--   --

Volume I

Khoros User's Manual

Chapter 1

GETTING STARTED

Introduction to the User's Manual

Primary Author(s):

John Rasure & Danielle Argiro

Printed: March 16, 1992

GETTING STARTED

A. INTRODUCTION

Khoros is a software environment for research which uses visual programming as a tool for software development for scientific visualization. The Khoros infrastructure consists of several layers of interacting subsystems. A user interface development system (UIDS) combines a high-level user interface specification with methods of software development and maintenance that are embedded in a code generation tool set. An interoperable data exchange format (VIFF) and algorithm library contain the application specific layer. The UIDS can be used to create three user interface styles: (1) a command line user interface, (2) a graphical, form/menu based user interface, and (3) the basic operations of a visual programming language. The result is an extensible visual programming environment.

The current library of over 260 routines has been developed to facilitate research in image processing, signal processing, pattern recognition, remote sensing, machine vision, and geographic information systems. Khoros is also a valuable teaching tool for signal and image processing.

B. SETTING UP YOUR ENVIRONMENT

In order for you to access the Khoros software from your login environment, the following issues must be addressed:

(1) Khoros environment variables must be defined.

(2) Your path must be set to include the Khoros binaries.

(3) Xdefaults for the Khoros system must be set by using the ".Xdefaults" file.

These are discussed in the following sections. Examples of the files described are included in the directory `$KHOROS_HOME/dotfiles`.

B.1 ENVIRONMENT VARIABLES

Environment variables for Khoros are typically put in a file called `.khoros_env` which should be created in the user's home directory and sourced at login. The most important environment variable, "KHOROS_HOME", indicates the directory where Khoros has been installed:

`% setenv KHOROS_HOME` *directory*

Another very important environment variable that is used by UNIX and Khoros is the "TMPDIR" variable. Khoros uses this directory as the location of all temporary files and images. It should be set to a large directory, as in:

`% setenv TMPDIR /usr/tmp`

Several of the maintenance and installation procedures (**kinstall** and **ksrcconf**) send mail to indicate their status. If you want this mail sent to a specific account, then include the following

line:
`% setenv KHOROS_MAIL` *user_name*

Programs will log their execution occurrence into a command log file if the "KHOROS_LOG" environment variable is set to a valid filename:
`% setenv KHOROS_LOG` *filename*
If you do not want command log files to appear at all, you may set KHOROS_LOG to /dev/null.

Programs will print out a warning message and a default value for each option that is not used if the "KHOROS_VERBOSE" flag is set. In general, this flag is set to 'no':
`% setenv KHOROS_VERBOSE no`

Specification of input files for Khoros programs may be shortened by use of the Keywords capability; this capability provides a shorthand for the specification of input files. The use of the Keywords capability is explained in detail in the last section of this chapter; on-line help can be accessed from the "Help" button that appears on the file browser. If desired, you may create your own Keywords file defining your own preferred shorthand to frequently-used input files, and set the "KHOROS_KEYWORDS" environment variable to it; otherwise, the Keywords shorthand is defined in the Keywords file:
`% setenv KHOROS_KEYWORDS $KHOROS_HOME/repos/Keywords`

When any new programs are added to the Khoros system, they must be installed in a Toolbox. A toolbox is simply a directory structure whose layout and contents mimic those of $KHOROS_HOME. A complete explanation of Toolboxes is contained at the end of Chapter 1 (Writing Programs) of the Khoros Programmer's Manual. The Toolbox file defines which toolboxes are accessable to you as a user; for example, if you want access to the "vfrog" program that is installed in the "amphibians" toolbox, you must have a Toolbox file containing an entry that references the "amphibians" toolbox. If desired, you may create your own Toolbox file defining your own preferred toolbox(es), and set the "KHOROS_TOOLBOX" environment variable to it; otherwise, the Toolbox(es) accessable to you are defined in the Toolbox file:
`% setenv KHOROS_TOOLBOX $KHOROS_HOME/repos/Toolbox`

The data transport mechanisms of Khoros allow you to specify input and output data for Khoros programs when the data is actually located on a different machine from which the Khoros programs in question are executed. The distributed computing capability of Khoros allows you to specify on which computer a particular Khoros program is to be executed. The data transport and distributed computing capabilities go hand in hand; together, they allow you to take advantage of all available computing resources and efficiently solve information processing problems.

For example, you might want to run the fast fourier transform program, **vfft**, on machine A, which has fast hardware and will be able to execute the cpu- intensive program in a short amount of time; however, you might want to get input data for **vfft** from machine B, which has more memory and is therefore where most of your large data sets are located. The "KHOROS_HOSTS" environment variable must point to the file which defines which machines may interact within the data transport mechanism; this ascii file simply lists all available computers on which you might execute Khoros programs or read/write data for Khoros programs. A computer MUST be listed in the Khoros_Hosts file before it can be used with data transport or distributed computing:
`% setenv KHOROS_HOSTS $KHOROS_HOME/repos/Khoros_Hosts`

Here is an example of the `.khoros_env` file. Create this file in your home directory, or copy the example file given in `$KHOROS_HOME/dotfiles/khoros_env`. Note the references to `$KHOROS_HOME/repos/Keywords`, `$KHOROS_HOME/repos/Toolbox`, and `$KHOROS_HOME/repos/Khoros_Hosts`. These three files may be used as examples of the Keywords, Toolbox, and Khoros_Hosts files; you may use the former two as they are, or you may copy them elsewhere and customize them for your personal use. Simply remