedure. If you cannot access **`xgen`**, consult the person who installed GRASS on your machine for more information.

The XGRASS help system is an important feature, allowing users to get relevant help when they need it, in an organized and intuitive fashion. Nearly all XGRASS dialog boxes support access to the help system. The help system itself provides a HyperText-like interface to GRASS help files, man pages, and glossaries.

Configuration of the XGRASS Software

XGRASS is installed as part of the GISGEN compilation procedure for GRASS. See the "GRASS Installation Guide" for more details. Once compiled, it is useful to configure you window manager to prepare for correct operation of XGRASS.

Window Manager Configuration

To more comfortably accommodate XGRASS, you may wish to make certain adjustments to the configuration of your window manager. If you use **`mwm`**, the Motif Window Manager, you should include the following resources in your `.Xdefaults` file (this file should be placed in your home directory):

```
Mwm*keyboardFocusPolicy:pointer
Mwm*interactivePlacement:False
```

If you use **twm** (or some variant of **twm**), include the following resources in your `.Xdefaults` file:

Dialog boxes must have a title bar in order to be closed with the window manager (instead of Control-C). To add a title bar to dialog boxes, add the following line to your .twmrc file:

```
DecorateTransients
```

A right-click in the title bar can close a dialog box. To enable that, add this line to your .twmrc file:

```
Button3 = : title : f.delete
```

See your window manager documentation for more information.

CHAPTER 2

The XGRASS Executive Menu System

The "heart and soul" of XGRASS is the XGRASS executive menu system. It provides access to all of the interfaces that make up the XGRASS interface to GRASS. The menu system itself is driven by an ASCII (plain text) specification file that describes the menu system and its functionality. The menu system has several built-in features that provide some very useful capabilities not present in GRASS. This chapter describes the interface to GRASS via the XGRASS executive menu system. More specifically, it will describe:

- the start-up procedure
- how to use the menu system (including customizing the menu system)
- the shutdown procedure

XGRASS Start-up

Databases, Locations, and Mapsets

GRASS needs to know the following three pieces of information:

- `DATABASE`—The UNIX directory where the GRASS data being accessed is located. A `DATABASE` may contain several locations. An example `DATABASE` might be `/data/grass`.
- `LOCATION` —The geographical area of interest. Locations usually contain several `MAPSET`s, but always contain a `PERMANENT` mapset. An example location is *spearfish*, the location distributed with the standard GRASS software.

- `MAPSET`—A `MAPSET` can be any name, and is used to keep your analysis of a geographic location separate from someone else's. An example mapset might be *smith* or *jone*

XGRASS implements a mechanism for soliciting this information from the user. This mechanism consists of the dialog window pictured in Figure 1. The user is

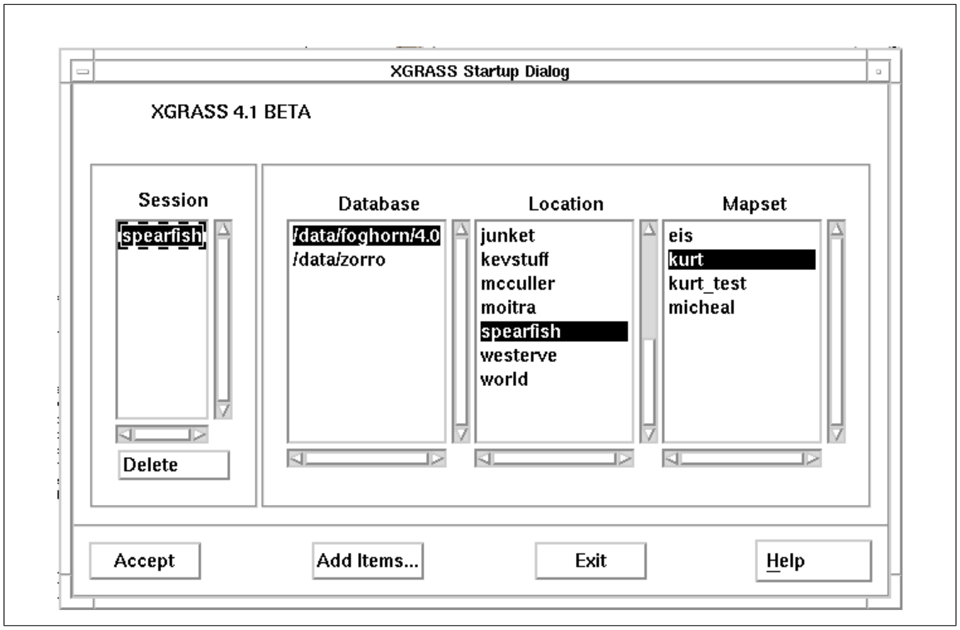

Figure 1 The XGRASS start-up dialog window.

presented with four lists: 1) Session, 2) Database, 3) Location, and 4) Mapset. The Session list, and sessions themselves, will be described in the next section. The Database list in Figure 1 contains two entries (`/data/foghorn/4.0`, and `/data/zorro`). When the user selects one of these databases, the Location list is automatically filled in with the locations available in that database. When the user selects a location, the Mapset list is automatically filled in with the mapsets that he or she owns. When a mapset is selected, XGRASS has the three pieces of information it needs to start-up a session. If the user selects **Accept**, the start-up dialog window will go away and the menu system will take its place on the screen.

Sessions

Sessions provide a way to save database, location, and mapset information into a named session. This mechanism may be used as a shortcut for users who often work in the same database, location, and mapset. The session not only stores database, location, and mapset information, but also a command history described later in this chapter (page 8). Once the user has saved or created a session, it will appear in the Session list of the start-up dialog window. When a session is selected, the associated database, location, and mapset will appear selected in the other three lists.

Adding New Sessions, Databases, Locations, or Mapsets

The "`Add Items...`" button allows you to create a session, add a database, create a location, or create a mapset. The add items sub-dialog window is depicted

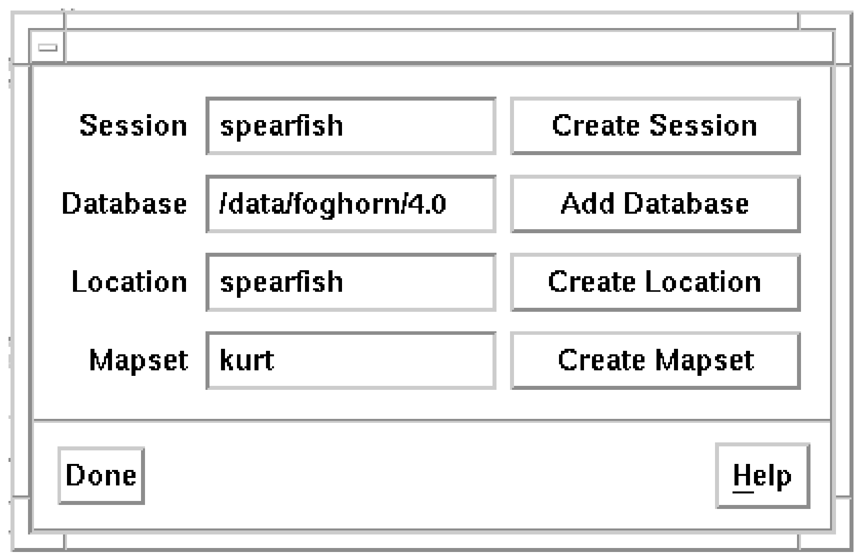

Figure 2 The "Add Items..." sub-dialog.

in Figure 2. The start-up dialog window and the add items dialog window are "connected," meaning that when a selection is made in the start-up dialog window, the corresponding field in the add items dialog window is filled in, and vice versa.

To add a new session, you simply:

1. Identify an existing database, location, and mapset triplet by typing them into the fields of the "add items" dialog window or by selecting them from the lists in the start-up dialog.
2. Type in the name of the session that you wish to create.
3. Press the Create Session button.

XGRASS will not allow you to create a new database. Instead, it allows you to add an existing database to your personal `databases` list (simply execute the command **`xgdatabases`**).

To add a new database:

1. Enter the path to the new database.
2. Select AddDatabase[1].

To create a new location:

1. Identify an existing database by typing it into the Database field in the Add Items dialog window or by selecting one from the Database list in the start-up dialog window.
2. Type the name of the location you want to create into the Location field of the Add Items dialog window.
3. Press Create Location. This will bring up an informational dialog window that details the information you will need to create a location. This information is as follows:
   a. The coordinate system for the new location.
   b. The zone for the location (UTM and State Plane coordinate systems).
   c. The coordinates of the geographical region of interest and the default grid resolution.
   d. A short description or title for the new location.

   If you have all of the required information, select Yes, otherwise obtain the required information and repeat the previous steps. Once you have all the information and have pressed Yes, a dialog window will appear that requests this information.
4. Fill in the required information. Be sure to press the "Set Default Region..." button. This will bring up a region dialog window. Once all the information is defined as desired, press OK and the location will be created.

To create a new mapset:

1. Identify an existing database and location by typing them into the appropriate fields of the add items dialog window or by selecting them from the lists in the start-up dialog window.
2. Type the name of the mapset you wish to create.
3. Press the Create Mapset button and the mapset will be created.

Command-Line Options

The xgrass command has the following command-line options and syntax:

```
xgrass [-v][-help][-patch][-version]
    [-Idir][-Dname][-Dname=def][-f specification-file]
    [-dbase XX -location YY -mapset ZZ][session-file]
```

Where:

1. If the Add Database button is "grayed out," then you do not have a personal databases list (use the xgdatabases command to create one).

`-v`	Causes **`xgrass`** to run in verbose mode. This is useful for debugging specification file formats.
`-help`	Prints a help message about command-line usage.
`-patch`	Prints version (patchlevel) information.
`-version`	Prints version (patchlevel) information.
-dbase	Uses the next argument as the database name.
`-location`	Uses the next argument as the location name.
`-mapset`	Uses the next argument as the mapset name.
`-I`*dir*	Adds *dir* to the **`cpp(1)`** include search list.
`-D`*name*	Defines *name* as 1 (true) to **`cpp(1).`**
`-D`*name=def*	Defines *name* as *def* to **`cpp(1).`**
`-f`	Uses the next argument as the specification file (see page 5).
session-file	Uses this session file at start-up, thus avoiding the start-up dialog altogether (see page 2).

The XGRASS Session

Once the start-up dialog has been satisfied and the database, location, and mapset information are known, the menu system can be created. The menu system and its functionality is completely described by a specification file.

The Specification File

A default specification file exists that is used by XGRASS unless you have created a customized specification file. The default specification file is located in your home directory and is called `xgrass-menu`. This section describes the format of this file.

The specification file contains a list of menu descriptions. Figure 3 details the syntax of a menu description:

```
menu-name
{
      item1   [mnemonic]   [accelerator]   function   [arguments]
      item2   [mnemonic]   [accelerator]   function   [arguments]
   .
   .
      item#   [mnemonic]   [accelerator]   function   [arguments]
}
```

Figure 3 The basic syntax of the XGRASS menu specification file.

The menu is given a name, and each item within the menu is given a name (this is used as a label, in most cases). The item is followed by an optional mnemonic and/or accelerator (these are explained in further detail in the next section). Then a function is given along with its arguments, if any are needed.

A menu description for a menu named DefaultMenu must exist in the specification file. This menu description is used for the top-level menu bar.

Mnemonic and Accelerators

Mnemonics and accelerators are used to provide keyboard shortcuts for users of the menu system. An accelerator is a keyboard sequence that will activate a menu item.

XGRASS Menu System Functions

XGRASS menu functions are used to identify what happens in response to user input or to define the appearance of the menus within the menu system. These functions are listed in Table 1.

Table 1 XGRASS menu system functions.

Name	Description
f.exec	Creates a button in the menu or sub-menu that, when pressed, executes its argument using the Bourne Shell (/bin/sh -c).The item is used as the text for the button.
f.exec_c	Same as f.exec, but captures the output in the XGRASS Output Editor.
f.exec_h	Same as f.exec, but also inserts the argument into the current command history.
f.exec_ch	Same as f.exec, but captures the output and inserts the argument into the command history.
f.xclip	Creates a button in a menu or sub-menu that, when pressed, will invoke a built-in **`xclip`** script using the argument as the name of the **`xclip`** script. The item is used as the text for the button.
f.pulldown	Creates a cascading button which, when invoked, will pulldown the sub-menu described with the name given in the argument. The item is used as the text for the button. This function is only available in the DefaultMenu (the top-level menu).
f.pullright	Creates a button with an arrow that, when pressed, invokes a pullright menu using the name given in the argument to indicate the sub-menu description. This function acts like f.pulldown, if it occurs within a horizontal tear-off menu. The item is used as the text for the button. This function can only be used in pulldown or pullright sub-menus.
f.hmenu	Creates a button that, when pressed, invokes a horizontally oriented tear-off menu using the argument to indicate the sub-menu description. The item is used as the text for the button.
f.menu	Same as f.hmenu.
f.vmenu	Creates a button that, when pressed, invokes a vertically oriented tear-off menu using the argument to indicate the sub-menu description.The item is used as the text for the button.
f.label	Creates a label. The item field is used as the text of the label. This function can only be used in pulldown or pullright sub-menus.
f.separator	Creates a separator. The item name is ignored in this case. This function can only be used in pulldown or pullright sub-menus.

Table 1 XGRASS menu system functions.

Name	Description
f.dbset	Creates a button in a menu or submenu that, when pressed, invokes the start-up dialog. This allows the user to change database location and mapset, save a session, or add new items. The item is used as the text for the button.
f.history_edit	Creates a button that, when pressed, invokes the XGRASS History Editor. This allows the user to edit the current history, save the current history, insert a previously saved history into the current history, or load a saved history in place of the current history. The item is used as the text for the button.
f.history_clear	Creates a button that, when pressed, clears the current history. The item is used as the text for the button.
f.history_replay	Creates a button that, when pressed, replays the current history. The item is used as the text for the button.
f.quit	Creates a button that, when pressed, invokes the shutdown process. The item is used as the text for the button.

The Default XGRASS Menu

4.1 Beta Note: The Default XGRASS Menu depicts the default menu specification shipped with the XGRASS 4.1 Beta version.

Operating the Default Menu System

The menu system generated by the default specification is depicted in Figure 4.

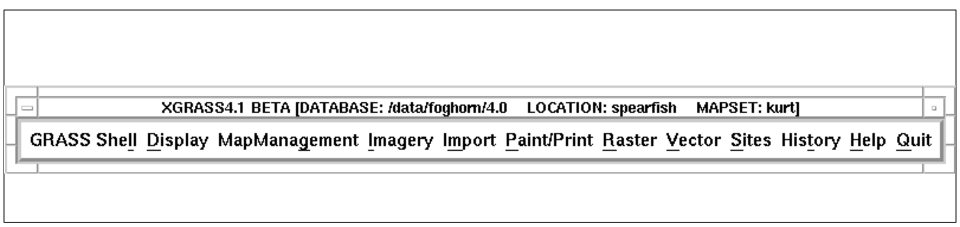

Figure 4 XGRASS default menu system,

Selecting GRASS Commands from the Default Menu System

Selecting GRASS commands via the XGRASS menu system is simple if one understands how the default menu is organized. There is probably no substitute for exploring the menu system to see how it is organized, however, for novices and experts alike, a useful rule-of-thumb exists. The XGRASS default menu system is divided into 12 major functions. Seen in Figure 4: **GRASS Shell**, **Display, Map Management, Imagery, Import, Paint/Print, Raster, Vector, Sites, History, Help,** and **Quit**. Each (not including GRASS Shell, History, Help, and Quit) contains those GRASS commands that begin with the letter that is underlined in that function's name. For example, all of the d. commands are contained in the **Display** menu, all of the r. commands are contained in the **Raster** menu, and so on.

Using the XGRASS menu system is as easy as pointing and clicking with your mouse. For example, suppose you are interested in performing the rather common function in GRASS—displaying a raster map (**`d.rast`** is the actual GRASS command that is executed). To start your search in the menu system for this operation, you would point on the **Display** option at the XGRASS menu bar and click on it with your left mouse button. This pulls down a sub-menu that contains a number of more specific functions. Since you are interested in displaying a raster map, you would then point and click on the word **Display** that appears in this sub-menu. Again, another sub-menu appears which contains a number of other, even more specific, options. Since you are interested in displaying a *raster* map, you would select the **Raster** option. This then brings you to another sub-menu. At this point, you would select the **Display Raster Map** option. Selecting this option would take you directly to an xclip program, which is the graphical user interface for **`d.rast`**. Our discussion of how to use these interfaces is the topic of Chapter 3.

Using the GRASS Shell

This version of XGRASS does not supply an interface for every GRASS command, thus the **GRASS Shell** menu item is provided. This menu item, when invoked, executes a terminal window running the GRASS command shell. To invoke any GRASS command, simply type the command as in the normal command-line version of GRASS. To exit the GRASS command shell, simply type `exit`.

Command Histories

XGRASS provides a useful tool for capturing, editing, and replaying the commands invoked during your current session. This feature is invoked from the **History** option on the XGRASS menu bar by pointing and clicking on it with your left mouse button.

To view and/or edit the current history, select the **View/Edit Current History** option. Using the current history editor is like using any other word processing program. You can cut, copy, and paste sections of the history file to your liking and then replay those commands at a later time. This, of course, can be useful for building demonstrations where you wish to have a series of GRASS commands execute in a given order, but don't want to manually step through the process of creating your demonstration in real time. What you can do is prepare your demonstration ahead of time and simply play it back at a later date using the replay current history command.

To replay the current history, select the **History** option from the XGRASS menu bar, and then select the **Replay Current History** option.

To clear the current history, select the **History** option from the XGRAS menu bar, and then select the **Clear Current History** option.

Getting Help

XGRASS provides the user with several different types or "levels" of help. These types of help range from the very specific, such as informing the user as to what information is required in an individual field while using XGRASS, to the more general, such as finding out the definition of a particular GRASS or GIS term.

In all, XGRASS provides five different types of help. These are: context-sensitive help, command-specific help, help on various GRASS topic areas, a GRASS and GIS glossary of terms, and an on-line user's manual. This help system was developed in a hypertext environment. By selecting any of the highlighted keywords or phrases in these documents, the user can link to various types of information relating to the usage of GRASS and XGRASS itself.

The following sections describe how to access and use each type of help.

XGRASS Context Help

Each XGRASS dialog window provides the user with "context" help. That is, help on the actual buttons, boxes, and interactive areas located in that dialog window. This type of help is accessed by selecting the **Help** menu located at the bottom right hand corner of each XGRASS dialog window. The first selection on the menu says "On Context," and when selected will change the cursor to a question mark. By placing the question mark over any area, button, or box in the dialog window and clicking, an information, or "i," box will appear with a description of the item selected as well as instructions for its use. This type of help is useful when the function of a particular area, box, or button is not immediately apparent.

Command-specific Help

Command specific help is also accessed under each XGRASS dialog window's Help menu. Listed are GRASS topics that relate specifically to the command being invoked. For example, if "List GRASS Files" is selected from the XGRASS menu system, that window will pop-up. The first box at the top of the dialog window provides a short description of the GRASS command that will be executed to list the GRASS files (in this case, **`g.list`**). In addition, by pulling down the dialog window's Help menu located in the bottom right-hand corner, there will be a menu selection entitled "Database Management Tools." The information contained in this topic area relates specifically to the use of database management tools, of which 'g.list' is one. This type of information is useful when knowledge is needed on how this command can be applied in performing specific tasks.

GRASS Topics

This help resource provides by topic area information about GRASS and its many capabilities. Each topic categorizes GRASS commands by their specific application within GRASS. General information is provided on a topic (e.g., map development), and the GRASS commands relevant to that topic are given. The option to access more detailed information on each GRASS command is also provided. GRASS Topics is accessed from the XGRASS menu bar under Help.

For example, suppose you are interested in copying raster files. By selecting **Help** from the XGRASS menu bar and by clicking on it with your left mouse button, the GRASS Topics dialog window appears. Next, you would select **Database Management Tools**. You would then point on the vertical scroll bar and, while holding down your left mouse button, drag the vertical scroll bar downward until the term “g.copy” is in your view. Selecting **g.copy** would then display a dialog window which explains the use of the g.copy command. To see the manual page for this command, you would select the term **g.copy** from the sentence entitled “Manual entry for g.copy.”

GRASS On-Line Glossary

The GRASS Glossary allows you to quickly access definitions of commonly-used GRASS and other GIS terms or phrases. The GRASS Glossary is accessed from the XGRASS menu bar under **Help**.

There are two ways to search the glossary. The first is called “First Letter Searching” and the second is called “Keyword Searching.”

First Letter Searching—Suppose, for example, that you are interested in seeing the term “raster” defined. To perform a first letter search of the term “raster,” you would first select the letter **R** from the “Select” area. Next, you would select the term **raster** from the “Matches” box to see its definition.

Keyword Searching—You can perform a keyword search when you want to see all the words or phrases in the Glossary that begin or contain a given keyword. For example, suppose you would like to see all the words in the Glossary that contain the word “arc.” To start the search for this keyword, you would select the **Keyword Search** button in the bottom left-hand corner of the GRASS Glossary dialog window. After the “Keyword Search” window appears, you would type in the term or phrase (or part of the term or phrase) in the “Keyword” box. In this case, you would type in the word **arc**. Next, since you are interested in seeing all the words that *contain* the word “arc,” you would choose the **Find words containing keyword** option. Next, you would execute the search by pointing and clicking on the **Execute** button. All the words and phrases in the Glossary that contain the word “arc” are then displayed in the “Matches” box. Finally, you would select the desired term from the “Matches” box to see its definition.

GRASS User's Manual

The GRASS Manual provides access to either brief or detailed definitions of GRASS Commands. This resource is accessed from the XGRASS menu bar under Help.

XGRASS Shutdown

When you are ready to exit the menu system, simply select **Quit XGRASS** from the "Quit" pulldown menu. This will invoke the Save/Exit Session dialog window depicted in Figure 5.

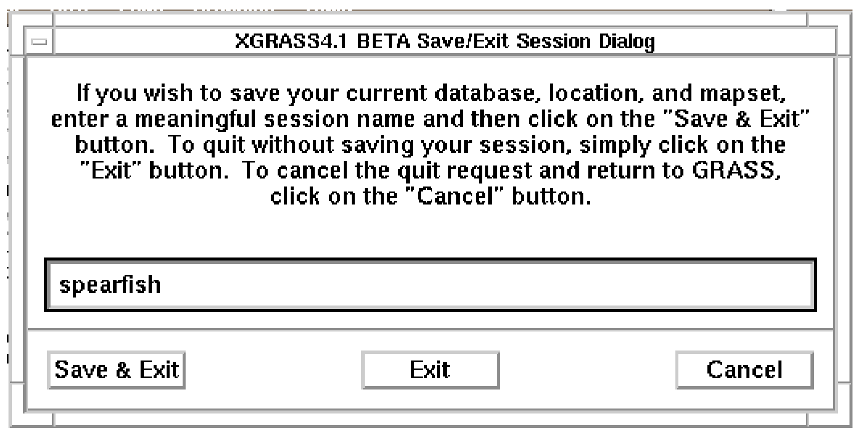

Figure 5 The XGRASS "Save/Exit Session Dialog Window."

Clicking **Save & Exit** will save the current database, location, and mapset, as well as the current command history into the session given by the text field in the dialog window, and then exit. Clicking **Exit** will, of course, exit without saving changes. Clicking **Cancel** will dismiss the dialog window without action.

CHAPTER 3

The `xclip` Command-Line Interfacing Program

XGRASS provides a consistent, easy-to-use interface development package called "XCLIP". **`xclip`** allows the user to quickly develop stand-alone interfaces for GRASS command-line programs via a simple, declarative syntax. The menu system provides a mechanism for the inclusion of such programs in an integrated way, described in "XGRASS Menu System Functions" on page 2-6. In the first section of this chapter, an **`xclip`** interface will be described by using the example of starting up a display monitor. The second section will describe in detail the specific syntax of an **`xclip`** program. The third and final section will describe the comman- line arguments that can accompany the **`xclip`** program itself.

The `xclip` Interface

Suppose for a moment that you are interested in starting the display monitor so that you can display some maps. Experienced users know that the GRASS command that performs this function is **`d.mon`**. However, instead of having to remember the command-line arguments that must be present for this function to work properly, XGRASS provides an easy-to-use interface for this command. This interface comes in the form of an **`xclip`** program.

To execute the **`xclip`** interface for **`d.mon`**, you would first select **Display** from the XGRASS menu bar. Next, you would select **Manage** from the pulldown menu that has been displayed. Because you are interested in starting the display monitor, you would then select **Control Display Monitor** from the pullright menu displayed. This sequence of selections displays the **`xclip`** interface to the **`d.mon`** command depicted in Figure 6 . The following sections describe each of the main portions of this interface.

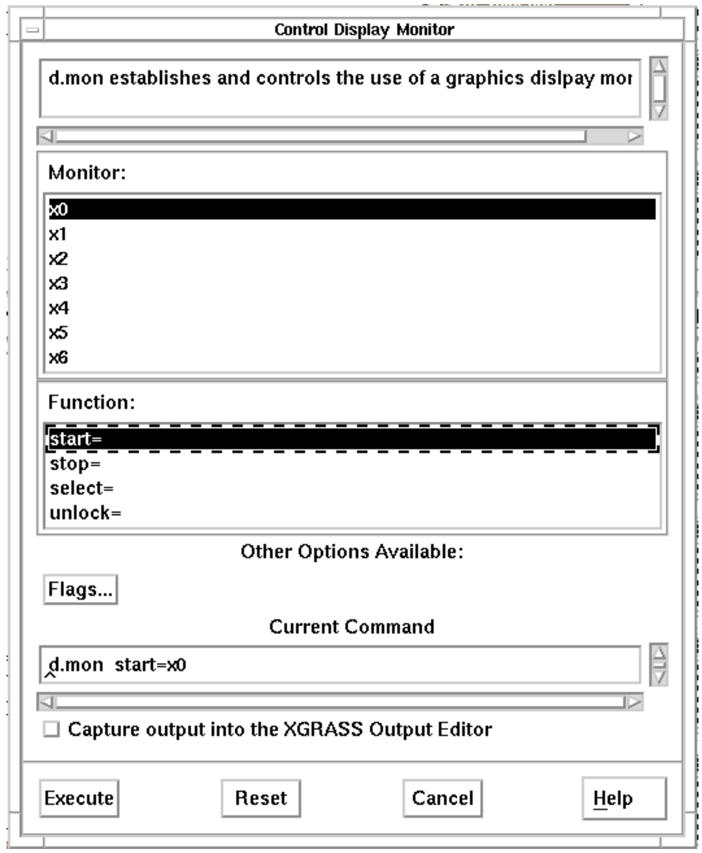

Figure 6 The **`d.mon xclip`** interface.

Title

Each **`xclip`** interface contains a title identical to the last selection the user made from the XGRASS menu bar. This title is located at the very top of the dialog window. Its purpose is to keep the user informed of the major function that this dialog window performs. In our example, the title informs you that you are controlling (starting) the display monitor.

Description

The scrolling window area below the title is the description area. The description area provides the user with a short summary of the GRASS command that this interface invokes. In our example, we are going to be invoking the **`d.mon`** command, so you will see a very short synopsis of the **`d.mon`** command in this window.

Parameters

On this particular interface, there are two parameter areas. The first contains information about the monitor to be displayed. For our example, we would like to use the x0 monitor, so you would use your mouse to select **x0**. When x0 is highlighted, it indicates your choice has been selected. When x0 is not highlighted, it indicates your choice has been de-selected.

The next parameter area in this interface contains information about the function to be performed. Since we are interested in starting the graphics monitor, you would select **start=** using your mouse.

Other Options Available

The Other Options Available area provides the user with optional parameters and flags that can be used by the GRASS command you will be invoking. By pointing and clicking with your mouse on the **Flags...** option, a sub-dialog window appears containing all the optional flags that you may select from. Since none of these options is what we would like to do for this example, you can dismiss this dialog window by pointing and clicking on the **Done** button located at the bottom left-hand corner of the flags dialog window. It is important to note that any number of flags and/or parameters can be located in the Other Options Available area depending on the GRASS command being invoked.

Current Command

You may have noticed as you selected x0 and start= that the area labelled "Current Command" was updated. This area reflects the GRASS command that is being executed along with its respective parameters. In essence, as you make selections using the **`xclip`** interface, you are "building" the GRASS command line. This area allows you to see the command line that you are building.

Editor

The "Capture output into the XGRASS Output Editor" flag allows the user to capture into a user-friendly editor any output produced by the GRASS command being invoked. This output can then be edited and saved by the user as necessary.

Execute

The **Execute** button is used to execute the GRASS command as it appears in the current command area in the dialog window. In our example, we have already selected the monitor (x0) and the function (start=), so we can now execute the command. With your left mouse button, point and click on the **Execute** button. After a moment, the x0 display monitor is displayed. At this point, you could display any number of raster and/or vector maps on this monitor.

Reset

The **Reset** button is used when you would like to reset the current command-line area back to its original, default state. Any selections you previously made would be removed from the current command area, and any defaults would be reinstated.

Cancel

The **Cancel** button is used when you are finished using the current command and wish to remove the **`xclip`** interface from the screen.

Help

The **Help** button is used to get context-sensitive help. To get context-sensitive help, point and click on the **Help** button with you left mouse button and drag down to the option that says “On Context” and then release the mouse button. After your mouse pointer turns into a question mark, place it over some area of interest and then click. This will give you either directions on what to put into a particular field or an explanation of what the function does.

Since you are now done with the **`d.mon`** **`xclip`** interface, you may stop the graphics monitor by selecting **stop=** and pointing and clicking on the **Execute** button. The graphics monitor is then removed from your screen.

The **`xclip`** script for the **`d.mon`** interface is given in Figure 7.

The `xclip` Input Script Format

In the previous example, you were shown the **`xclip`** interface developed for the **`d.mon`** command. In this section, the **`xclip`** input script format will be described. This format has a unique and very easy-to-use syntax. Once you get the gist of how to build **`xclip`** scripts, you will be able to customize your own interfaces in a very short time. The following sections describe each of the available keywords in the **`xclip`** input script as well as their respective parameters.

General Interface Description

The general interface description describes the most general part of the **`xclip`** interface. This description appears at the top of the **`xclip`** input script. The keywords that can be used in this area are: program, title, commandstring, description, help, helpfile, helpwidgetref, errorcodes, and capture. Each of these keywords is described next.

```
program:"d.mon"

title:"Control Display Monitor"

commandstring:"[-(l)(L)(p)(r)(s)] [(function)][(monitor)]"

description:"d.mon establishes and controls the use of a graphics disl-
pay monitor."

help:"This program allows you to establish and Control the Display Moni-
tor."
helpwidgetref:"04.wind.mgmt/04.02.mng.frm:Management of Display Frames"

errorcodes:"0,1"

capture:false

{
 parameter monitor
          description:"Monitor:"
type:enum:"x0,x1,x2,x3,x4,x5,x6"
help:"Select the monitor you wish to manipulate."
                   optional:true
                    default:"x0"
     requires:parameter:function
                precludes:flag:l
                precludes:flag:L
                precludes:flag:p
                precludes:flag:r
                precludes:flag:s
                               ;

 parameter function
         description:"Function:"
type:enum:"start=,stop=,select=,unlock="
help:"Select the function you wish to perform."
                   optional:true
      requires:parameter:monitor
                precludes:flag:l
                precludes:flag:L
                precludes:flag:p
                precludes:flag:r
                precludes:flag:s
                               ;

 dialog "Flags"
             description:"Flags"
 help:"Select the desired flag."

 {

 flag l
                         key:"l"
 description:"List all monitors"
help:"Select this flag if you want to list all the monitors."
                precludes:flag:L
                precludes:flag:p
                precludes:flag:r
                precludes:flag:s
    precludes:parameter:function
     precludes:parameter:monitor
                               ;
```

Figure 7 The **`d.mon xclip`** script.

```
 flag L
                         key:”L”
description:”List all monitors (with current status)”
help:”Select this flag if you want to list all the monitors with their
current status.”
                precludes:flag:l
                precludes:flag:p
                precludes:flag:r
                precludes:flag:s
    precludes:parameter:function
     precludes:parameter:monitor
                               ;

 flag p
                         key:”p”
description:”Print name of currently selected monitor”
help:”Select this flag if you want to print the name of the currently se-
lected monitor.”
                precludes:flag:l
                precludes:flag:L
                precludes:flag:r
                precludes:flag:s
    precludes:parameter:function
     precludes:parameter:monitor
                               ;

 flag r
                         key:”r”
description:”Release currently selected monitor”
help:”Select this flag if you want to release the currently selected mon-
itor.”
                precludes:flag:l
                precludes:flag:L
                precludes:flag:p
                precludes:flag:s
    precludes:parameter:function
     precludes:parameter:monitor
                               ;

flag s
                         key:”s”
description:”Do not automatically select when starting”
help:”Select this flag if you do not want to automatically select this
monitor when starting.”
                precludes:flag:l
                precludes:flag:L
                precludes:flag:p
                precludes:flag:r
      requires:parameter:monitor
                               ;
 }
}
```

Figure 7 The **d.mon xclip** script.

program

The `program` keyword indicates the GRASS program that the interface is for. This is the program that will actually be executed when the **xclip** interface submits the command to the operating system. For example, if we were building an interface for a command called **d.rast**, we would include the following program keyword and its associated parameter as the very first keyword in our script:

```
program:”d.rast”
```

title

The `title` keyword describes the text that will appear at the very top of the dialog window. This title should be very brief and concisely describe the function of the command. For example, if we were building an interface to display vector maps, we might include the following title keyword and associated parameter:

```
title:"Display Vector Maps"
```

commandstring

The `commandstring` keyword describes the syntax of the GRASS command-line being built. Those items inside of the ['s will only be displayed if at least one flag or parameter inside of them is set. Those items inside of the ('s are the internal names used for flags and parameters. In other words, they are variables. An example commandstring and its associated parameters is illustrated below.

```
commandstring:"[-(p)(q)] map=(name1) [color=(name2)]"
```

where, `p` and `q` are optional flags, `name1` is a required parameter, and `name2` is an optional parameter.

description

The `description` keyword is used to describe the GRASS command being invoked. This description should contain text that briefly explains the GRASS command you are developing the interface for. An example description is given below.

```
description:"d.rast displays user-specified raster
files"
```

This description field is the first area displayed after the title in the dialog window.

help

The `help` keyword *at this level* is the most general type of context-sensitive help. It should briefly describe what this particular GRASS command does. The following is an example:

```
help:"This program allows the user to display raster
maps."
```

helpfile

The `helpfile` keyword can be used to hold context-sensitive text help for help that is longer or possibly reusable elsewhere. Again, this help should briefly describe what this particular GRASS command does. The following is an example:

```
helpfile:"$XGRASSHELPDIR/xclip/filename.ext"
```

Notice that the parameter to this keyword is a UNIX directory and filename combination.

helpwidgetref

The `helpwidgetref` keyword will create an entry in the pulldown Help menu at the bottom right-hand corner of the interface. This entry can then correspond to some other file that contains help about the GRASS command being invoked. The following is an example:

```
helpwidgetref:"16.glossary/file.def:Title of File"
```

errorcodes

The `errorcodes` keyword provides a comma-separated list of acceptable exit codes for the command you are interfacing to. Some programs exit with a non-zero exit code even though they exit successfully. In the example below, exit codes of either 0 or 1 will be interpreted as normal by **xclip**.

```
errorcodes:"0,1"
```

capture

The `capture` keyword indicates whether or not the "Capture output into the XGRASS Output Editor" option should be available on the **xclip** interface. Capturing output into the XGRASS Output Editor might be useful for capturing output normally sent to standard output. If the output option is to be available, capture should be set to `true`, as in the example below. If the output option is not to be available, capture should be set to `false`.

```
capture:true
```

Main Interface Description

The interface description always starts with a left curly brace ({) and ends with a right curly brace (}). The main interface description contains all of the information for specific interface parameters and flags. The following sections describe each of the keywords that can be included in the main interface description.

parameter

The `parameter` keyword defines the parameters that are to appear on the command-line. The associated parameter name corresponds directly with the name that appears in the commandstring discussed previously. For example,

```
parameter name1
```

indicates that `name1` is a parameter that is to appear on the command-line of the interface.

flag

The `flag` keyword defines the flags that are to appear on the command-line. As with the parameter keyword, the associated flag name corresponds directly with the name that appears in the commandstring. For example,

```
flag p
```

indicates that `p` is a flag that is to appear on the command-line of the interface.

description

The `description` keyword is used as the label associated with the parameter or flag. Following is an example.

```
description:”Raster map to be used as input:”
```

type

The `type` keyword describes the characteristics of a parameter. There are several different types. Each can have from none to several parameters. The following are the types of parameters that can exist in an **`xclip`** input script as well as examples of their use:

character - The `character` type indicates that the current parameter should be input as a simple alphanumeric field. The character type has no parameters. For example,

```
type:character
```

will produce an input field that will accept any combination of alphanumeric characters.

integer - The `integer` type indicates that the current parameter is to be input as an integer slider. For example,

```
type:integer:”0:100:50”
```

will produce a slider ranging from 0 to 100 starting at 50.

float (or double) - The `float` (or `double`) type indicates that the current parameter is to be input as a floating point slider. For example,

```
type:double:”0:100:50:2”
```

will produce a slider ranging from 0.00 to 1.00 starting at 0.50

logical - The `logical` type indicates that the current parameter is to be input as an on/off or true/false toggle button. For example,

```
type:logical:”Kurt:Bob”
```

will produce a toggle button that displays “Kurt” when set, “Bob” when not set.

enum - The `enum` type indicates that the current parameter is to be input as an enumerated list of choices. For example,

```
enum:"violet,red,orange,yellow,green,blue"
```

will produce a scrolling window containing the colors that may be selected.

filename - The `filename` type indicates that the current parameter is to be input as an ASCII input field capable of accepting any combination of alphanumeric characters. For example,

```
filename:
```

will produce an input field that will accept any combination of alphanumeric characters. In addition, the character type has an optional parameter. Adding the optional "edit" parameter will cause an **editor** button to appear on the screen that, when selected, will invoke a text editor. For example,

```
filename:edit
```

will cause an **editor** button to appear on the screen.

database_element - The `database_element` type indicates that the current parameter is to be input as browser. The available data types are raster, vector, sites, icons, labels, region, and group. For example,

```
type:database_element:raster
```

will allow selection of a raster map using the browser *if* the 'input' keyword (described later) is true.

optional

The `optional` keyword indicates whether or not the current parameter is optional. If it is optional, it will only be displayed on the command line if a value for it has been selected or input by the user. For example,

```
optional:false
```

will display the parameter on the command line even if no value for the parameter has been selected or input.

Optional Keywords

There are also a number of other keywords that are optional. These are described in the following sections.

default

The `default` keyword is used to supply a default value to the current parameter. This default value will be set at run-time. For example,

```
default:"defaultvalue"
```

will set the value of the current parameter to defaultvalue.

help

The `help` keyword at this level is a very specific type of context-sensitive help. The following is an example:

```
help:"Enter the name of the raster map to be used as in-
put."
```

helpfile

The `helpfile` keyword can be used to hold the very specific context-sensitive help for longer text. The following is an example:

```
helpfile:"$XGRASSHELPDIR/xclip/filename.ext"
```

precludes

The `precludes` keyword indicates that giving the current parameter a value will “unset” some other given parameter(s) or flag(s). You may include as many precludes and requires items as you need. For example,

```
precludes:flag:"q"
```

will “unset” the flag `q` if the current parameter or flag is given a value.

```
precludes:parm:"name1"
```

will “unset” the parameter `name1` if the current parameter or flag is given a value.

requires

The `requires` keyword indicates that the current parameter or flag requires that some other flag or parameter be set. For example,

```
requires:flag:"p"
```

will cause the flag `p` to be set if the current parameter or flag receives a value.

```
requires:parm:"name1"
```

will cause the parameter `name1` to be set if the current parameter or flag receives a value.

multiple

The `multiple` keyword indicates that the current parameter can have multiple values. This keyword only works with filename, database_element, and enum types, however. For example,

```
database_element:raster:multiple
```

will allow you to select multiple raster maps from the list of available raster maps.

input

The `input` keyword indicates whether or not this is an input parameter. Input is either true or false. For example,

```
input:true
```

will allow you to input alphanumeric characters into the input field via the keyboard.

Dialogs

Dialogs allow you to place optional and/or other parameter and/or flag information in the "Other Options Available" area of the main dialog window. The keyword descriptions are exactly the same as those described above. The following is an example of a dialog that contains two flags:

```
dialog "Flags"
description:"Flags"
help:"Select the flag you wish to set by pointing and
clicking on it."
{
   flag q
      key:"q"
      description:"Run quietly without printing messages."
      precludes:flag:"p"

   flag p
      Key:"p"
      description:"Print all messages while running."
      precludes:flag:"q";
}
```

Command-Line Options

The **`xclip`** command has the following command-line options and syntax:

`xclip [-v] [-|+c] [-help] [-patch] [-version] [-Idir] [-Dname] [-Dname=def] [-flag name:"on"|"off"] * [-parm name:value] * [script-name`

Where:

`-v`	verbose mode
-/+c	do/do not capture output (overrides script)
`-help`	print this message
`-patch`	print version (patchlevel) information
`-version`	ditto

-Idir	add dir to cpp(1) include search list
-Dname	define name as 1 (true) to cpp(1)
-Dname=def	define name as def to cpp(1)
-flag	default flag “name” to “on” or “off”
-parm	default parameter “name” to “value”

NOTE: -flag and -parm can occur multiple times in the **`xclip`** command-line

CHAPTER 4

Other Related Programs

This chapter describes a set of programs that were custom-built instead of taking advantage of the **`xclip`** interfacing mechanism. These programs are executed from the menu system using the `f.exec` menu system function. Because the semantics of their use is different from interfaces generated using **`xclip`**, we describe them each here. Note that these programs can be called from within the GRASS shell or from shell scripts (assuming the environment variables are set correctly).

xgdisplay

`xgdisplay` is a program for the display and output of GRASS data. It is the beginning of a new set of display and hardcopy output functionality that will be appearing over the next few years. This program allows the user to display GRASS data and simple graphics in a highly interactive way, and to generate PostScript output.

4.1 Beta Note: The 4.1 Beta version of **`xgdisplay`** will not produce hardcopy output.

Figure 8 depicts the **`xgdisplay`** user interface. The interface consists of four major areas; 1) a pulldown menu bar, 2) a row of graphical element buttons, 3) the main drawing area, and 4) an information panel. A popup menu is also provided. Each of these areas will be discussed in detail in the following sections.

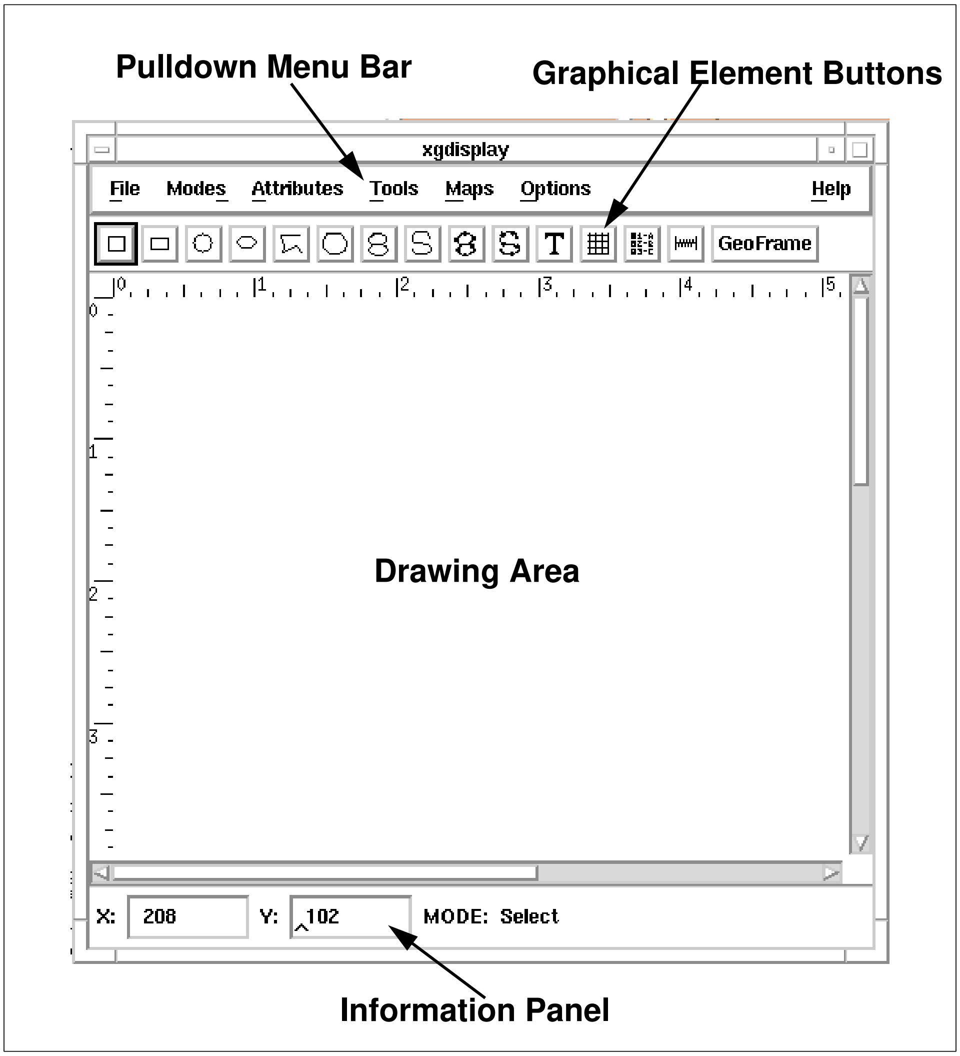

Figure 8 The **xgdisplay** graphical interface with major functional areas designated.

Graphical Element Buttons

Because **xgdisplay** is mostly graphic- oriented we will begin by describing the graphical elements that the user has available. From the left of the graphical element button bar, the graphical elements are square, rectangle, circle, ellipse, polyline, polygon, closed approximated spline, open approximated spline, closed interpolated spline, open interpolated spline, label (text), grid, legend, bar scale, and GeoFrame. The GeoFrame is a key graphical element that provides a mechanism for the user to indicate a rectangular area of the drawing area within which to put spatially referenced elements, such as raster, vector, and sites maps, and grids.

Drawing Graphical Elements

To draw a graphical element the user simply clicks on the appropriate button with the left mouse button and moves into the drawing area. Once the mouse is in the drawing area the following steps are taken:

- Square, Rectangle, Circle, Ellipse, and GeoFrame

  Press the left mouse button to anchor a corner of the element. Drag the mouse and a banding outline of the bounding rectangle (or square for squares and circles) will appear, changing with the mouse movement. Release the mouse to establish the other corner of the element. The element will appear selected. (When a graphical element is selected it will have a small square at each corner and at each mid-point of the sides of the bounding rectangle. See "Moving/Resizing Graphical Elements" on page 4-3).
- Polyline, Polygon, and all Splines

  Click the left mouse button to establish the first point. Move the mouse, and a banding line will appear. Click the left mouse button to establish each intermediate point. When finished, click the middle mouse button and the figure will be complete and will be selected.
- Label

  Click the left mouse button to place the label. The item will appear selected. You must then invoke the **Set Label Attributes...** menu item to enter the text (see page 5).
- Grid

  A grid will be placed over the selected GeoFrame. This means that nothing will happen unless a GeoFrame is drawn and selected. No further interaction is needed.
- Legend and Bar Scale

  Click the left mouse button to place the upper left-hand corner of the element, the element will be drawn and selected. Select the **Set Legend Attributes...** (page 5) or **Set Scale Attributes...** button from the menu system to make further adjustments.

Selecting Graphical Elements

Graphical elements can be selected in one of two ways; both require that the interface is in select mode. This can be accomplished by selecting the **Activate Select Mode** (page 4) menu item from the menu system, or the **Select Mode** (page 8) button from the popup menu. Once in select mode, simply click the left mouse button on the graphical element you want to be selected. Alternatively, press the left mouse button in an area outside of any graphical element and drag a bounding box around the element(s) that you want to select, and then release the mouse button.

Moving/Resizing Graphical Elements

To move a graphical element, you must first be in select mode and have the element(s) that you want to move selected (see above section). To move the element(s) simply press the mouse in the element(s), drag to the desired location, and release the mouse button. To resize an element, you must first be in select mode and have the element that you want to resize selected. Once this is accomplished, press the left mouse button in one of the eight small squares, drag to the appropriate location, and release the mouse button.

Reshaping Graphical Elements

Certain graphical elements can be reshaped. These include polylines, polygons, and all of the splines. To reshape an element, select it, and then invoke the **Activate Poly Reshape Mode...** menu item (see page 4 for more information). The figure will be drawn with square boxes at each vertex. To reshape, simply press the left mouse button in the appropriate box and drag to the desired location. Once you have moved the vertex to the appropriate position, release the mouse button. To complete the reshaping, put the interface back into select mode by selecting the **Activate Select Mode** (page 4) menu item from the menu system, or select the **Select Mode** (page 8) button from the popup menu.

Pulldown Menu

The pulldown menu bar is where most of the non graphic **`xgdisplay`** functionality is located. The structure of the menu bar is given here:

- File
  - Open...

    This button allows the user to open a saved **`xgdisplay`** script. The script describes all objects on the interface the script was saved. It will popup the Open dialog (see “Open Dialog” on page 4-9 for more information).
  - Save...

    This button allows the user to save all of the graphics in the drawing area to a script. It will pop up the Save dialog (see “Save Dialog” on page 4-9) for more information).
  - **Quit**

    This button quits the interface.
- Modes
  - Activate Select Mode

    This button activates the select mode.which allows the user to select graphical elements for further interaction. See “Selecting Graphical Elements” on page 4-3.
  - Activate Poly Reshape Mode

    This button activates the poly reshape mode which allows the user to reshape polylines, polygons, and splines. See “Reshaping Graphical Elements” on page 4-4.
  - Activate Highlight Mode

    This button activates the highlight mode.

4.1 Beta Note:

The **Activate Highlight Mode** menu item is not available in the Beta release.

- Activate Modify Colors Mode

  This button activates the modify colors mode. This mode allows the user to modify the color table of raster maps. It will not become active unless the user has selected a GeoFrame with a raster map in it (see “Selecting Graphical Elements” on page 4-3 and “Raster...” on page 4-6). This menu item will pop up the Modify Colors dialog (“Modify Colors Dialog” on page 4-9).
- Attributes
  - Set Grid Attributes...

    This mode will not become active unless the user has selected a GeoFrame (see “Selecting Graphical Elements” on page 4-3). This menu item will pop up the Grid Attributes Dialog (see “Grid Attributes Dialog” on page 4-10).
  - Set Legend Attributes...

    This mode will not become active unless the user has selected a legend (see “Selecting Graphical Elements” on page 4-3). This menu item will pop up the Legend Attributes Dialog (see “Legend Attributes Dialog” on page 4-11).
  - Set Scale Attributes...

    This mode will not become active unless the user has selected a Bar Scale (see “Selecting Graphical Elements” on page 4-3). This menu item will pop up the Bar Scale Attributes Dialog (see “Scale Attributes Dialog” on page 4-13).
  - Set Label Attributes...

    This mode will not become active unless the user has selected a Label (see “Selecting Graphical Elements” on page 4-3). This menu item will pop up the Label Attributes Dialog (see “Label Attributes Dialog” on page 4-14).
- Tools
  - Graphic Tool Box...

    This menu item will invoke the Graphic Tool Box dialog. Its operation is discussed in “Graphic Tool Box” on page 4-14.
  - Set Region for Selected GeoFrame...

    This menu item will invoke the Modify Region dialog. Its operation is discussed in “Modify the Region Dialog” on page 4-14.
  - Set Highlight Color...

    This menu item invokes the highlight mode.

4.1 Beta Note:

The **Set Highlight Color...** menu item is active in the Beta release, but accomplishes nothing because the **Activate Highlight Mode...** menu item is not available.

- Set Default Site Symbol (Standard)...

  This menu item invokes a standard site selection dialog, providing a mechanism for the selection of the four standard GRASS icon shapes.
- Set Default Site Symbol (Freehand)...

  This command provides a mechanism for making freehand drawn graphical elements the current site icon. Simply select any graphical element(s) you desire to make the site icon and that will be made the current icon.
- Set Default Site Symbol (Pixmap)...

  This menu item invokes a file selection dialog. It provides a mechanism for retrieving an icon generated using the pixmap command as the current sites symbol.

4.1 Beta Note:

Note that none of the site symbols work correctly at the time of the Beta release, due to unforeseen difficulties.

- Maps
  - Raster...

    This menu item invokes a browser allowing the selection of a raster map for display. For more information on its operation, see "xgbrowser" on page 4-19.
  - Delete Raster

    This menu item will delete the current raster map for the selected GeoFrame (see "Selecting Graphical Elements" on page 4-3).
  - Vector...

    This menu item invokes a browser allowing the selection of a vector map for display. For more information on its operation, see "xgbrowser" on page 4-19.
  - Delete Vector...

    This menu item will delete a vector map for the selected GeoFrame (see "Selecting Graphical Elements" on page 4-3). It invokes the Delete Vector dialog (see "Delete Vector and Delete Sites Dialog" on page 4-19).
  - Sites...

    This menu item invokes a browser allowing the selection of a sites map for display. For more information on its operation, see "xgbrowser" on page 4-19.
  - Delete Sites...

    This menu item will delete the current raster map for the selected GeoFrame (see "Selecting Graphical Elements" on page 4-3). It invokes the Delete Sites Dialog (see "Delete Vector and Delete Sites Dialog" on page 4-19).
- Options
  - Set Position Readout Units To Pixels

This menu item will cause the position readouts displayed in the information panel at the bottom of the interface to display screen units instead of inches or millimeters. If the rulers were displaying millimeters previously, they will change to display inches.

- Set Position Readout Units to Inches

  This menu item will cause the position readouts displayed in the information panel at the bottom of the interface to display inches instead of pixels or millimeters. If the rulers were displaying millimeters previously, they will change to display inches.

- Set Position Readout Units to Millimeters

  This menu item will cause the position readouts displayed in the information panel at the bottom of the interface to display millimeters instead of pixels or inches. If the rulers were displaying inches previously, they will change to display millimeters.

- Set One Vector file

4.1 Beta Note: THIS MENU ITEM WAS MISTAKENLY LEFT IN THE INTERFACE!

- Set One Sites file

4.1 Beta Note: THIS MENU ITEM WAS MISTAKENLY LEFT IN THE INTERFACE!

- Activate Query Raster Mode

  This menu item will popup the Query Raster Dialog and places the interface into Query Raster Mode. There must be a selected GeoFrame with a raster map displayed in it for this option to operate. See “Selecting Graphical Elements” on page 4-3 and “Query Raster Dialog” on page 4-19 for more information.

- Activate Query Vector Mode

4.1 Beta Note: This menu item is not active in the Beta version of this software.

- Help
  - General...
  - Current Mode...
  - Context

4.1 Beta Note: The items in the Help menu are not active in the Beta version of the interface.

Drawing Area

The drawing area is a scrollable area that is the location of all actual drawing activity. It consists of the drawing area itself, vertical and horizontal rulers, and vertical and horizontal scrollbars. Items are drawn in the drawing area by selecting the intended object to draw and then sizing and placing them according to the instructions in "Graphical Element Buttons" on page 4-2.

Popup Menu

To provide rapid access to several commonly used functions, we have created a popup menu. This menu is activated by pressing the right mouse button in the drawing area, dragging to the intended menu selection, and releasing the mouse. Alternatively, you may click the right mouse button, and then click again on the intended menu selection. The following items are available from the popup menu:

- Select Mode

  This menu item will place the interface in Select mode (see page 4).
- Front

  Selecting this menu item will bring selected graphical elements to the front of a stack of elements. See "Selecting Graphical Elements" on page 4-3.
- Back

  This menu item will put selected graphical elements to the back of a stack of graphical elements. See "Selecting Graphical Elements" on page 4-3.
- Delete

  Selecting this item will delete all selected items. See "Selecting Graphical Elements" on page 4-3
- Refresh

  This menu item, when activated, will redraw the entire drawing area.

Information Panel

The Information Panel has three components: a horizontal mouse position readout, a vertical mouse position readout, and a display of the currently active interface mode. The position readout indicates the mouse position in one of three units:

1. Pixels—this readout unit reports the mouse position in terms of the number of screen pixels with (0,0) in the upper left-hand corner.
2. Inches—this readout unit report the mouse position in inches measured from the upper left-hand corner.

3. Millimeters—this readout unit measures the mouse position in millimeters measured from the upper left-hand corner. This readout mode is activated using the pulldown menu item described in “Set Position Readout Units to Millimeters” on page 4-7.

Open Dialog

The Open Dialog is invoked whenever the user desires to load a saved **`xgdisplay`** script. The dialog that is popped up is a browser. For more information on its operation, see “xgbrowser” on page 4-19.

Save Dialog

The Save Dialog is invoked whenever the user wants to save the graphical elements displayed in the current configuration of the interface. The dialog is depicted in Figure 9. Simply type a name that can be recognized later into the text field and press the “OK” button. Press “Cancel” to stop this action.

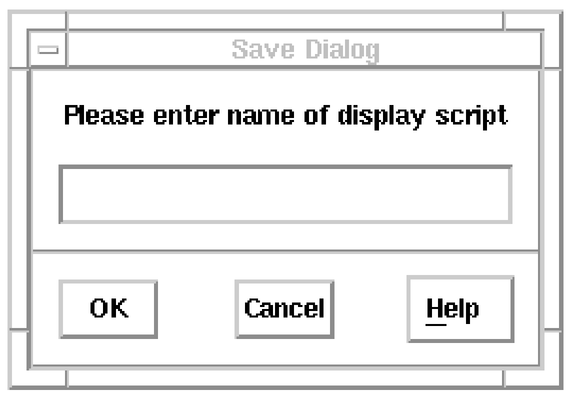

Figure 9 The Save Dialog.

Modify Colors Dialog

This dialog is invoked when the interface is placed in Modify Colors Mode (see Figure 9); the user must have selected a GeoFrame with a raster map displayed in it for this mode to become active. The dialog is depicted in Figure 10. The components of this dialog are:

1. a matrix of buttons that are colored as the colors of the raster map in the selected GeoFrame, which when selected becomes the color currently being modified;
2. a label that displays the name of the raster map, the category number of the currently modified category, and the category description;
3. a single color editing dialog (usage of this interface is described in “Color Editing Dialog” on page 4-17); and

4. a row of buttons.

To edit a color, select a category to modify by clicking the left mouse button on the desired button in the matrix of buttons, and by manipulating the sliders to reach the appropriate color. To make all changes permanent, i.e., to write the color table into the GRASS database, click the left mouse button on the "OK" button. To cancel all changes, click the left mouse button on the "Cancel" button.

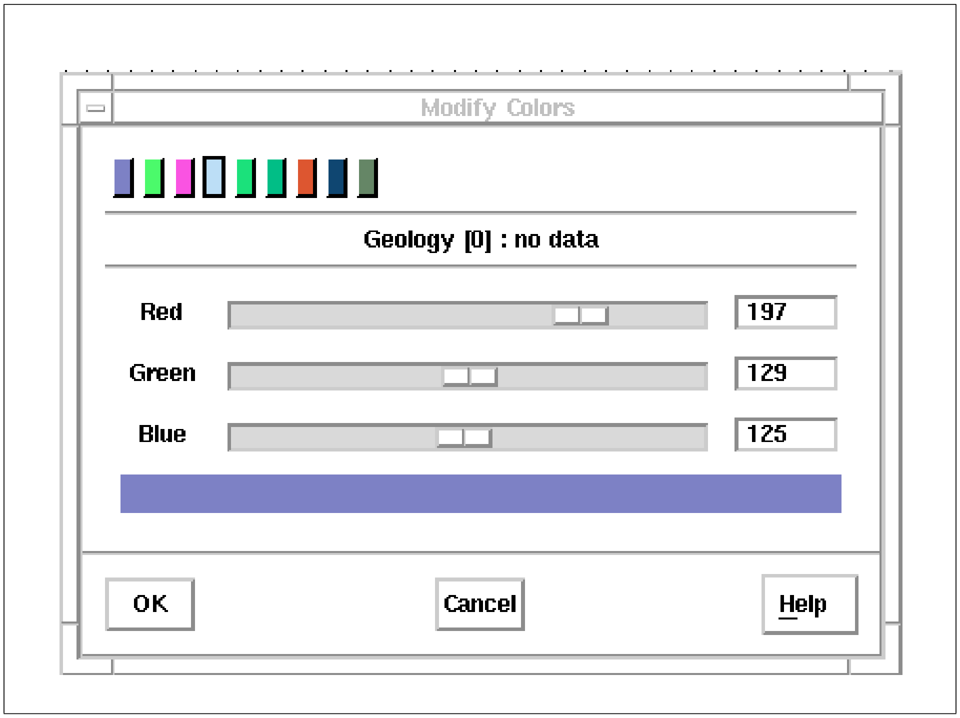

Figure 10 The Modify Colors Dialog.

Grid Attributes Dialog

This dialog is invoked whenever the user desires to change the appearance of a grid overlaid in a GeoFrame. The user need not have a selected GeoFrame and a grid displayed for this dialog to be invoked, although changes will not be evident until a GeoFrame is selected and a grid is displayed using the Grid graphical element button. The dialog is depicted in Figure 11. The components of the dialog are as follows:

1. a pair of toggle buttons for toggling the grid on and off and for toggling the display of grid labels;
2. a pair of increment boxes, one for setting the gap between grid lines (measured in meters) and for setting the number of grid lines between labels;
3. a line pattern interface;
4. another increment box for setting the width of the grid lines;

5. a color option menu for selecting the color of the grid lines;
6. a color option menu for selecting the color of the label text;
7. a button that invokes a font dialog; and
8. a “Dismiss” button for dismissing the dialog along with the usual help menu.

To make changes to a grid, simply manipulate the items in the dialog until the desired appearance is reached. If a GeoFrame with a grid displayed is selected, the changes will occur interactively, otherwise the changes will be active when the next grid is displayed.

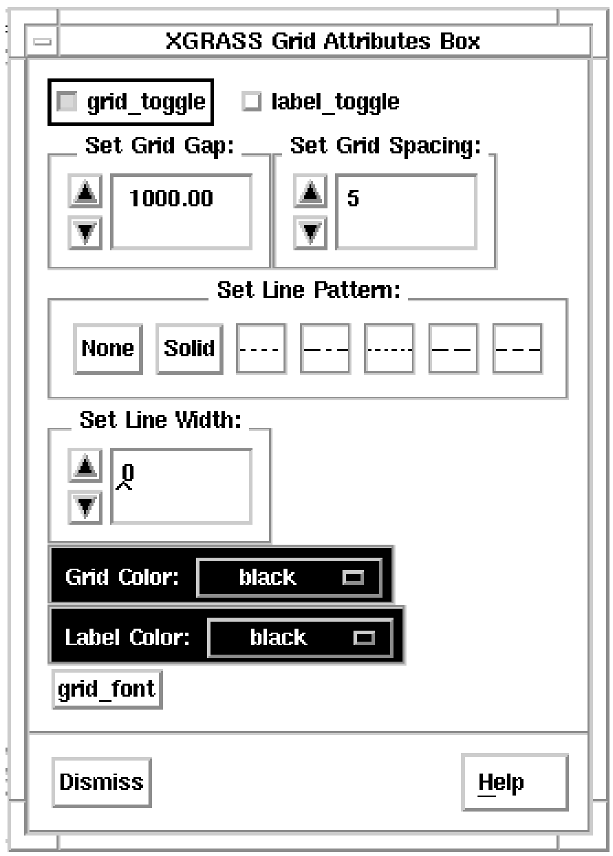

Figure 11 The Grid Attributes Dialog.

Legend Attributes Dialog

This dialog is invoked whenever the user desires to change the appearance of a legend connected to a GeoFrame. The user must have a selected legend displayed for this dialog to be invoked. The dialog is depicted in Figure 12. The components of the dialog are as follows:

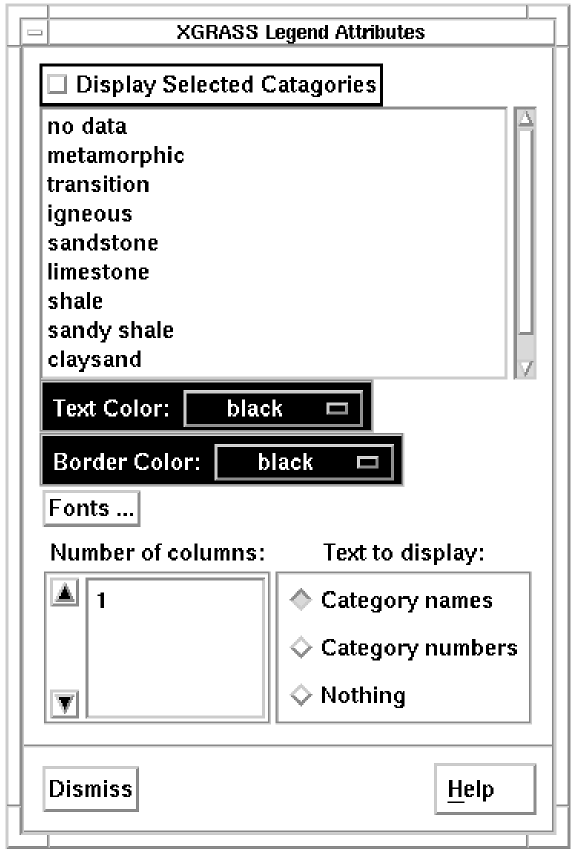

Figure 12 Legend Attributes Dialog.

1. a toggle button for toggling the display of all categories or selected categories;
2. a list allowing the selection of the categories to display;
3. a color option menu for selecting the color of the legend text;
4. a color option menu for selecting the color of the legend color box outline;
5. a button that invokes a font dialog;
6. an incrementor for the number of columns to use for the display of the legend,
7. a set of toggle buttons for selecting whether to display the category name, category numbers, or nothing next to the colors in the legend; and
8. a “Dismiss” button for dismissing the dialog along with the usual help menu.

To make changes to a legend simply manipulate the items in the dialog until the desired appearance is reached.

Scale Attributes Dialog

This dialog is invoked whenever the user wants to change the appearance of a bar scale. A bar scale object must already be placed within the drawing area. The dialog (depicted in Figure 13) contains the following:

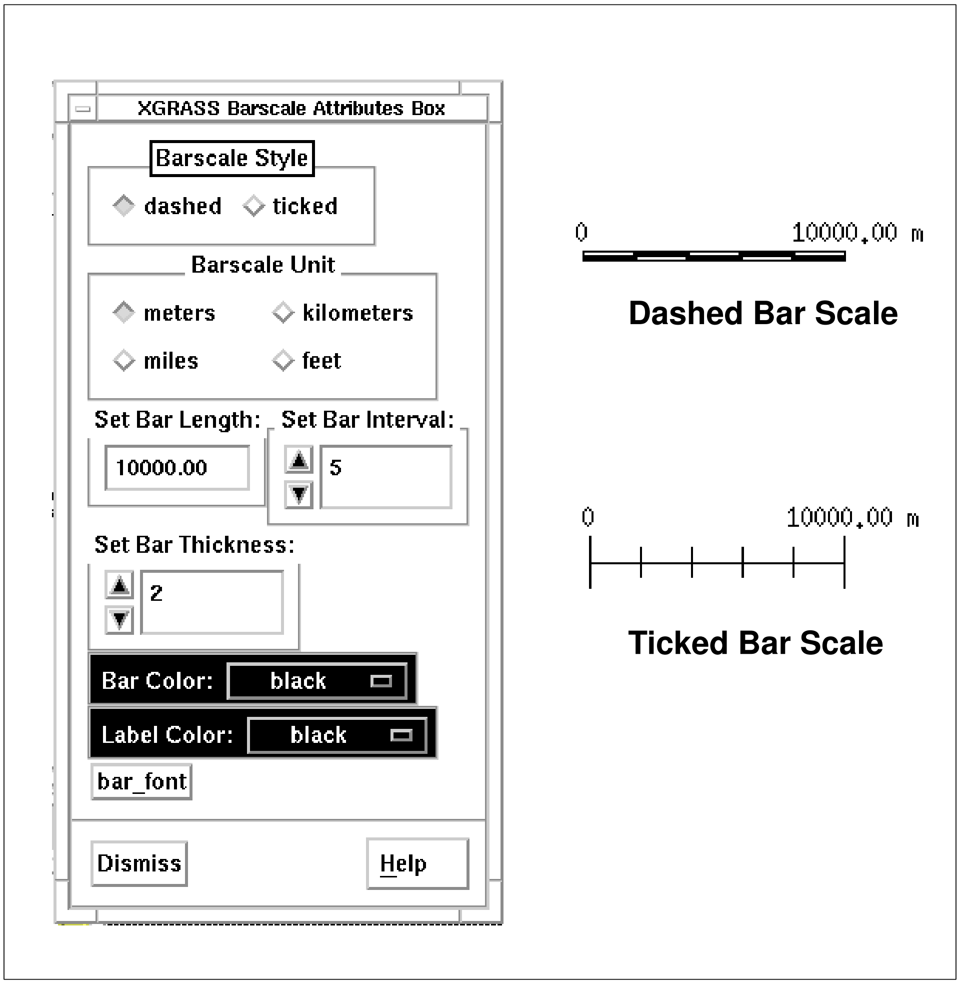

Figure 13 The Bar Scale Attributes Dialog and examples of bar scale styles.

1. a pair of toggle buttons which determine the style of the bar scale (dashed or ticked, see Figure 13 for examples of these styles);
2. a set of four toggle buttons which determine the units of the bar scale;
3. an incrementor interface that determines the length (in the current units) of the bar scale);
4. an incrementor interface the determines the number of intervals in the bar scale;
5. an incrementor that determines the thickness of the bar scale;
6. a color option menu that determines the color of the bar scale;

7. a color option menu that determines the color of the bar scale text, and a button that invokes a font dialog; and
8. the usual “Dismiss” button and “Help” menu.

To make changes to a bar scale simply manipulate the items in the dialog until the desired appearance is reached.

Label Attributes Dialog

This dialog is invoked whenever the user wishes to add text to the drawing area. A label object must already be placed within the drawing area before the label attributes dialog can be used. The dialog contains a scrolled text entry area, a text color option menu and a button that will activate a font selection dialog. To change the text of the label; simply type the desired text into the scrolled text entry area.

Graphic Tool Box

This dialog is invoked whenever the user desires to change the appearance of a graphical element other than a label, a grid, a legend, or a bar scale. The dialog (depicted in Figure 14) contains the following:

1. a row of buttons that determine the fill pattern of the object,
2. a row of buttons that determines the line pattern for the object,
3. an incrementor that determines the line thickness,
4. a color option menu determining the foreground color of the object,
5. a color option menu determining the background color of the object, and
6. the usual “Dismiss” button and “Help” menu.

To change the appearance of an object, simply select the object and interact with the dialog until the desired appearance is reached. This dialog also determines the current values for all of these graphical attributes for graphical elements, meaning that the next graphical element displayed will inherit the values shown in the Graphic Tool Box.

Modify the Region Dialog

This dialog is invoked whenever the user desires to change the region associated with a GeoFrame. The dialog is depicted in Figure 15 and contains the following elements:

1. six text entry fields associated with the current north, south, east, and west edges of the region being modified, as well as the east-west and north-south resolution values;
2. a “drawing” area where the white area represents the current region;
3. an “Align/Grid...” button that, when selected, invokes the align/grid options dialog (depicted in and described below);
4. a “Saved...” button that, when selected, invokes a browser interface allowing the selection of a previously saved region;

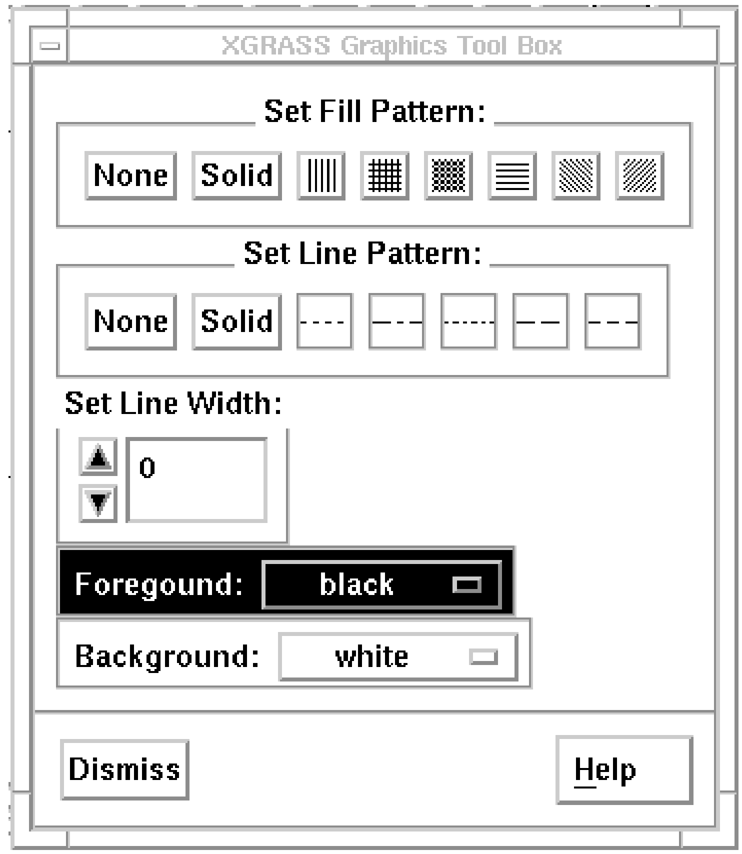

Figure 14 The Graphic Tool Box dialog.

5. a “Default Region” button that, when selected, sets the region to the default region;
6. a “Save Region” button and text entry area, that allows the user to enter a name (specifying the name of a region to save) and click the button thus saving a named region; and
7. the usual “OK” button, “Cancel” button, and “Help” menu.

The user may change any of the four edge parameters by typing a new value into the four text entry fields associated with the edge parameters. This has the effect of setting the corresponding edge to the entered value; the new region is displayed within the drawing area to the right. Alternatively, the user may “draw” the desired region by pressing the left mouse button in the drawing area to establish one corner of the new region, then dragging the mouse to sweep out the region and releasing the mouse button to complete the definition. The fields in the four edge parameter text entry fields will change appropriately. The user may change the north-south or east-west resolution by entering the new value into the appropriate text entry field.

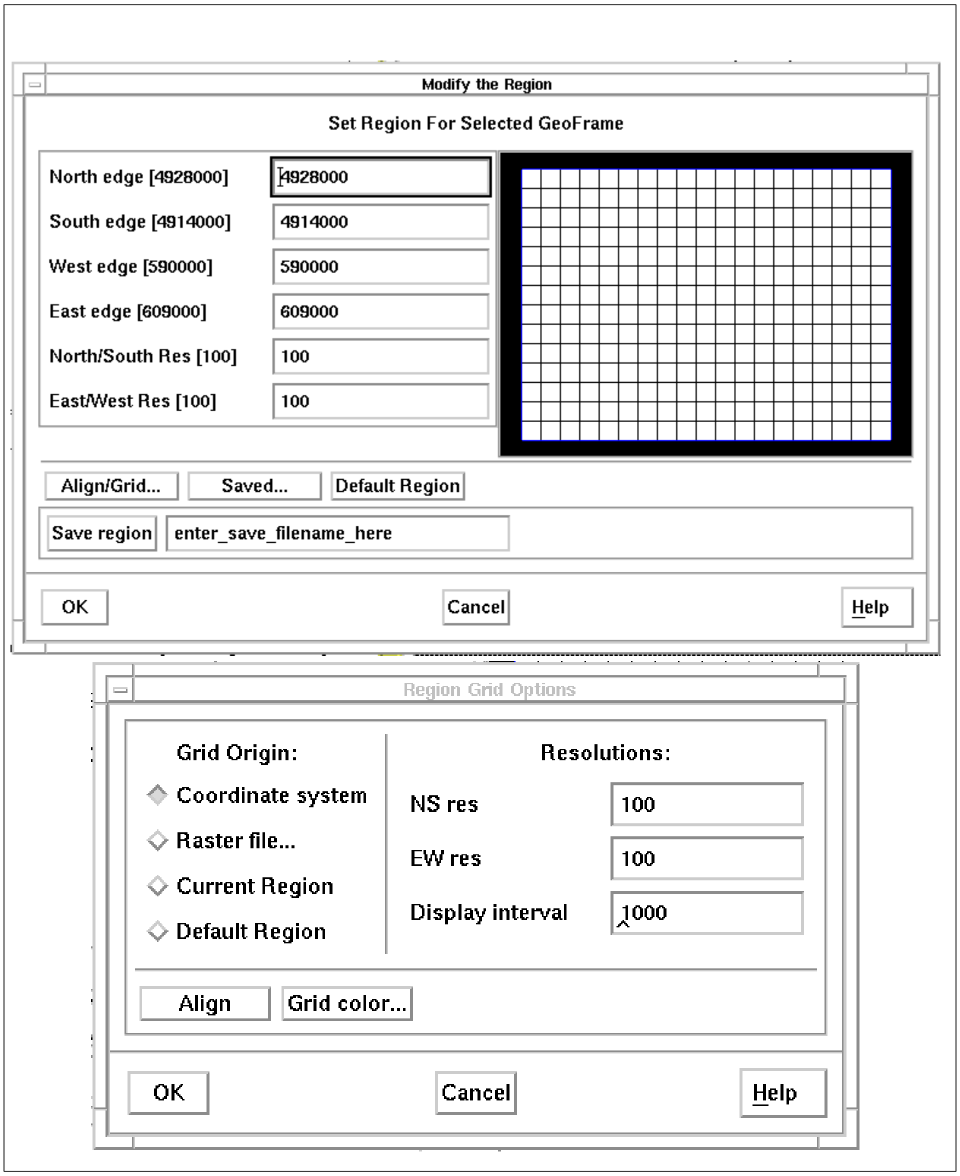

Figure 15 The Modify Region Dialog, along with the Grid Options sub-dialog.

In many cases, the user might desire to align the current region with some coordinate system, raster map, the current region boundary, or the default region boundary. This is accomplished by clicking on the "Align/Grid..." button. When selected, the "Align/Grid Options" dialog is invoked (see Figure 15). This dialog provides a series of four toggle buttons that, when selected, will align the current region (defined by modifying the edge fields or by rubber-banding a region in the main dialog) to the appropriate grid origin. These origins are described here.

1. Coordinate system—this button will align the current region to the coordinate system of the current location (typically UTM or Lat-Long) and the resolu-

tions to the right of the toggle buttons. For example, if the user sweeps out a new region and clicks on the "Coordinate system" toggle button with the "NS res" and "EW res" text fields set at 100, then when the user selects the "Align" button, the region would be aligned with a multiple of 100 in both the north-south and east-west directions. Thus the edge fields will be rounded to the nearest 100 meters.

2. Raster file...—as indicated by the ellipsis, activating this toggle button invokes a browser dialog. The user must select a raster map with which to align the current region. Once a raster map has been selected, the "NS res" and "EW res" text entry fields will be replaced by the resolution of the map. When the user selects the "Align" button, the region will be rounded to the nearest north-south and east-west multiples of the resolution with respect to the "origin" of the raster map.
3. Current Region—activating this toggle button will make the alignment origin the current region (the region as it was when the Modify Region dialog was invoked). The alignment occurs as stated in the Coordinate system case.
4. Default Region—activating this toggle button will make the alignment origin the default region. The alignment occurs as stated in the Coordinate system case.

Modifying the value of the "NS res" and "EW res" fields will affect the alignment process by making it round to the nearest multiple of the resolution in that direction with respect to the origin specified. Modifying the "Display Interval" field will set the displayed grid to the interval entered. These fields do not take effect until a return key is pressed. The user may modify the color of the grid displayed in the main region dialog by pressing the "Grid color...." button and interacting with the single color editing dialog that appears (see "Color Editing Dialog" on page 4-17 for more information).

To make the region change take affect, the user must select the "OK" button on the main dialog, selecting "Cancel" will cancel the region change.

Color Editing Dialog

This dialog (sometimes incorporated as part of a larger dialog) allows the user to modify a color by adjusting the red, green, and blue components of a color. The interface has three combinations of components, one for red, one for green, and one for blue. These combinations are made up of a label specifying which component is being edited, a slider that when moved affects the value of that color component, and a text entry field that displays the current value of that component (and may be modified directly). When any one of the three components color values change, a bar just below the sliders will change to display the color as it currently exists.

The "OK" button will make the change to the color, the "Cancel" button will revert the color back to the value it had when the dialog was invoked.

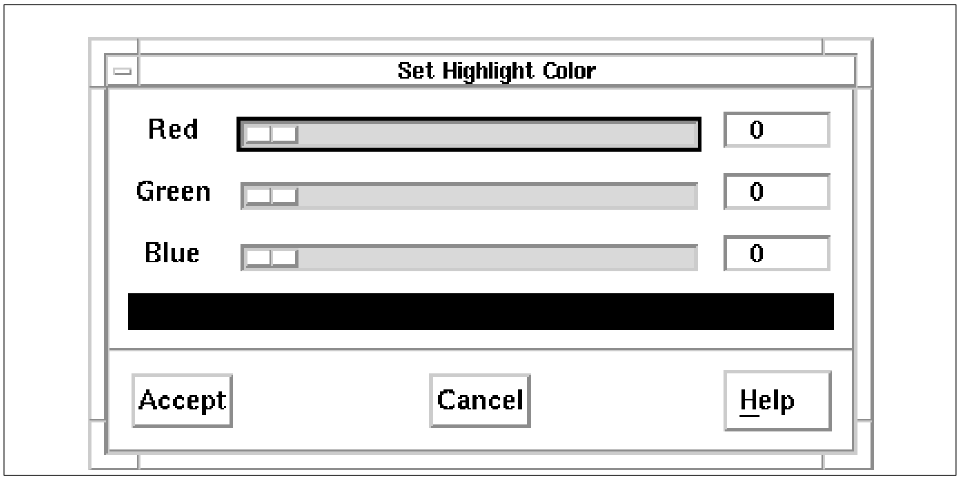

Figure 16 The color editing dialog.

Font Dialog

This dialog is invoked from several other dialogs. It provides a mechanism for the selection of X Window System fonts. The dialog is shown in Figure 17. The interface consists of a scrolling list of fonts, a text entry field, an “OK” button, a “Cancel” button and a “Help” menu. The user simply selects on of the items from the list and clicks the left mouse button on the “OK” button to select the font. Alternatively, the user may type the name of the font into the text entry area and press the “OK” button. Pressing the “Cancel” button will dismiss the dialog with no changes.

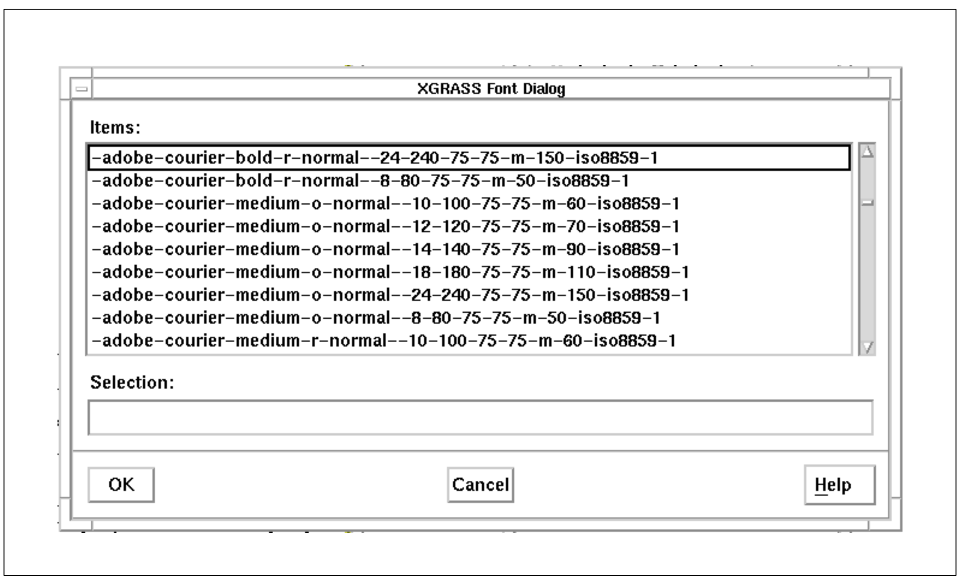

Figure 17 The Font Dialog.

Query Raster Dialog

This dialog is invoked whenever the user desires to determine specific information concerning a displayed raster map.To invoke this dialog, a GeoFrame with a raster map displayed in it must be selected. The dialog that appears is depicted in Figure 18. Simply click the left mouse button in the selected GeoFrame and the values will be filled in the dialog. Press the "Dismiss" button to dismiss the dialog and revert to select mode.

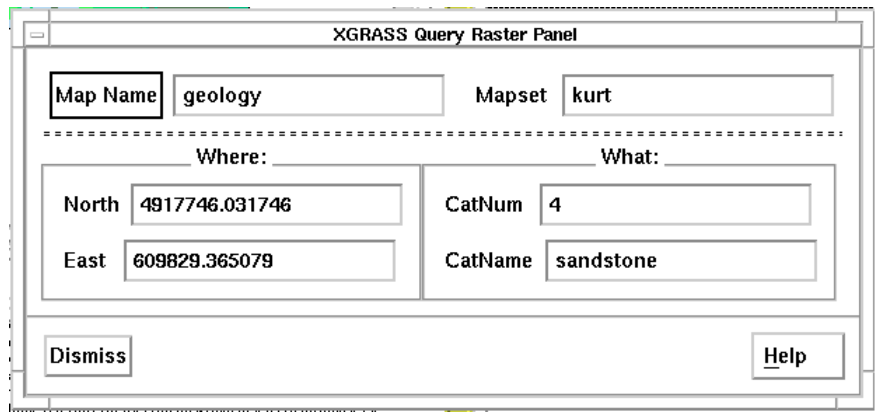

Figure 18 The Query Raster Dialog.

Delete Vector and Delete Sites Dialog

These dialogs are invoked whenever the user needs to delete a vector (or sites) map from a GeoFrame. Obviously, a selected GeoFrame containing a vector (or sites) map must exist before this dialog can be activated. The dialog appears in Figure 19. To delete a vector (or sites) map, simply select the desired name from the list and press the "OK" button. Press the "Cancel" button to cancel the delete operation.

xgbrowser

`xgbrowser` is a generic tool that supports browsing and selection of GRASS database elements. A typical browser configuration is displayed in Figure 20. The **`xgbrowser`** program can be configured to have one, two, or three selection panels. In each of these panels the top portion contains a Mapset option menu. When the user presses the button on the option menu a menu will pop up. The user can then select the mapset from this menu. The contents of the mapset menu are those mapsets in the current mapset search path. The low portion of the **`xgbrowser`** is a scrollable list that the user can pick from to select a database element. **`xgbrowser`** can be configured to support single selections (one database

Figure 19 The Delete Vector Dialog.

element at once) or multiple selections (more than one database element). **xg-browser** will output the name and mapset (name@mapset) for the item selected. If the list is configured for multiple selections, **xgbrowser** will output a comma-separated list of name-mapset combinations.

Command-Line Usage

xgbrowser [options]

Where the options are:

-numLists n (1, 2 or 3)	Sets the number of panels in the browser
-list[123]IsStatic	Enables the ability of the user to change the mapset for the panel indicated, this is the default.
+list[123]IsStatic	Disables the ability of the user to change the mapset for the panel indicated.
-initialMapset[123] name	Sets the initial mapset of the given browser panel to name, otherwise the mapset displayed will be the first one in the mapset search list.
-promptLabelString prompt	Sets the prompt label to the string given by prompt. If you want a multi-word prompt enclose it in double quotes.

and one of:

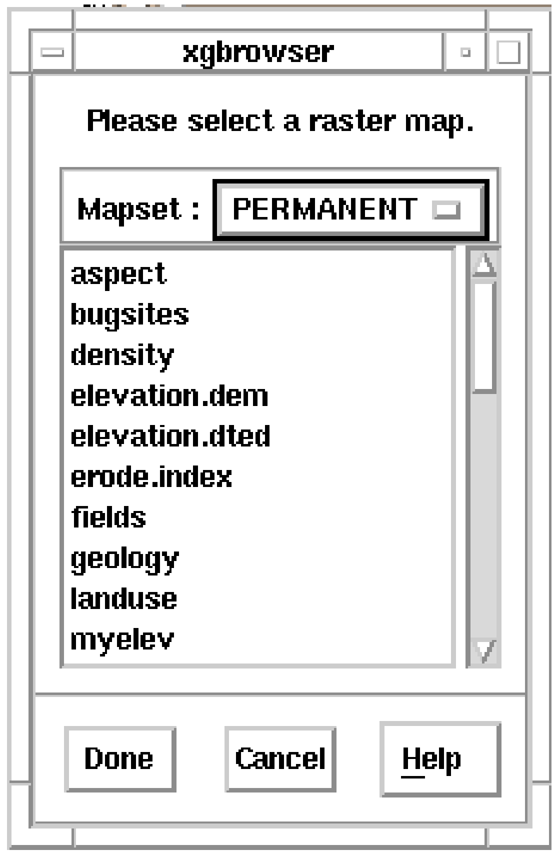

Figure 20 Example **`xgbrowser`** showing a single panel.

-raster	Indicates the raster database element.
-vector	Indicates the vector database element.
-site	Indicates the sites database element.
-region	Indicates the region database element.
-icon	Indicates the icon database element.
-label	Indicates the label database element.
-group	Indicates the group database element.
-asciidlg	Indicates the ASCII dlg database element.
-dlg	Indicates the binary dlg database element.
-asciivector	Indicates the dig database element.
-segment	Indicates the segment database element.
-user_defined db_element	Indicates the database element given by db_element.

and one of:

-single	Enable single selection mode.
-multiple	Enables multiple selection mode.

xgregion

xgregion is a generic tool that supports the manipulation of the GRASS region. The **xgregion** program can be configured to change the default region or the current (active) region. The interface is fully described in "Modify the Region Dialog" on page 4-14 and is depicted in Figure 15.

Command-Line Usage

xgregion [options]

Where the options are:

-gridColor colorname	Sets the color of the grid to the given color.
-grid	Turns the grid display OFF.
+grid	Turns the grid display ON.
-promptLabelString string	Sets the prompt label to the given string.

One of:

-current	Will modify the current region.
-default	Will modify the default region.

xgaccess

xgaccess is a tool that supports the specification of the permissions on a mapset. The mapset affected is the current mapset. The interface (depicted in Figure 21) is extremely simple. It consists of two toggle buttons, which, if selected, allow access to the mapset for group members or everyone. There are no command-line options needed with this command.

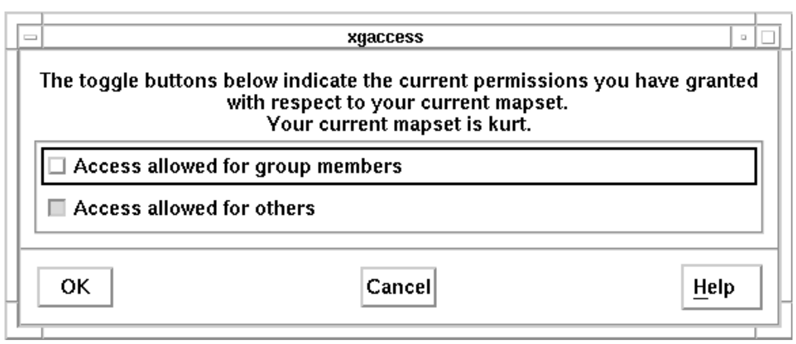

Figure 21 The **xgaccess** interface.

xgmapsets

xgmapsets is a tool that provides the mechanism for a user to set their mapset search path. This is the set of mapsets that are searched for database elements when a mapset is not specifically mentioned. The interface, depicted in Figure 22, consists of two scrolling lists and two buttons. The list on the left is the list of all available mapsets for the current location. The list on the right contains the mapsets in the current mapset search path. To add a mapset to the mapset search path, select one of the mapsets from the list on the left and press the “Add” button. To remove a mapset from the mapset search path, select a mapset from the list on the right and press the “Remove” button. To activate the change, press the “OK” button, to cancel press the “Cancel” button. This command has no command-line arguments.

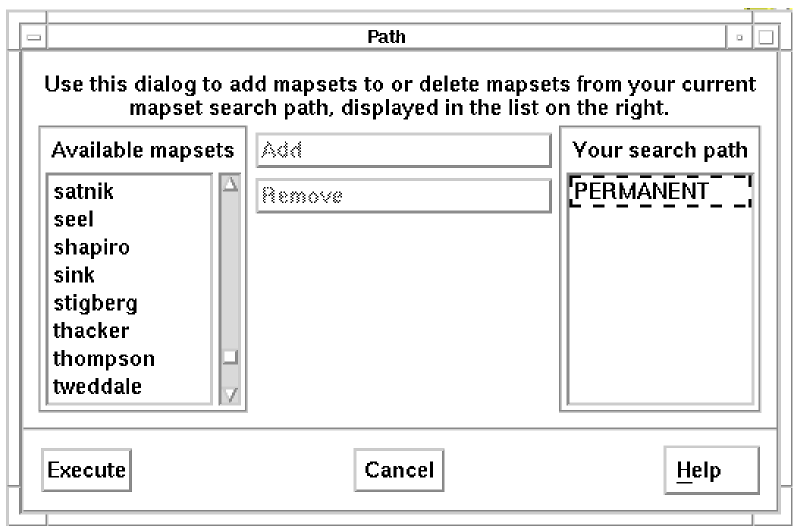

Figure 22 The **xgmapsets** interface.

xgmapcalc

xgmapcalc is an interface to the GRASS program **r.mapcalc**. The interface is depicted in Figure 23, and has the following major elements:

1. a text entry area for the name of the resultant map;
2. a large scrolling text entry area for the **r.mapcalc** expression;
3. a matrix of buttons corresponding to the syntactic elements of the **r.mapcalc** language;
4. a raster browser for the selection of maps to put into the expression; and

5. an "Execute" button, a "Cancel" button, and the usual "Help" menu.

To form an expression, simply click on the syntactic elements to add into the expression and they will appear in the expression area. To insert a map name into the expression, position the cursor in the expression text area, select the desired map, and press the "Insert" button. To run the expression, press the "Execute" button. To cancel and exit the interface, press the "Cancel" button. The interface does not check the validity of the expression, but it does provide some aids to the construction of valid expressions.

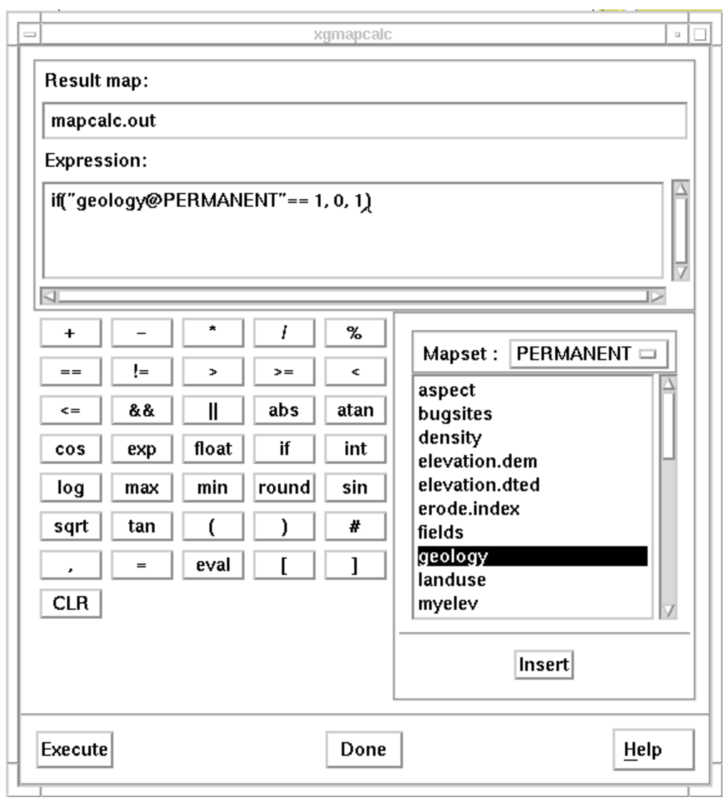

Figure 23 The **`xgmapcalc`** interface.

xgreclass

`xgreclass` is a tool that supports reclass on raster maps. The interface is depicted in Figure 24, and consists of the following elements:

1. a button that when selected invokes a raster map browser and a text entry area, either of which can be used to indicate the raster map to reclassify;

2. a text entry field that is used for the title field in the raster map header file for the newly reclassified map;
3. a text entry field that is used for the name of the new map;
4. a scrolling list containing the categories of the original map;
5. an incrementor containing the current new map category number;
6. a text entry field for the new map category label;
7. a pair of buttons for grouping or ungrouping selected categories (described in the next paragraph);
8. a scrolling list of newly created categories; and
9. the usual “Execute” button, “Cancel” button, and “Help” menu.

To reclassify an existing map to create a new map, the user must first indicate which map to reclassify. This is done by pressing the raster icon button in the area labeled “Original Map:”, which invokes a raster map browser, and selecting the appropriate map (as described in “xgbrowser” on page 4-19), or by typing the name of the map into the text entry field immediately adjacent to the raster icon button. The user must also type in the name of the map to be created in the text entry area labeled “New Map:”. The “New Map Title” field is automatically filled in by xgreclass, but can be changed by typing the title information into the text field. Once the name of the original map has been determined, the category list labeled “Original Categories:” will be filled in.

Simply select the original categories to group into a new category, set the new category number and name, and press the “Group Selected Categories” button. These actions will remove the selected categories from the original category list and make an entry in the new category list corresponding to the category number and name entered. Continue this process until the desired categories have been reclassified. To ungroup a new category (to view the categories involved, for example) simply select an entry from the new category list and press the “Ungroup Selected Category” button. These actions will remove the category from the new list and place the category number in the incrementor, the category name in the name text entry, and display the original categories as selected in the original category list. This situation, provided no change is made to the state of the interface, will allow the category to be re-grouped by pressing the group button, or, changes can be made to reflect the desired state.

Command-Line Usage

`xgreclass` [options]

Where the options are:

-map mapname	Sets the input map to mapname.
-newMap mapname	Sets the output map to mapname.

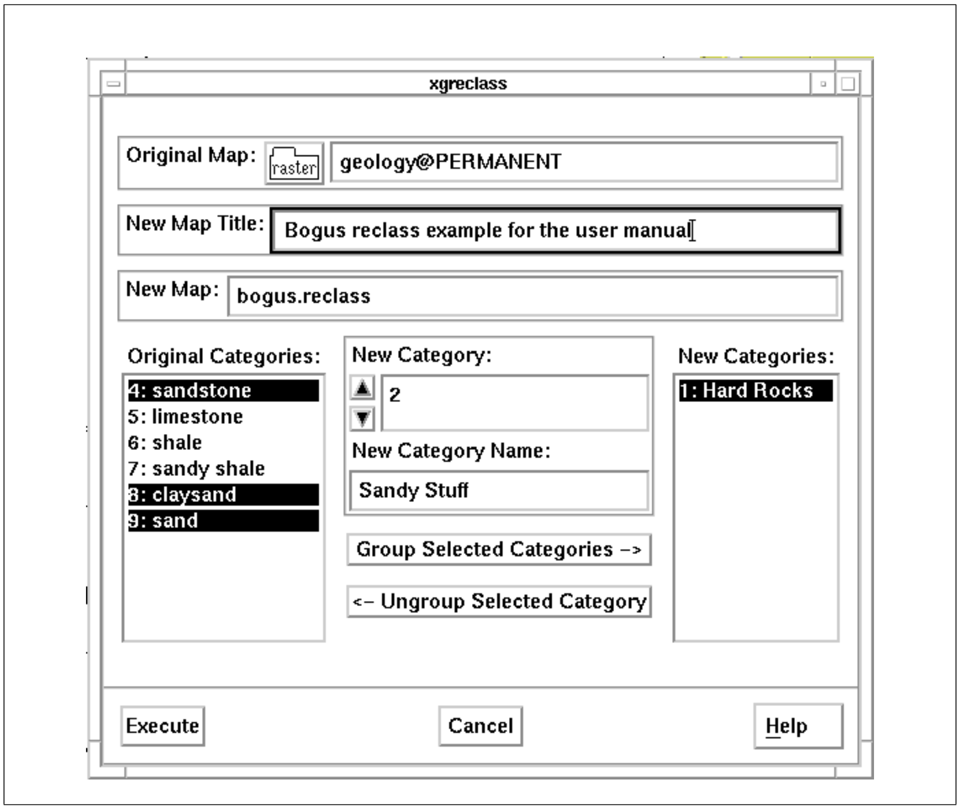

Figure 24 The **`xgreclass`** interface.

xgbuffer

`xgbuffer` is a tool that supports the creation of buffer zones by interfacing with the GRASS command **`r.buffer`**. The interface is depicted in Figure 25. The interface has the following components:

1. a button that when selected invokes a raster map browser and a text entry area, either of which can be used to indicate the raster map from which to generate the buffer zones;
2. a text entry field for the output map name;
3. a set of four toggle buttons for the units of measurement;
4. a scrolling table of buffer zone distances;
5. an “Accept” and “Reset” button used in conjunction with the buffer zone table;
6. a text field showing the **`r.buffer`** command-line; and
7. the usual “OK”, button, “Cancel” button, and “Help” menu.

To create a buffer zone map, the user must first indicate the map to use in the creation of buffer zones. This is done by pressing the raster icon button in the area labeled “Input Map:”, which invokes a raster map browser, and selecting the appropriate map (as described in “xgbrowser” on page 4-19), or by typing the name of the map into the text entry field immediately adjacent to the raster icon button. The user must also type in the name of the map to be created in the text entry area labeled “Output Map:”. Once these map names have been specified the user must input information concerning the specific buffer zones to calculate. The units of measurement are selected by clicking on the appropriate toggle button. Next the buffer zones are entered into the scrolling table. The number entered corresponds to the distance from a non-zero category in the input map. For example, an entry in zone 1 of 100 and an entry in zone 2 of 200 would create a map with four values: the first would include those grid cells corresponding with the non-zero cells, the second would include those cells within 100 meters of the non-zero cells, the third would be all cells within 200 meters of the non-zero cells, and the fourth would be all cells greater than 200 meters from non-zero cells in the input map (see the **r.buffer** manual entry for more details).

Note that for changes or new entries to become noticed in the text fields or table of the **xgbuffer** interface to become noticed, the user must hit the return key.

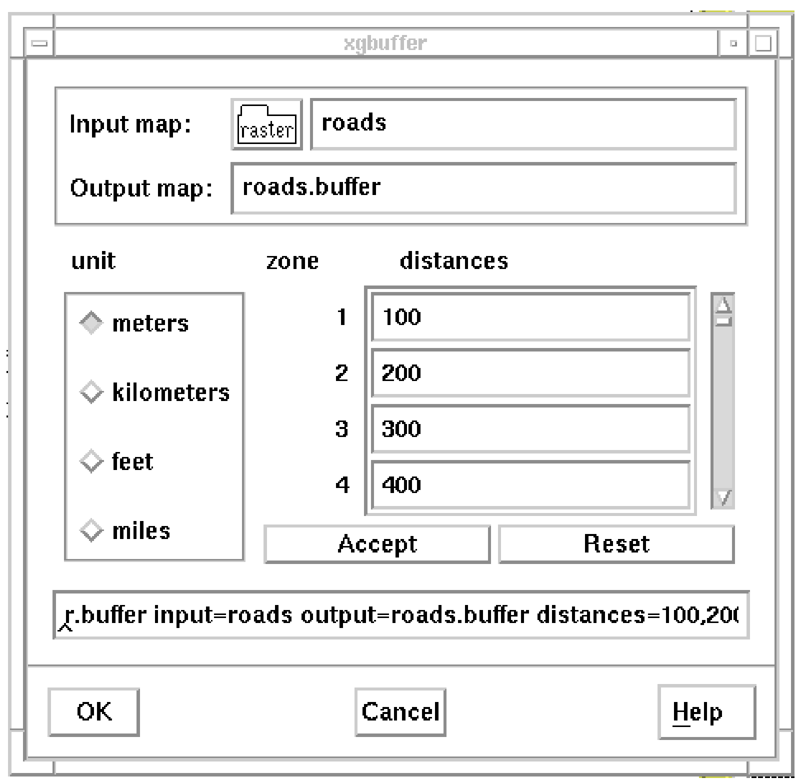

Figure 25 The **xgbuffer** interface.

xgsupport

`xgsupport` provides for the raster support mechanism which enables the user to update specific information associated with raster maps, such as the region associated with the map, its resolution, UTM zone (if appropriate), its history, category file, etc. (see the **`r.support`** manual page for complete details). The interface is made up of a main interface dialog and several sub-dialogs. The main dialog is depicted in Figure 26.

To run the supporting operations on a raster map the user must first indicate which map to use. This can be accomplished by pressing the raster icon button, which invokes a raster map browser, and selecting the appropriate map (as described in “xgbrowser” on page 4-19), or by typing the name of the map into the text entry field immediately adjacent to the raster icon button. Once this is done, the user must select the operation desired. The currently available operations are described below.

- Edit Header...—this operation leads the user through several dialogs. The first (depicted in the center of Figure 26) requests the number of row, columns, and bytes per cell for the actual map data. Once this information has been entered, information about the geographic region represented by the map is requested (see the last panel in Figure 26). After this information has been entered, the user is presented with the resulting information and asked for acknowledgment of its correctness (not shown). At any time during this operation the user may cancel by clicking on the “Cancel” button.
- Edit Categories...—this operation allows the user to edit information concerning the categories represented by the raster map. The dialog depicted in Figure 27 is invoked for this purpose. This interface allows the user to edit the title of the map in the first text entry field. A scrollable list of categories is presented along with an incrementor for the category value, and a text entry field for the category name. To edit a category number or name the user must select the category to edit. Once selected the value and name will be displayed in the appropriate field and can be edited. To add a new category, the user must fill in the value and name fields and press the “Add category” button. To affect the changes the user should click on the “Execute” button, to cancel the entire operation, click on the “Cancel” button.
- Edit History...—this operation allows the user to edit the history file for the raster map. The dialog presented to the user is shown in Figure 28. The user simply edits the field corresponding to the field in the history file to be changed. Tto execute the changes, simply press the “execute” button, to cancel the changes click on the “cancel” button.
- Update Histogram/Range—this operation has no interaction and simply updates histogram and range information on the raster map.

To dismiss the xgsupport interface, click on the “Done” button.

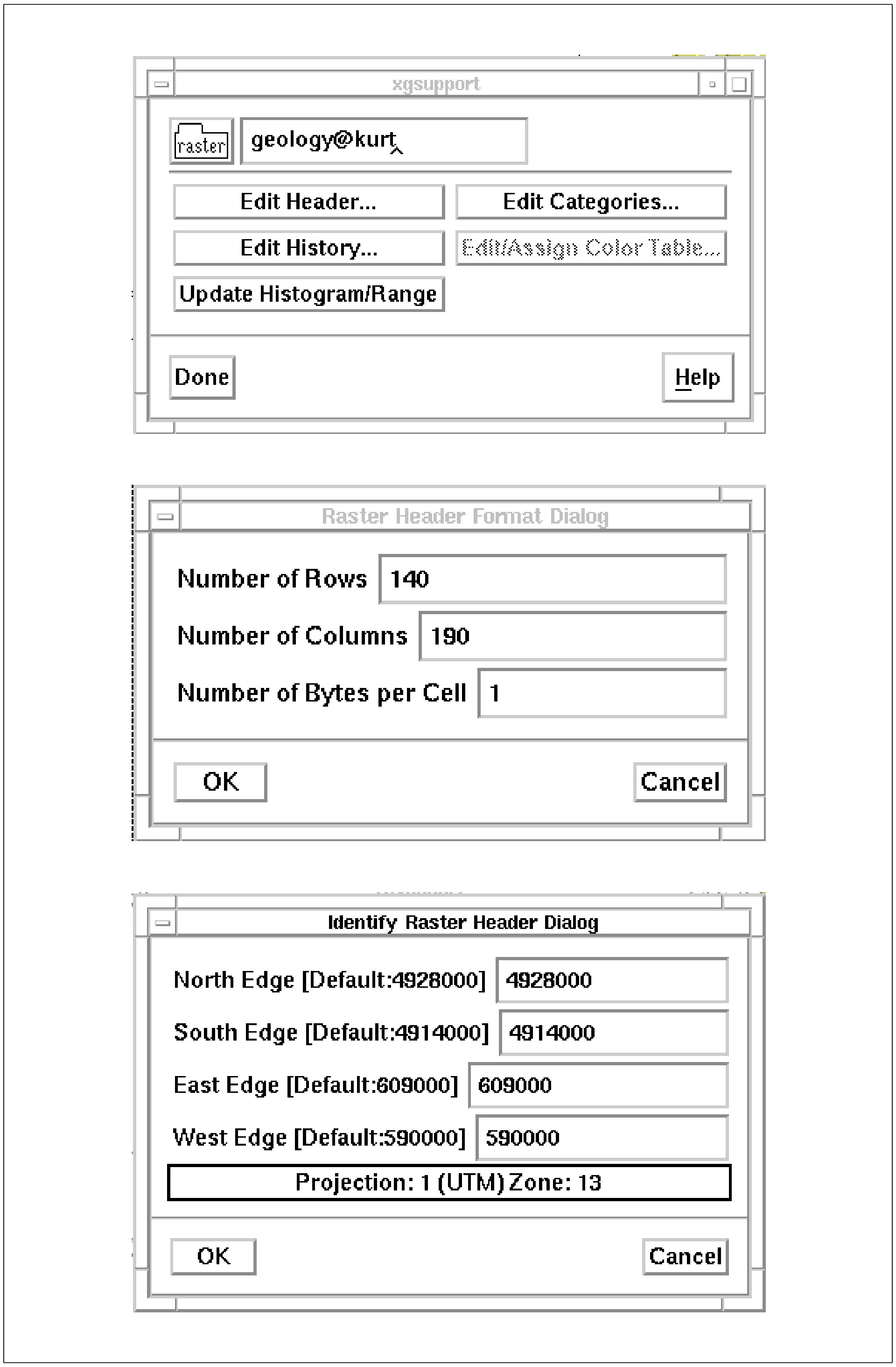

Figure 26 The **`xgsupport`** main panel, and associated sub-dialogs.

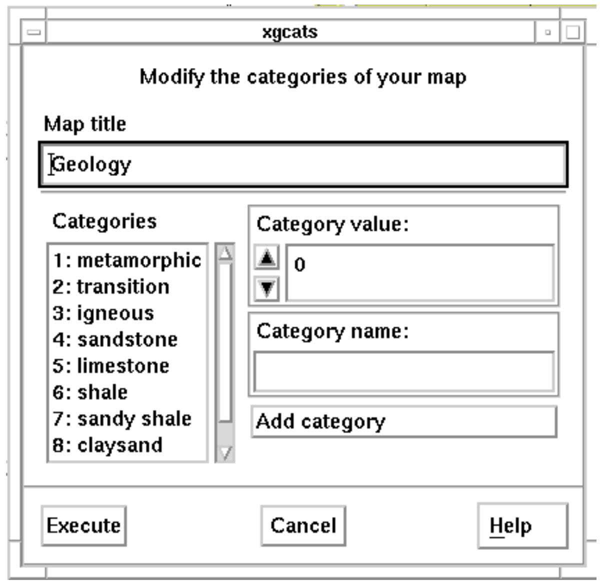

Figure 27 The **`xgcats`** dialog, invoked from the **`xgsupport`** main panel.

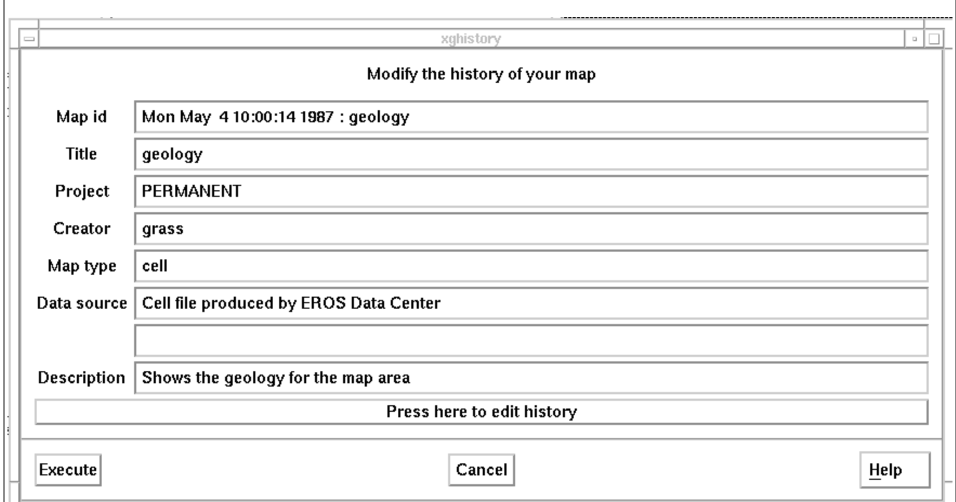

Figure 28 The **`xghistory`** dialog, invoked from the **`xgsupport`** main panel.

Command-Line Usage

`xgsupport` [options] [raster map name]

Where the options are:

-header	The interface will appear with the "Edit Header..." button active.
-cats	The interface will appear with the "Edit Categories..." button active.
-stats	The interface will appear with the "Update Histogram/Range" button active.
-history	The interface will appear with the "Edit History..." button active.

CHAPTER 5

The **xgen** Application Generator Program

xgen is a program designed for the rapid generation of graphical user interfaces to existing shell-level programs based on the X Window System and Motif. A simple, hierarchically oriented language allows an interface designer to define a graphical user interface that is used to collect information from the user and execute shell level commands based on the information. The **xgen** software takes as input an interface description, or script. This script is interpreted, the interface is created, and then operated by **xgen**. The interface gathers information from the user, and, at the appropriate time, composes command lines and runs executable programs based on this information. These executable programs may or may not have their own X Window System interfaces.

xgen fits into the XGRASS software scheme by providing a way to build interfaces to sets of commands without knowledge of a programming language. Completed **xgen** applications can then be integrated into the menu system using the `f.exec` function.

Introduction

When developing an interface using `xgen`, the designer should take into account the nature of the application and subdivide it into logical components or steps that will represent particular tasks within the application interface[1]. These steps will most likely entail the use of a few commands or programs. The major appli-

1. The author acknowledges that this is not necessarily the best strategy for graphical user interface design, yet given the capability of `xgen`, its intended use, and application experience, I feel that this is the best strategy when using `xgen`.

cation divisions are called *environments*. Environments can be further divided into smaller subtasks called *shells*. These shells will most likely involve the execution of a single command and can be used to gather the information needed to run one command or to control the flow of information in an environment or the application as a whole. *Menu* shells are vertically paned, whereas *commandboard* shells provide several layout mechanisms. Each shell is made up of a window on the screen containing a set of *interface objects* that are the particular interface devices used to gather "bits" of information from the user. There are eleven object types in `xgen`: label, separator, button, message, textentry, multiline, list, toggle, slider, table, and pulldown. Each interface object and its object-specific resources are described in "Interface Objects" on page 5-14. These objects, shells, and environments make up a hierarchy (see Figure 29).

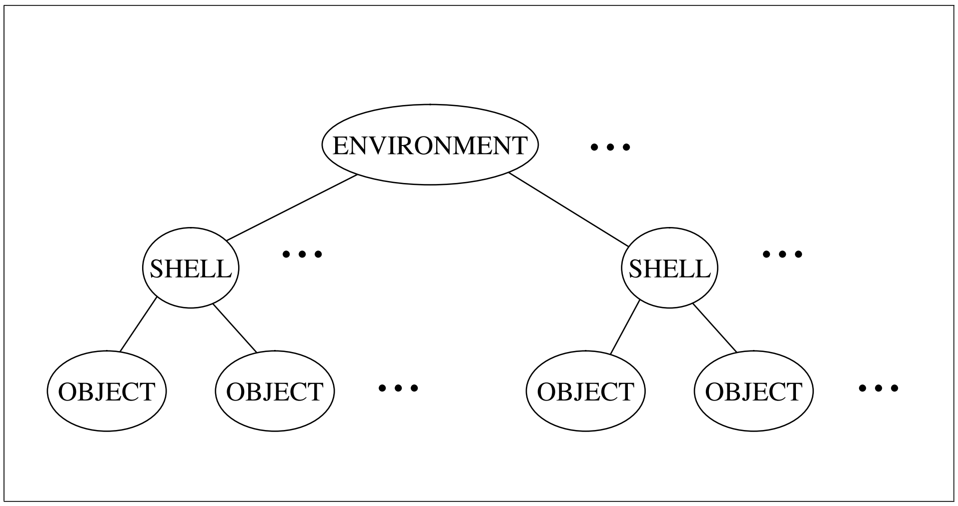

Figure 29 An application hierarchy.

Environments, shells, and interface objects are all modified by *resources*. *Global resources* define the default appearance (such as the font and background color) of all shells and objects in the interface (see Table 3 on page 5-11). *Common procedural resources* are those that may be used to modify environments, shells, and objects in terms of behavior or function (such as setting the sensitivity of an object to user input or running a command). These common procedural resources are listed in Table 2 on page 5-8. *Environment resources* define what happens when the environment is activated (see Table 4 on page 5-11). *Shell* and *object resources* can either affect the appearance of the interface (*aspect* resources) or the behavior of the interface (*procedural* resources). Shell resources define the appearance of an interface window containing objects as well as actions to be taken when the shell window is made visible ("popped up"). Object resources modify the physical appearance of the object and are used to give the object behavior. See Tables 5, 8, 9, 10, and 11 in "Functional Reference" for listings of shell and object resources and their functionality.

Resources can contain *variables*. They are implemented using UNIX environment variables; thus UNIX environment variables set in the shell before execution will be available in xgen (e.g., SPOOLER, HOME, and PATH). Once a variable has been set within xgen, it can be considered global and can be accessed at any time. It will continue to have that value until it is reset or until xgen is terminated. Processes executed are given the same UNIX environment so variables set in xgen will be in the UNIX environment of these processes. More detail on variables is given in "Resources" on page 5-7.

Input Script

The input script format reflects the hierarchical structure of the application. It contains one or more environment descriptions, each of which contains one or more shell descriptions. Each shell description contains one or more object descriptions. The elements of the hierarchy have their own specification syntax and modifying resources. xgen input scripts are preprocessed, by default, using the C preprocessor. This allows for expansion of macros, including files, etc. (See your cpp(1) manual page for more details on its operation.)

Environments are described in the input script using the environment keyword, followed by a unique name, or handle. Any references to the environment in the xgen script will use this handle. Resources affecting environments are displayed in Table 4 in "Functional Reference." All common procedural resources are available and are activated when the environment is made active. The rest of the environment description consists of a list of shell descriptions enclosed by curly braces, { and }.

A shell can be of either type: menu or commandboard. A menu shell is used as a control mechanism within or between environments; commandboard shells are most useful for assembling and running the command-lines needed for the application, although there is no actual restriction on their use. Shells are described in the input script using the menu or commandboard key words, followed by a unique handle, and a list of shell resources. Shell resources are listed in Table 5 in "Functional Reference." All of the common procedural resources work in shell descriptions. These resources are evaluated when the shell is popped up. The remaining portion of the shell description is a list of objects that will be placed within the shell. The object list is enclosed by parentheses, (and).

Objects are specified in the input script by the key word specifying the type of object, followed by an object handle and a list of object resources. Object descriptions are terminated with a semicolon, ;. All object handles must be unique within a shell. All objects have a set of resources in common, which are listed in Table 10.

All resources are defined using a key word and a colon followed by a value. This construction is called a *pair*. For example, to set the background color of an object to light powder blue, the pair would be background:LightPowderBlue. Values can be one of four types:

- integer,
- decimal (floating point),

- boolean, or
- character string.

Values must be of the appropriate type, or, they may be specified as a variable. Variables are described with a dollar sign before the name of the variable, for example, `$Variable`.

`xgen` input scripts, the environments, shells, objects, and their modifying resources are organized as depicted in Figure 30. The examples on pages 5-5 and 5-7 should give the reader a greater appreciation for the format of the input script.

```
global resources
environment environment-handle
        list of environment resources
{
        shell shell-handle
            list of shell resources
        (
                object object-handle
                    list object resources ;
                more objects
        )
        more shell descriptions
}
more environment descriptions
```

Figure 30 The basic organization of the input script.

Program Operation

`xgen` uses a `yacc`(1)-generated parser to read the input script describing the interface appearance and operation and from this information builds an internal data structure. The data structure is then indexed to the first environment description which is then activated. This activation causes resources modifying the environment to be evaluated. `xgen` uses a mandatory environment resource containing the names of one or more shells to determine which shell within that environment to activate initially. The shells in this initial list are then located by indexing into the internal data structure and activated in the order they appear in the initial shell list. The procedural resources of the shell are evaluated first, then the shell and its component interface objects are created. Variables are evaluated at certain key times during `xgen` execution:

1. in environment resources, when the environment is activated;
2. in shell and object aspect resources, when the shell they belong to is “popped up”;
3. in object procedural resources, when the object is activated.

See “Variables” on page 5-21 for an example of variable usage.

Commands are executed within `xgen` using `fork`(2) and `execl`(2) or `execlp`(2). If desired, the standard output of the command can be captured and standard input can be directed to come from a file. When a command fails, the standard error output is displayed in a message box dialog. Interactive commands can be executed, although this functionality is limited (see for details). Execution of interactive commands are implemented by opening a `pipe`(2), doing a `fork` and `exec`, performing a `dup`(2) on the pipe, and then passing subsequent command input down the pipe and capturing and displaying the command output.

Hello, World

The `xgen` input script in Figure 31 contains a single environment with a single shell that implements a “Hello, World” interface. The shell is a menu shell con-

```
font:"*adobe*helv*r*norm*12*" /* Global resource for default font */
/* Begin an environment description, its handle is "hello" */
environment hello
    initialshells:main /* The list of shells to popup initially */
{
    /* Begin a menu shell description, its handle is "main" */
    menu main
        titlestring:"xgen Hello World program" /* Set the shell title */
    (
        /* Begin a label description */
        label "Hello, World"
            ;
        /* Begin a button description */
        button "Echo It"
            runforeground:"echo Hello, World" /* Executed command */
            ;
        /* Begin a button description */
        button "Quit"
            exit:0 /* Quit */
            ;
    )
}
```

Figure 31 The Hello, World input script.

taining a label, a button that will print a string to the calling terminal, and a button to quit the interface. The specification of a global font resource will result in the font “*adobe*helv*r*norm*12*” appearing as the default font in the interface. The resulting interface is depicted in Figure 32 . When `xgen` is invoked, the first environment (in this case, `hello`) is activated. The shell `main` is then popped up, with the shell title set to “`xgen` Hello World Program,” If the user clicks on the “Echo It” button, `xgen` will execute the command `echo Hello, World`. If the user selects the button “Quit,” `xgen` will exit with an exit status of 0.

Abstract Example

Consider the following scenario (graphically depicted in Figure 33) as an example of these concepts. We have an application which accomplishes Tasks A and B. To accomplish A, we need to run commands C, D, and E. To accomplish Task B, we need to invoke commands F and G. We would design an interface containing two environments, one for each task. Within each environment we would design

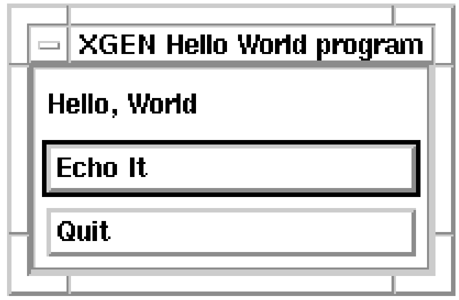

Figure 32 The Hello, World interface.

shells containing appropriate interface objects to gather command-line arguments and execute each command.

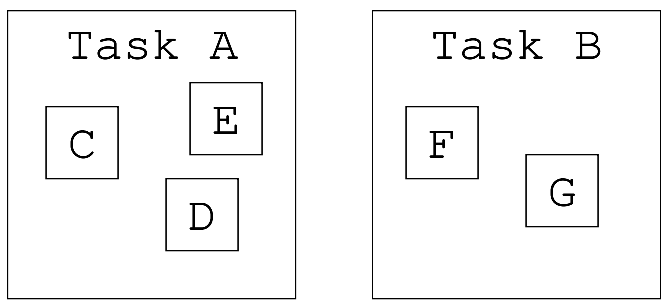

Figure 33 An abstract example.

Using the example in Figure 33, we can construct a partial `xgen` input script. We will assume for the sake of the example that each command takes only one argument and that it can be specified using a textual input from the user. This script is shown in Figure 34. (More insight into the input script format is provided in "Functional Reference" on page 5-7.)

When `xgen` is invoked using this script, the environment `TaskA` will be activated. The shell `CommandC` will then be popped up. Once the user has entered something, say `XyZ`, into the textentry object `CInput` and hit the return key, the command `CommandC XyZ` will be executed. The shell `CommandD` will then be popped up. Once the user has entered something, say `aBc`, into the textentry object `DInput`, the command `CommandD aBc` will be executed and the shell `CommandE` will be popped up. Note that the shells `CommandC` and `CommandD` are still available, and can be activated again by entering something and hitting return. If a shell is already popped up it will not be recreated. Once the user has en-

```
environment TaskA
    initialshells:CommandC
{
    commandboard CommandC
    (
        textentry CInput
            runforeground:"CommandC [CInput]" popup:CommandD ;
    )
    commandboard CommandD
    (
        textentry DInput
            runforeground:"CommandD [DInput]" popup:CommandE ;
    )
    commandboard CommandE
    (
        textentry EInput
            runforeground:"CommandE [EInput]" getenv:TaskB ;
    )
environment TaskB
    initialshells:CommandF
{
    commandboard CommandF
    (
        textentry FInput
            runforeground:"CommandF [FInput]" popup:CommandG ;
    )
    commandboard CommandG
    (
        textentry GInput
            runforeground:"CommandG [GInput]" exit:0 ;
     )
}
```

Figure 34 A script based on the example from Figure 33.

tered something, say `XyZ`, into the textentry object `EInput` and hits return, the command `CommandE XyZ` will be executed. The `getenv` resource will then cause all shells within the `TaskA` environment to be popped down and the environment `TaskB` to be activated and a similar process begins.

One can see from this example how a sort of flow of control can be implemented. These examples are small, but should provide the reader with a good idea of how `xgen` works.

Functional Reference

This portion of the chapter provides the interested reader with a complete description of the elements of the `xgen` script language.

Resources

Resources modify the default appearance or behavior of elements of the hierarchy.

All resources are defined using a key word and a colon followed by a value. This construction is called a *pair*. For example, to set the background color of an object to light powder blue, the pair would be `background:LightPowderBlue`. Values can be one of four types:

- integer,
- decimal (floating point),
- boolean, or
- character string.

Values must be of the appropriate type, or, they may be specified as a variable. Variables are evaluated at certain key times:

1. in environment resources, when the environment is activated;
2. in shell and object aspect resources, when the shell they belong to is "popped up";
3. in object procedural resources, when the object is activated.

Variables are described with a dollar sign before the name of the variable, for example, `$Variable`. They are implemented using UNIX environment variables; thus UNIX environment variables set in the shell before execution will be available in `xgen` (e.g., `SPOOLER`, `HOME`, and `PATH`). Once a variable has been set within `xgen`, it can be considered global and can be accessed at any time. It will continue to have that value until it is reset or until `xgen` is terminated. Processes executed are given the same UNIX environment so variables set in `xgen` will be in the UNIX environment of these processes.

Common Procedural Resources

There are common procedural resources used to set variable values, run commands, change the command interpreter, start an interactive program, or set options concerning how to execute command-lines. An example is the `set` resource, which is used to set a variable to a value specified. It is described as `setVAR:value` where the variable name containing the value will be `VAR` (for example, `setHOMEDIR:/usr/kurt`). The value is checked for variables and expanded at the time the set command is activated. (For example, after the above set command, the value for another set command `setBINDIR:$HOMEDIR/bin` would result in the variable `BINDIR` having the value */usr/kurt/bin*.) The entire set of common procedural resources are described in Table 2. The values of these common procedural resources are checked for variables and expanded at the time they are activated.

Table 2 Common Procedural Resources.

Key Word	Type	Possible Values	Description
runforeground	string	any string	Executes the command in string and waits for the command to complete.
runbackground	string	any string	Executes the command in string, but does not wait for completion of the command.

Table 2 Common Procedural Resources.

Key Word	Type	Possible Values	Description
commandarg	string	any string	Appends the value of string on to the end of the next `runforeground` or `runbackground` command seen in the input script.
captureoutput	string	`editor`, `append`, or any string	Captures the output of the next `runforeground` or `runbackground` command into the file name given by string, in a simple editor if the string is `editor`, or it will append to a common editor if the string is `append`. (This editor will be created the first time an `append` string is seen.)
notify	boolean	on, off, true, false	Notifies the user when a `runbackground` command is completed.
setVAR	string	any string	Sets the UNIX environment variable `VAR` to the value of the string.
inputfrom	string	the name of an existing file	Sets the standard input stream for the next `runforeground` or `runbackground` command to the given file.
insensitive	string	the handle of a shell or an object	Sets the given shell or object insensitive (not responsive to user input). Object handle may be specified as "shell:object" if more than one object has a given handle.
sensitive	string	the handle of a shell or an object	Sets the given shell or object insensitive (not responsive to user input). Object handle may be specified as "shell:object" if more than one object has a given handle.
commandshell	string	an executable program name	Sets the command interpreter for `runforeground` or `runbackground` commands to string. The default `commandshell` is `/bin/sh -c`.

Command Strings

Several resources allow their values to be the result of a command specified in a resource string. In other words, their standard output is used as the value of the resource. Variables within command strings will be expanded before the command-line is executed unless they are preceded by with another $. Double quotes that need to be passed to the shell in a command string can be quoted using a \ character. As an example of the last two points, consider the command string:

```
!awk '{printf \"%s:\",$$0}' /tmp/junk
```

When expanded, this command string would be:

```
awk '{printf "%s:",$0}' /tmp/junk
```

This would then be passed to the shell for execution. (This command-line has the effect of joining the lines in the file */tmp/junk*, separating them with colons.) Square brackets, [and], also have special meaning to `xgen` and can be quoted in any string using the \ character.

The `interactive` Resource

Normal use of `xgen` is to gather information from a user through the setting of variables and interaction with interface objects culminating with a `runforeground` or `runbackground` procedure activated in response to user interaction. The `runforeground` or `runbackground` value is then passed to `/bin/sh` for execution. Obviously, some commands are interactive, meaning that they already have a character-based user interface. They run and prompt the user for information (usually on standard output) and read the response (usually on standard input). The environment resource `interactive` was designed to allow the programmer to provide an interface to these types of programs. (Note that for some programs it is very difficult or impossible to provide a useful interface with `xgen`. This is primarily due to the inconsistent use of `stderr` and `stdout` by these programs.) When `interactive` is used to invoke a command, the full path and all the command-line options must be specified. `interactive` modifies the behavior of the common procedural resources in the following ways:

- `runforeground` — instead of running the string as a command, the string is passed to the interactive program as a line of input. (NOTE: you cannot send control characters to an interactive command.)

  `KillInterActive` is a `runforeground` value that will stop the execution of an interactive command that normally requires an EOF (`^D`) to stop running (e.g., `cat`, `nslookup`). In other words, the pair `runforeground:"KillInterActive"` would kill such a program.
- `runbackground` — behaves exactly like `runforeground` (the background aspect is ignored).
- `captureoutput` — no effect.
- `inputfrom` — no effect.

The `interactive` command will not stop when the environment from which it was executed is stopped, so be sure to stop the execution of the interactive command using `runforeground:"quit"` (providing the command with instructions to stop), or with the `runforeground:"KillInterActive"` pair.

Global Resources

Certain resources that affect the default appearance of the user interface may be set at the beginning of an `xgen` script. These resources are listed in Table 3 .

Environments

Environments are described in the input script using the `environment` key word, followed by a unique name, or handle. Any references to the environment

Table 3 Global Resources

Key Word	Type	Possible Values	Description
font	string	any legal X Window font description	Sets the default font in the interface to the value given.
foreground	string	any legal X Window color specification	Sets the default foreground color to the value given.
background	string	any legal X Window color specification	Sets the default background color to the value given.

in the `xgen` script will use this handle. Resources affecting environments are displayed in Table 4. The `initialshells` resource is mandatory. All special resources are available and are activated when the environment is made active. The rest of the environment description consists of a list of shell descriptions enclosed by curly braces, { and }.

Table 4 Environment Resources

Key Word	Type	Possible Values	Description
initialshells	string	a list of valid shell handles	Specifies the shells that are popped up when the environment becomes active.
interactive	string	a UNIX command-line	Specifies an interactive program to start that normally runs with interactive input (See "The interactive Resource" on page 5-10).

Shells

A shell can be of either type: menu or commandboard. A menu shell is used as a control mechanism within or between environments; commandboard shells are most useful for assembling and running the command-lines needed for the application, although there is no actual restriction on their use. Shells are described in the input script using the `menu` or `commandboard` key words, followed by a unique handle, and a list of shell resources. Shell resources are listed in Table 5. All of the common procedural resources work in shell descriptions. These resources are evaluated when the shell is popped up. The remaining portion of the shell description is a list of objects that will be placed within the shell. The object list is enclosed by parentheses, (and).

Table 5 Shell Resources

Key Word	Type	Possible Values	Description
x	integer	any integer number	Places the shell at `x` pixels from the left-hand side of the screen. If negative, the position is measured from the right-hand side of the screen.
y	integer	any integer number	Places the shell at `y` pixels from the top of the screen. If negative, the position is measured from the bottom of the screen.

Table 5 Shell Resources

Key Word	Type	Possible Values	Description
dx	real	any real number, in the range 0.0 to 100.0	Places the shell at the x position as a percentage of the screen width from the left-hand side. If negative, it is measured from the right-hand side of the screen.
dy	real	any real number, in the range 0.0 to 100.0	Places the shell at the x position as a percentage of the screen width from the left-hand side. If negative, it is measured from the right-hand side of the screen.
width	integer	any integer number	Sets the width of the shell in pixels.
height	integer	any integer number	Sets the height of the shell in pixels.
foreground	string	any string representation of an X Window color	Will set the default foreground color of the shell to the color described in the value.
background	string	any string representation of an X Window color	Will set the default background color of the shell to the color described in the value.
titlestring	string	any string	Sets the title of the shell to the value of the string. The title is by default the handle of the shell. When this resource is set, the internal handle will not change, but the title displayed in the title area of the window manager frame will change.
decorations	string	a list of valid decorations	Affects `mwm` window decorations. Has no effect when used with other window managers.
functions	string	a list of valid functions	Affects `mwm` window menu functions. Has no effect when used with other window managers.
override	boolean	`on`, `off`, `true`, or `false`	Overrides the window manager intervention. This resource is dangerous and definitely not compliant with X Window standards, use sparingly.
help	string	any string	Causes a menu bar with a help button to appear on the shell. When activated, it displays the string in a popup dialog.

The `decorations` and `functions` resources

The resources decorations and functions work only with the mwm window manager and serve the purpose of allowing control of the window manager decorations and window menu functions, respectively. The possible values for these resources are listed in Tables 6 and 7.

Menu Shells

As stated before, menus are frequently used to control the application interface. They can contain button, label, or separator objects only (see "Interface Objects" on page 5-14 for more information). These objects are vertically paned and of

Table 6 Values for the decorations Resource

Value	Description
all	All decorations are displayed.
border	The border is displayed.
resizeh	The resize handles are displayed (includes border).
title	The title bar is displayed (includes border).
menu	The window menu is displayed (includes title bar).
minimize	The maximize button is displayed (includes title bar).
maximize	The minimize button is displayed (includes title bar).

Table 7 Values for the functions Resource

Value	Description
all	All functions are included in the window menu.
resize	Includes the `f.resize` window management function.
move	Includes the `f.move` window management function.
close	Includes the `f.kill` window management function.
minimize	Includes the `f.minimize` window management function.
maximize	Includes the `f.maximize` window management function.

equal size. There are two kinds of menu shells: static menus and dynamic menus. Static menus are those that are fully described before run-time in the input script. Dynamic menus are *not* fully described until run-time. More specifically, each time a dynamic menu is popped up it is updated. The dynamic menu consists of a set of buttons derived from one of three sources. One button is created for each element from the source. The sources and a description of their elements follows:

- The contents of a file. An element is one line of text terminated by a newline.
- The contents of a directory. An element is a file or directory name.
- The output of a command. An element is one line of textual output terminated by a newline. This source is specified as a command string.

The contents of a dynamic menu is specified by the `updatefrom` resource. For example, `updatefrom:"!ls"` would execute a listing of the current directory and make a button for each line of its output. The method for tying actions to menu buttons is described later. Static menus or dynamic menus can be forced to display only a specific number of items using the `visibleitems` resource; a scroll bar will be added if necessary. See “Dynamic Menus” on page 5-21, for an example of the use of dynamic menus. Resources specific to menu shells are listed in Table 8.

Table 8 Menu Shell-Specific Resources

Key Word	Type	Possible Values	Description
updatefrom	string	a command-line, directory name, or a file name	Creates the ability to add a button named `dynamic` to the shell description that will create some number of buttons (see text on page 5-13).
maxheight	integer	any integer, less than the height of the screen	Forces menus to be less than or equal to the height specified.
maxwidth	integer	any integer, less than the width of the screen	Forces menus to be less than or equal to the width specified.
visibleitems	integer	any integer less than the number of items in the menu	Force the menu to display only the number of items specified and adds a scroll bar if more items exist.

Commandboard Shells

As stated before, commandboard shells are most frequently used to gather command-line information and to execute commands within the application interface. Commandboard shells can contain any object. These objects are vertically or horizontally paned when the `pane` resource is `true`. Resources specific to commandboard shells are shown in Table 9.

Table 9 Commandboard Shell-Specific Resources

Key Word	Type	Possible Values	Description
pane	boolean	`on`, `off`, `true`, or `false`	Arranges the objects contained in the shell in a vertically tiled manner further described by the `panetype` and `orientation` resources.
panetype	string	one of `squish`, `pack`, or `none`	Affects the method of tiling. `pack` makes all objects the same size in both dimensions. `squish` makes them they size they want to be in the orientation dimension, and `none` does no packing at all.
forcesize	boolean	`on`, `off`, `true`, or `false`	Forces the width and height attributes of a non-paned commandboard to be heeded, regardless of the positions of the objects within the shell.
orientation	string	`horizontal` or `vertical`	Affects the orientation of a paned commandboard.
columns	integer	any integer	Sets the number of columns (or rows, depending on the orientation) in a paned commandboard providing a simple mechanism to do tabular layouts.

Interface Objects

There are eleven object types in `xgen`: `label`, `separator`, `button`, `message`, `textentry`, `multiline`, `list`, `toggle`, `slider`, `table`, and `pulldown`.

Each interface object and its object-specific resources are described in the following sections. Objects are specified in the input script by the key word specifying the type of object, followed by an object handle and a list of object resources. Object descriptions are terminated with a semicolon, ;. All object handles must be unique within a shell. All objects have a set of resources in common, which are listed in Table 10.

Table 10 Object Resources

Key Word	Type	Possible Values	Description
x	integer	any integer	Sets the x position of the object in pixels from the left side of the enclosing shell.
y	integer	any integer	Sets the y position of the object in pixels from the top of the enclosing shell.
width	integer	any integer, less than the width of the display	Sets the width of the object.
height	integer	any integer less than the height of the display	Sets the height of the object.
font	string	any string describing a valid X Window font	Sets the font used to display text in the object.
foreground	string	any string representing a valid X Window color	Sets the foreground color of the object.
background	string	any string representing a valid X Window color	Sets the background color of the object.
titlestring	string	any string	Sets the name of the object for display purposes (the internal name remains the handle given in the script).

Buttons, textentries, toggles, and (in the future) lists (so called, *active objects*) can have procedural resources associated with them and are invoked when the object is activated. Activation occurs in the following manner:

- Buttons are activated by clicking the mouse on the button,
- Textentries are activated by pressing the Return key,
- Toggles and lists are activated by selecting one of the toggle buttons or list items.

These procedural resources include the common procedural resources (see Table 2) and object specific procedural resources that are given in Table 11.

The `updateobject` resource

The object procedural resource `updateobject` serves to allow for the update of objects in response to user actions. The value of the `updateobject` resource is specified by a colon-separated pair of shell, object, and value. For example, as-

Table 11 Object Procedural Resources

Key Word	Type	Possible Values	Description
popup	string	a valid shell handle	Causes the shell referenced to be created and "popped up" to the screen.
popdown	string	a valid shell handle	Causes the shell referenced to be "popped down" from the screen.
getenv	string	a valid environment handle	Causes the environment referenced to be activated. This will pop up all shells referenced in the new environments `initialshells` resource to be popped up, and all shells in the current environment to be popped down.
updateobject	string	a pair containing "shell:object:value"	Causes the object in shell to receive the value specified. The specific characteristic affected by the `updateobject` for each object is listed in the section "The updateobject resource" on page 5-15
clear	string	a list of valid object handles	Clears the given o.bjects to an initial state. Object handle may be specified as "shell:object" if more than one object has a given handle.
store	string	any string	Stores the contents of the given sting into the X Window "clipboard."
exit	integer	any integer	Exits `xgen` with the given exit code.

suming that a shell exists that is called TopLevel, and a label in that shell is named Status, the value "`TopLevel:Status:Working...`" would change the text of the label to the string "Working...". The specific aspect of the object changed with an updateobject procedure is as follows (if not listed, the object cannot be updated):

- Label — The label is changed to the value specified.
- Button — The text of the button is changed to the value specified.
- Textentry — The text is changed to the value specified.
- Multiline — The text is changed to the value specified.
- Slider — Changes the value of the slider; the value in this case must be a real number or integer in the range of the slider.
- Table — The table is changed according to the value specified, and the way that the table is referenced. The table reference can be the entire table, an entire row, or a single cell. The format for this reference is the same as described in the last paragraph of the section "Table" on page 5-19. The value must be a `separator` separated list, unless the reference is to a single cell.

Label

Label objects are textual or iconic objects that have no action, but are useful for labeling areas of shells. An icon, or pixmap, is specified as the name of a file containing the bitmap in X Window format and the `labelpixmap` resource is used for this purpose. The `alignment` resource is used to align the text or pixmap within the label area. The alignment can be `center`, `left`, or `right`, with obvious results.

Separator

Separator objects are linear objects used to visually divide the shell real estate. Separators have no actions associated with them. They can have a `horizontal` or `vertical` value for the `orientation` resource, but they are oriented horizontally by default. The `separatortype` resource describes the appearance of the separator. The following are the types available:

- `singleline` and `doubleline` — a single or double line in the foreground color;
- `singledashedline` and `doubledashedline` — a single or double dashed line in the foreground color;
- `noline` — just the space that would be used if a line were there;
- `shadowetchedin` or `shadowetchedout` — these are three-dimensional-appearing lines that have some thickness.

Button

Buttons are labeled objects that are selectable using the left mouse button. Actions can be tied to them using the common procedural resources (see Table 2) and object procedural resources (see Table 11). Buttons also have the two label resources `labelpixmap` and `alignment`. A `help` resource can be set to a string which will be popped up into a help dialog box when help for the button is activated by clicking with the Shift key held down.

Message

Messages are multiline labels and have no actions. They do have two aspect resources, `alignment` and `columns`. `alignment` works exactly like the label and button resource of the same name. The `columns` resource has an integer value that describes the maximum number of characters that will fit on each line. If the message is longer than the number of columns specified, the message is divided at word breaks and spread over as many lines as it takes to display the entire message.

Textentry

Textentry objects are one-line text editors useful for gathering textual input from the user. They have only one additional aspect resource: `valuestring`. The value of this resources is a string that will be used as the initial value of the object. Textentry objects can be cleared using the `clear` procedure tied to an active object, such as a button. The value of a textentry object can be used in any of the common procedural resources' value strings by enclosing the handle of that object in square brackets. For example, if an application had a textentry object named t1 and the user had entered */usr/include* into the object, then `runforeground:"ls -l [t1]"` would give a long listing of the */usr/include* directory.

Multiline

Multiline objects are multiline textentry objects and are functionally similar to textentries. They have two resources: `rows` and `columns` which control the size of the object in characters. For example, the pairs `columns:80` and `rows:24` specifies a multiline that is 80 characters wide by 24 characters tall. The `scrolled` resource can be either `true` or `false`, and specifies whether or not `xgen` provides scroll bars for the object.

List

List objects are groups of selectable labels. They are useful for gathering selections of files or command-line arguments from the user. When the `visible-items` resource is used to restrict the number of list items appearing, a scroll bar is provided. List objects can be cleared using the `clear` procedure tied to an active object. Elements of the list are specified explicitly using the `listelement` resource. Each `listelement` resource pair specifies the text displayed in a list element. Following each `listelement` resource is an optional `valuestring` resource used to specify the actual value of the list object. This value can be utilized in any of the common procedural resource value strings by employing the object notation described in the textentry section.

For list objects with multiple values, there exists a `listseparator` resource; the value of this resource is a string that separates the selected items when the value of the object is requested. For example, if the `listseparator` value consists of a blank and the selected list items are a, b, r, and g, then the value of the list object would be "a b r g." The `listseparator` can be a special value `newline`, with obvious results. List elements may also be described using the `updatefrom` resource; it behaves exactly as in the dynamic menu shell.

List objects may also have a `listtype` resource specified that determines the selection mechanism behavior (how items are selected and how many items may be selected) for that list object. Items are selected when they appear highlighted after all mouse action has been completed. The following describes the possible selection types:

- `single` — Items are selected by moving the mouse pointer to the one desired and clicking the mouse. Any previously selected item is deselected. Only one item in the list can be selected at a time.
- `multiple` — Items are selected by moving the mouse pointer to the one desired and clicking the mouse. Any previously selected item will remain selected. If the item clicked on is already selected it will become unselected. Any number of items in the list may be selected.
- `extended` — Items are selected in contiguous ranges with one action. The mouse button is depressed over the first item in the range and then dragged, with the button still depressed, to the last item in the range, and released. Be-

ginning a new selection process will deselect all previously selected items. Any number of items in a contiguous range can be selected.

- `browse` — This selection type is similar to the extended selection, but only one item will be selected as the pointer is moved across the list items. The item under the pointer when the button is released is the one selected. One item at a time may be selected.

No actions can be explicitly tied to list objects with the current release, but this feature is planned for future releases. Therefore it is necessary to use them in conjunction with an active object.

Toggle

Toggle objects are very similar to list objects in that they are useful for gathering selections of files or command-line arguments. They are used when the number of possible items to select is small. All items are specified using the `listelement - valuestring` mechanism described above. Selections are made by clicking the mouse on the item or on the small button next to the item. When an item is selected, the small button will appear to be depressed. There are two types of selection state, 1-of-many and N-of-many, and these are indicated graphically by a diamond- or square-shaped button, respectively. The value of a toggle items is retrievable using the same technique described in the textentry object section. The toggle object also supports the `listseparator` resource as described in the list object. The `listseparator` can be a special value `newline`, with obvious results. The `toggletype` resource is used to indicate which selection state is to be used. The possible value for this resource is `radio` (which indicates the 1-of-many scheme), otherwise the N-of-many state is used.

Slider

Slider objects are useful for obtaining input in a relatively small numerical range. Sliders are made up of a scrolling region with a handle for changing the value of the slider, a readout that will display the current value, and a label. The `minimum`, `maximum`, and `startvalue` resources are used to indicate the minimum and maximum slider value and the starting value of a slider object. These values are specified as integer numbers. The `decimalpoint` resource is used to specify the number of decimal points to left shift the decimal point in the slider value for display and retrieval. For example, a value of 12345 with a `decimalpoint` value of 3 results in 12.345. The `orientation` resource is used to specify whether the slider is oriented horizontally (`horizontal`, this is the default orientation) or vertically (`vertical`). The `titlestring` resource is used as the label for the slider object.

Table

Table objects are useful for displaying or editing tabular data for input from or output to database commands. A table is a complex object consisting of a title and a matrix of text widgets with row and column headings.

The titlestring resource is used as the title for the table. The rowheadings and columnheadings resources are used to set the row and column headings, respectively. These must contain a list of words or phrases separated by the character string specified in the separator resource. The values in the table may be initialized using the rowvalue resource, where each occurrence is set as the next row in the table. There must also be lists of elements separated by the character string specified in the separator resource. The separator used by tables can have the special newline value. The tablevalue resource will load an entire table at once. Its value is a file name containing the row values, in appropriate format, or a command string whose standard output will generate the appropriate output.

The table will be columns wide by rows long. The rowsdisplayed and columnsdisplayed resources are used to control the number of rows and columns displayed and scroll bars will be added if necessary. When the scroll bars are operated, the row and column headings will scroll with the matrix. The columnwidth or rowheight may be set to an integer number of characters if the defaults are not large enough. The resources fixedrows and fixedcolumns may be used to control the number of columns to keep fixed (meaning that they will not scroll with the rest of the matrix). These resources are mutually exclusive (i.e., only one can be used at one time).

Like lists, toggles, sliders, and textentry objects, the table values can be specified in value strings using the square bracket notation. The contents of an entire table can be specified using the string [table1], a single row using [table1(1)], or a single element using [table(1,2)]. The rows and columns are numbered starting at (1,1) in the upper left-hand corner, increasing right and down. The table values are output in the same format as input, meaning that the output will be separated by the character string specified in the separator resource.

Pulldown

Pulldown menu objects are also more complex than the others. They are used when many actions must be invoked from a single commandboard shell and can contain only button or separator objects. The syntax of pulldown menus is different from other objects in that they are specified much like a menu shell terminated with a ;. Each pulldown menu description listed in a shell description is placed in a menu bar that stretches across the top of the shell in the order that they occur in that description. For example, the script in Figure 35 will create a menu named “File” with two separated items.

xgen Code Examples

In this section several examples are given that will illustrate the use of various xgen capabilities. In each subsection an xgen script is given that demonstrates some unit of functionality.

```
pulldown File
    (
        button Item1
            runforeground:"echo You pressed me..."
        ;
        separator ASeparator
            separatortype:shadowetchedin
        ;
        button Quit
            exit:0
        ;
    )
    ;
```

Figure 35 A partial script detailing the pulldown syntax.

Variables

The script in Figure 36 demonstrates the use of variables. When the user presses the button "Echo a string with variable" the interface will print the string "advance." If the user presses the button "Set a variable" and then presses "Echo a string with variable" again, the interface will print the string "advTHANKSance."

```
environment a
    initialshells:b
{
    menu b
        setVAR:""
    (
    button "Set a variable"
        setVAR:"THANKS" ;
    button "Echo a string with variable "
        runforeground:"echo adv${VAR}ance" ;
    button "Quit"
        exit:0 ;
    )
}
```

Figure 36 An example of the use of variables.

Object Values

This section demonstrates the retrieval of object values. The interface generated by the script listed in Figure 37 will present two rows of toggle buttons. When the user presses the Echo Choices button, `xgen` will echo all of the selected toggle button values in the cc object followed by all of the selected toggle button values in the dd object. For example, if the user selects blue and red from cc and orange from dd, the interface will print the string "blue red orange".

Dynamic Menus

Figure 38 demonstrates the use of a dynamic menu. The number of items visible is controlled using the `visibleitems` resource. In this example, the menu is constructed using the `updatefrom` resource. This results in a menu of file and directory names that are in the current directory. Notice that only one object de-

```
environment aa
    initialshells:bb
{
    commandboard bb x:100 y:100 width:200
    (
        label "Color One:" x:0 y:0 ;
        toggle cc x:0 y:40 orientation:horizontal
            listelement:"blue"
            listelement:"red"
            listseparator:" " ;
        label "Color Two" x:0 y:100 ;
        toggle dd x:0 y:130 orientation:horizontal
            listelement:"yellow"
            listelement:"orange"
            listseparator:" " ;
        button "Echo Choices" x:0 y:190
            runforeground:"echo [cc] [dd]" ;
    )
}
```

Figure 37 An example of retreiving object values.

scription is needed, and that its handle is dynamic. This button description also controls the procedures to be taken when any of the items are selected. In this case, a variable (XXX) is set to be the value of the button label and then echoed to the screen. This is accomplished using the special set value buttonname.

```
environment a
initialshells:b
{
    menu b visibleitems:8 updatefrom:"!ls"
    (
        button dynamic
            setXXX:"buttonname"
            runforeground:"echo $XXX" ;
    )
}
```

Figure 38 A dynamic menu example.

Updating Object Values

This section demonstrates the update of an object in response to an action associated with another object. Figure 39 depicts a simple table object. It consists of two shells, a menu, and a commandboard. The menu is used to control the update of objects in the commandboard.

```
environment a
initialshells:"b c"
{
    menu b
    (
    button "Update Button"
        updateobject:"c:BUTTON:I been updated" ;
    button "Update Table(1,1)"
        updateobject:"c:TABLE(1,1):5" ;
    button "Quit"
        exit:0 ;
    )
    commandboard c
        x:300 y:0 pane:on panetype:squish setVAR:""
    (
    label LABEL ;
    button BUTTON
        setVAR:"ButtonPressed"
        updateobject:"c:TEXT:$VAR" ;
    textentry TEXT
        valuestring:"This is a text" ;
    table TABLE
        rows:2 rowheadings:"1,2" rowsdisplayed:2
        columns:2 columnheadings:"1,2" columnsdisplayed:2
        rowvalue:"1,2" rowvalue:"3,4" ;
    )
}
```

Figure 39 An example of object updating and the table object.

Appendix A

xgrass Default Menu Specification

```
Menu DefaultMenu
{
"GRASS Shell"          _l      Meta<Key>l      f.pulldown ShellMenu
"Display"              _D      Meta<Key>D      f.pulldown DisplayMenu
"MapManagement"        _g      Meta<Key>g      f.pulldown MapManagementMenu
"Imagery"              _I      Meta<Key>I      f.pulldown ImageryMenu
"Import"               _m      Meta<Key>m      f.pulldown ImportMenu
"Paint/Print"          _P      Meta<Key>P      f.pulldown PaintPrintMenu
"Raster"               _R      Meta<Key>R      f.pulldown RasterMenu
"Vector"               _V      Meta<Key>V      f.pulldown VectorMenu
"Sites"                _S      Meta<Key>S      f.pulldown SitesMenu
"History"              _t      Meta<Key>T      f.pulldown HistoryMenu
"Help"                 _H      Meta<Key>H      f.pulldown HelpMenu
"Quit"                 _Q      Meta<Key>Q      f.pulldown QuitMenu
}

/***** ShellMenu *****/
Menu ShellMenu
{
"Shell"          f.exec "xterm -e xgrass.sh"
}

/***** DisplayMenu *****/

Menu DisplayMenu
{
"Display Tool"                                 f.exec "xgdisplay &"
"Manage"                                       f.pullright DisplayMenu1
"Display"                                      f.pullright DisplayMenu2
"Report"                                       f.pullright DisplayMenu3
}

Menu DisplayMenu1
{
```

```
"Control Display Monitor"                      f.xclip "d.mon.xclip"
"Erase Display Frame"                          f.xclip "d.erase.xclip"
"Manage Display Frames"                        f.xclip "d.frame.xclip"
"Establish Color Table"                        f.xclip "d.colormode.xclip"
}

Menu DisplayMenu2
{
"Raster"                                       f.pullright DisplayMenu21
"Vector"                                       f.pullright DisplayMenu22
"Sites"                                        f.pullright DisplayMenu23
"Text"                                         f.pullright DisplayMenu24
"Graphics"                                     f.pullright DisplayMenu25
}

Menu DisplayMenu21
{
"Display Raster Maps"                          f.xclip "d.rast.xclip"
"Display HIS Values"                           f.xclip "d.his.xclip"
"Display Red, Green, Blue Overlays"            f.xclip "d.rgb.xclip"
"Display 3-Dimensional Images"                 f.xclip "d.3d.xclip"
/*
"d.profile"                                    f.label
"d.rast.zoom"                                  f.label
"d.rast.num"                                   f.label
"d.rast.arrow"                                 f.label
"d.rast.edit"                                  f.label
"d.colors"                                     f.label
"d.zoom"                                       f.label
*/
}

Menu DisplayMenu22
{
"Display Vector Maps"                          f.xclip "d.vect.xclip"
"Display USGS DLG-3 Files"                     f.xclip "d.vect.dlg.xclip"
}

Menu DisplayMenu23
{
"Display Sites Markers"                        f.xclip "d.sites.xclip"
"Display Point Graphics"                       f.xclip "d.points.xclip"
"Display Points as Icons"                      f.xclip "d.icons.xclip"
}

Menu DisplayMenu24
{
"Display Map Title"                            f.xclip "d.title.xclip"
"Display Legend"                               f.xclip "d.legend.xclip"
"Display Text Labels"                          f.xclip "d.label.xclip"
"Display Text Labels for Paint Output"         f.xclip "d.paint.labels.xclip"
"Select Text Font"                             f.xclip "d.font.xclip"
"Draw Text"                                    f.xclip "d.text.xclip"
/*
"d.labels"                                     f.label
"d.menu"                                       f.label
*/
}
```

```
Menu DisplayMenu25
{
"Display Color Table"                              f.xclip "d.colortable.xclip"
"Display Geodesic Line"                            f.xclip "d.geodesic.xclip"
"Display Rhumbline"                                f.xclip "d.rhumbline.xclip"
"Display Histogram"                                f.xclip "d.histogram.xclip"
"Overlay Bar Scale and North Arrow"                f.xclip "d.scale.xclip"
"Overlay Grid"                                     f.xclip "d.grid.xclip"
}

Menu DisplayMenu3
{
"Raster"                                           f.pullright DisplayMenu71
/*
"Vector"                                           f.pullright DisplayMenu72
"Location"                                         f.pullright DisplayMenu73
*/
}

Menu DisplayMenu71
{
"Display Histogram"                                f.xclip "d.histogram.xclip"
"Display Legend"                                   f.xclip "d.legend.xclip"
/*
"d.measure"                                        f.label
"d.profile"                                        f.label
"d.what.rast"                                      f.label
*/
}

/*
Menu DisplayMenu72
{
"d.measure"                                        f.label
}

Menu DisplayMenu73
{
"d.where"                                          f.label
}
*/

/***** MapManagementMenu *****/

Menu MapManagementMenu
{
"Set Region from Default"    f.exec "g.region -d"
"Set Region from Raster Ma   f.exec "g.region rast=`xgbrowser -numLists 1 -raster
-single`"
"Modify Current Region"      f.exec "xgregion -grid -current &"
"Modify Default Region"      f.exec "xgregion -grid -default &"
"Access"                     f.pullright MapManagementMenu1
"List"                       f.pullright MapManagementMenu2
"Copy"                       f.pullright MapManagementMenu3
"Rename"                     f.pullright MapManagementMenu4
"Remove"                     f.pullright MapManagementMenu5
}

Menu MapManagementMenu1
```

```
{
"Change Database/Location/Mapset"             f.dbset
"Set Mapset Permissions"                      f.exec "xgaccess &"
"Set Mapset Search Path"                      f.exec "xgmapsets -title Path &"
}

Menu MapManagementMenu2
{
"List GRASS Files"                            f.xclip "g.list.xclip"
}

Menu MapManagementMenu3
{
"Raster"                                      f.xclip "g.copyrast.xclip"
"Vector"                                      f.xclip "g.copyvect.xclip"
"Sites"                                       f.xclip "g.copysites.xclip"
"Icon"                                        f.xclip "g.copyicon.xclip"
"Labels"                                      f.xclip "g.copylabels.xclip"
"Region"                                      f.xclip "g.copyregion.xclip"
"Group"                                       f.xclip "g.copygroup.xclip"
}

Menu MapManagementMenu4
{
"Raster"                                      f.xclip "g.renamerast.xclip"
"Vector"                                      f.xclip "g.renamevect.xclip"
"Sites"                                       f.xclip "g.renamesites.xclip"
"Icon"                                        f.xclip "g.renameicon.xclip"
"Labels"                                      f.xclip "g.renamelabels.xclip"
"Region"                                      f.xclip "g.renameregion.xclip"
"Group"                                       f.xclip "g.renamegroup.xclip"
}

Menu MapManagementMenu5
{
"Raster"                                      f.xclip "g.removerast.xclip"
"Vector"                                      f.xclip "g.removevect.xclip"
"Sites"                                       f.xclip "g.removesites.xclip"
"Icon"                                        f.xclip "g.removeicon.xclip"
"Labels"                                      f.xclip "g.removelabels.xclip"
"Region"                                      f.xclip "g.removeregion.xclip"
"Group"                                       f.xclip "g.removegroup.xclip"
}

/***** ImageryMenu *****/

Menu ImageryMenu
{
"Extract"                                     f.pullright ImageryMenu1
"Manage"                                      f.pullright ImageryMenu2
"Classify"                                    f.pullright ImageryMenu3
"Rectify"                                     f.pullright ImageryMenu4
"Correct"                                     f.pullright ImageryMenu5
"Enhance"                                     f.pullright ImageryMenu6
}

Menu ImageryMenu1
{
"Extract Header Information from LANDSAT MSS" f.xclip "i.tape.mss.h.xclip"
```

```
""      f.separator
"Use GRASS Shell for These Routines"    f.label
""       f.separator
"i.tape.mss"                                    f.label
"i.tape.tm"                                     f.label
"i.tape.other"                                  f.label
}

Menu ImageryMenu2
{
""      f.separator
"Use GRASS Shell for These Routines"    f.label
""       f.separator
"i.build.blk"                                   f.label
"i.group"                                       f.label
"i.target"                                      f.label
"i.camera"                                      f.label
"i.mod.camera"                                  f.label
}

Menu ImageryMenu3
{
"Perform Canonical Components Analysis"         f.xclip "i.cca.xclip"
"Perform Principle Components Analysis"         f.xclip "i.pca.xclip"
"Perform Zero-Crossing Edge Detection"          f.xclip "i.zc.xclip"
""      f.separator
"Use GRASS Shell for These Routines"    f.label
""       f.separator
"i.class"                                       f.label
"i.maxlik"                                      f.label
"i.cluster"                                     f.label
}

Menu ImageryMenu4
{
""      f.separator
"Use GRASS Shell for These Routines"    f.label
""       f.separator
"i.rectify"                                     f.label
"i.rectify.blk"                                 f.label
"i.points"                                      f.label
"i.build.blk"                                   f.label
}

Menu ImageryMenu5
{
"Perform Fast Fourier Transformation"           f.xclip "i.fft.xclip"
"Perform Inverse Fast Fourier Transformation"   f.xclip "i.ifft.xclip"
"Perform Zero-Crossing Edge Detection"          f.xclip "i.zc.xclip"
}

Menu ImageryMenu6
{
"Perform HIS to RGB Transformation"             f.xclip "i.his.rgb.xclip"
"Perform RGB to HIS Transformation"             f.xclip "i.rgb.his.xclip"
""      f.separator
"Use GRASS Shell for These Routines"    f.label
""       f.separator
"i.colors"                                      f.label
```

```
"i.composite"                                   f.label
"i.grey.scale"                                  f.label
"i.median"                                      f.label
}

/***** ImportMenu *****/

Menu ImportMenu
{
"Examine"                                       f.pullright ImportMenu1
"Extract"                                       f.pullright ImportMenu2
"Convert"                                       f.pullright ImportMenu3
}

Menu ImportMenu1
{
"Get DEM Data File Descriptions"                f.xclip "m.dem.examine.xclip"
"Get DTED Data File Descriptions"               f.xclip "m.dted.examine.xclip"
"Find Geographic Region Information"            f.xclip "m.tiger.region.xclip"
""      f.separator
"Use GRASS Shell for These Routines"    f.label
""       f.separator
"m.examine.tape"                                f.label
}

Menu ImportMenu2
{
"Extract DEM Data from Tape"                    f.xclip "m.dem.extract.xclip"
"Extract DMA DTED Data from Tape"               f.xclip "m.dted.extract.xclip"
"Extract USGS DTED Data from Tape"              f.xclip "m.dmaUSGSread.xclip"
"Extract Landuse/Landcover Data from Tape"      f.xclip "m.lulc.read.xclip"
}

Menu ImportMenu3
{
"Convert Geocentric Coordinates to Geographic"  f.xclip "m.gc2ll.xclip"
"Convert Geographic Coordinates to Geocentric"  f.xclip "m.ll2gc.xclip"
"Convert Geographic Coordinates to UTM"         f.xclip "m.ll2u.xclip"
"Convert UTM Coordinates to Geographic"         f.xclip "m.u2ll.xclip"
"Convert Region UTM Coordinates to Geographic"  f.xclip "m.region.ll.xclip"
"Shift Datum"                                   f.xclip "m.datum.shift.xclip"
"Create Raster Map from CTG File"               f.xclip "m.lulc.USGS.xclip"
"Rotate Elevation Data"                         f.xclip "m.rot90.xclip"
"Flip Elevation Data"                           f.xclip "m.flip.xclip"
}

/***** PaintPrintMenu *****/

Menu PaintPrintMenu
{
""      f.separator
"Use GRASS Shell for These Routines"    f.label
""       f.separator
"p.chart"                                       f.label
"p.colors"                                      f.label
"p.icons"                                       f.label
"p.labels"                                      f.label
"p.map"                                         f.label
"p.select"                                      f.label
```

```
}

/***** RasterMenu *****/

Menu RasterMenu
{
"Analyze"                                        f.pullright RasterMenu1
""                                               f.separator
"Import"                                         f.pullright RasterMenu2
"Export"                                         f.pullright RasterMenu3
"Convert"                                        f.pullright RasterMenu4
""                                               f.separator
"Develop"                                        f.pullright RasterMenu5
""                                               f.separator
"Report"                                         f.pullright RasterMenu6
""                                               f.separator
"Support"                                        f.exec "xgsupport &"
}

Menu RasterMenu1
{
"Single Cell"                                    f.pullright RasterMenu11
"Overlay"                                        f.pullright RasterMenu12
"Neighborhood"                                   f.pullright RasterMenu13
"Region"                                         f.pullright RasterMenu14
"Proximity/Reclass"                              f.pullright RasterMenu15
"Terrain"                                        f.pullright RasterMenu16
}

Menu RasterMenu11
{
"Calculate Data Clumps Volume"                   f.xclip "r.volume.xclip"
"Report Basic Raster Information"                f.xclip "r.info.xclip"
"Report Raster Statistics"                       f.xclip "r.report.xclip"
"Print Category Values List"                     f.xclip "r.describe.xclip"
"Generate Area Statistics"                       f.xclip "r.stats.xclip"
"Query Category Values/Labels"                   f.xclip "r.what.xclip"
"Tablulate Mutual Category Occurences"           f.xclip "r.coin.xclip"
}

Menu RasterMenu12
{
"Create Category Values Cross Product"           f.xclip "r.cross.xclip"
"Create Composite from Category Values"          f.xclip "r.patch.xclip"
"Use Inference Engine"                           f.xclip "r.infer.xclip"
"Develop Bayesian Expert System"                 f.xclip "r.binfer.xclip"
"Use Map Calculator"                             f.exec "xgmapcalc &"
""      f.separator
"Use GRASS Shell for These Routines"    f.label
""       f.separator
"r.weight.new"                                   f.label
"r.combine"                                      f.label
}

Menu RasterMenu13
{
"Perform Neighborhood Analysis"                  f.xclip "r.neighbors.xclip"
"Show Cost Between Locations"                    f.xclip "r.cost.xclip"
"Trace Flow Through Elevation Model"             f.xclip "r.drain.xclip"
```

```
"Generate Slope and Aspect Maps"               f.xclip "r.slope.aspect.xclip"
"Clump Data"                                   f.xclip "r.clump.xclip"
"Thin Non-Zero Cells Denoting Linear Features" f.xclip "r.thin.xclip"
"Grow Contiguous Areas by One Cell"            f.xclip "r.grow.xclip"
"Output Values Lying on Transect Lines"        f.xclip "r.transect.xclip"
"Calculate Distance Buffers"                   f.exec "xgbuffer &"
"Use Map Calculator"                           f.exec "xgmapcalc &"
}

Menu RasterMenu14
{
"Find Average Values in Values Map"            f.xclip "r.average.xclip"
"Clump Data"                                   f.xclip "r.clump.xclip"
"Calculate Data Clumps Volume"                 f.xclip "r.volume.xclip"
""      f.separator
"Use GRASS Shell for These Routines"    f.label
""       f.separator
"r.mask"                                       f.label
}

Menu RasterMenu15
{
"Calculate Distance Buffers"                   f.exec "xgbuffer &"
"Reclass Data in Raster Map"                   f.exec "xgreclass &"
}

Menu RasterMenu16
{
"Generate Watershed Subbasin Map"              f.xclip "r.basins.fill.xclip"
"Show Cost Between Locations"                  f.xclip "r.cost.xclip"
"Trace Flow Through Elevation Model"           f.xclip "r.drain.xclip"
"Create Elevation Map from Contour Map"        f.xclip "r.surf.contour.xclip"
"Perform IDW"                                  f.xclip "r.surf.idw.xclip"
"Perform Exhaustive IDW"                       f.xclip "r.surf.idw2.xclip"
"Generate Watershed Maps"                      f.xclip "r.watershed.xclip"
"Generate Slope and Aspect Maps"               f.xclip "r.slope.aspect.xclip"
"Clump Data"                                   f.xclip "r.clump.xclip"
"Perform Line-of-Sight Analysis"               f.xclip "r.los.xclip"
"Model Ballistic Trajectory"                   f.xclip "r.traj.xclip"
"Review Ammunition and Weapon Data Base"       f.xclip "r.traj.data.xclip"
"Calculate Data Clumps Volume"                 f.xclip "r.volume.xclip"
"Use Map Calculator"                           f.exec "xgmapcalc &"
}

Menu RasterMenu2
{
"Convert ASCII Raster Text to Binary Map"      f.xclip "r.in.ascii.xclip"
"Convert Raster Data to UTM-referenced Map"    f.xclip "r.in.ll.xclip"
""      f.separator
"Use GRASS Shell for These Routines"    f.label
""       f.separator
"r.in.sunrast"                                 f.label
}

Menu RasterMenu3
{
"Convert Raster Map to ASCII Text File"        f.xclip "r.out.ascii.xclip"
}
```

```
Menu RasterMenu4
{
"Create Binary Vector File from Thinned Raster" f.xclip "r.line.xclip"
"Extract Raster Edges and Convert to Vector"    f.xclip "r.poly.xclip"
"Thin Non-Zero Cells Denoting Linear Features"  f.xclip "r.thin.xclip"
}

Menu RasterMenu5
{
"Create Randomly Located Sites"                 f.xclip "r.random.xclip"
"Create Composite from Category Values"         f.xclip "r.patch.xclip"
"Create/Modify Raster Map's Color Table"        f.xclip "r.colors.xclip"
"Compress/Decompress Raster Files"              f.xclip "r.compress.xclip"
"Resample GRASS Raster Map Data"                f.xclip "r.resample.xclip"
"Rescale Range of Category Values"              f.xclip "r.rescale.xclip"
"Use Raster File Matrix Filter"                 f.xclip "r.mfilter.xclip"
}

Menu RasterMenu6
{
"Find Average Values in Values Map"             f.xclip "r.average.xclip"
"Calculate Data Clumps Volume"                  f.xclip "r.volume.xclip"
"Print Category Values and Labels"              f.xclip "r.cats.xclip"
"Tablulate Mutual Category Occurences"          f.xclip "r.coin.xclip"
"Output Covariance/Correlation Matrix"          f.xclip "r.covar.xclip"
"Report Basic Raster Information"               f.xclip "r.info.xclip"
"Print Category Values List"                    f.xclip "r.describe.xclip"
"Output Values Lying on User-Defined Lines"     f.xclip "r.profile.xclip"
"Report Raster Statistics"                      f.xclip "r.report.xclip"
"Generate Area Statistics"                      f.xclip "r.stats.xclip"
"Query Category Values/Labels"                  f.xclip "r.what.xclip"
}

/***** VectorMenu *****/

Menu VectorMenu
{
"Analyze"                                       f.pullright VectorMenu1
""                                              f.separator
"Import"                                        f.pullright VectorMenu2
"Export"                                        f.pullright VectorMenu3
"Convert"                                       f.pullright VectorMenu4
""                                              f.separator
"Develop"                                       f.pullright VectorMenu5
""                                              f.separator
"Report"                                        f.pullright VectorMenu6
}

Menu VectorMenu1
{
"Display Area/Perimeter Information"            f.xclip "v.area.xclip"
}

Menu VectorMenu2
{
"Convert ARC/INFO Data to GRASS Vector"         f.xclip "v.in.arc.xclip"
"Convert Vector from ASCII/DLG-3 to Binary"     f.xclip "v.in.dlg.xclip"
"Convert Files from DXF to GRASS Vector"        f.xclip "v.in.dxf.xclip"
"Import TIGER Files to GRASS Vector"            f.xclip "v.in.tiger.xclip"
```

```
"Convert Vector Map from ASCII to Binary"       f.xclip "v.in.ascii.xclip"
""      f.separator
"Use GRASS Shell for These Routines"    f.label
""       f.separator
"v.import"                                      f.label
}

Menu VectorMenu3
{
"Convert GRASS Vector Files to ARC/INFO"        f.xclip "v.out.arc.xclip"
"Convert Vector Map from Binary to DLG-3"       f.xclip "v.out.dlg.xclip"
"Convert GRASS Vector Files to DXF"             f.xclip "v.out.dxf.xclip"
"Convert Vector Map from Binary to ASCII"       f.xclip "v.out.ascii.xclip"
""      f.separator
"Use GRASS Shell for These Routines"    f.label
""       f.separator
"v.out.moss"                                    f.label
}

Menu VectorMenu4
{
"Convert Vector Map to Raster"                  f.xclip "v.to.rast.xclip"
"Convert Vector Pointer Data to Sites List"     f.xclip "v.to.sites.xclip"
"Clean Dead Lines"                              f.xclip "v.clean.xclip"
"Transform ASCII Vector Map Coordinate System"  f.xclip "v.transform.xclip"
"Trim Small Lines and Extra Points"             f.xclip "v.trim.xclip"
}

Menu VectorMenu5
{
"Create Map of User-Defined Grid"               f.xclip "v.mkgrid.xclip"
"Create GRASS File for USGS Quadrangle"         f.xclip "v.mkquads.xclip"
"Patch Multiple Vector Files"                   f.xclip "v.patch.xclip"
"Prune Points"                                  f.xclip "v.prune.xclip"
"Process Spaghetti-Digitized Vector Files"      f.xclip "v.spag.xclip"
"Attach Labels to Contour Lines"                f.xclip "v.cadlabel.xclip"
""      f.separator
"Use GRASS Shell for These Routines"    f.label
""       f.separator
"v.mkcontour"                                   f.label
"v.support"                                     f.label
"v.digit"                                       f.label
}

Menu VectorMenu6
{
"Print Information About Vector Map"            f.xclip "v.stats.xclip"
""      f.separator
"Use GRASS Shell for These Routines"    f.label
""       f.separator
"v.db.rim"                                      f.label
}

/***** SitesMenu *****/

Menu SitesMenu
{
"Analyze"                                       f.pullright SitesMenu1
""                                              f.separator
```

```
"Import"                                        f.pullright SitesMenu2
"Export"                                        f.pullright SitesMenu3
"Convert"                                       f.pullright SitesMenu4
""                                              f.separator
"Develop"                                       f.pullright SitesMenu5
""                                              f.separator
"Report"                                        f.pullright SitesMenu6
}

Menu SitesMenu1
{
"Perform Surface Generation"                    f.xclip "s.surf.idw.xclip"
""      f.separator
"Use GRASS Shell for These Routines"    f.label
""       f.separator
"s.menu"                                        f.label
"s.db.rim"                                      f.label
}

Menu SitesMenu2
{
"Convert ASCII File to Sites List"              f.xclip "s.in.ascii.xclip"
}

Menu SitesMenu3
{
"Convert Sites List to ASCII File"              f.xclip "s.out.ascii.xclip"
}

Menu SitesMenu4
{
"N/A"                                           f.label
}

Menu SitesMenu5
{
""      f.separator
"Use GRASS Shell for These Routines"    f.label
""       f.separator
"s.menu"                                        f.label
}

Menu SitesMenu6
{
""      f.separator
"Use GRASS Shell for These Routines"    f.label
""       f.separator
"s.menu"                                        f.label
"s.db.rim"                                      f.label
}

/***** History *****/

Menu HistoryMenu
{
"View/Edit Current History"                     f.history_edit
"Replay Current History"                        f.history_replay
"Clear Current History"                         f.history_clear
}
```

```
/***** HelpMenu *****/

Menu HelpMenu
{
"How to Use Help"      f.exec "xghelp -label
                              \"How to Use Help\" 01.instruc/01.01.help &"
"Open GRASS Topics"    f.exec "xghelp -label \"GRASS Topics\" &"
"Open GRASS Glossary"  f.exec "xgglossary &"
"Open GRASS Manual"    f.exec "xghelp -label \"GRASS Manual\" 17.manual/17.menu &"
}

/***** QuitMenu *****/

Menu QuitMenu
{
"Quit GRASS"                                  f.quit
}
```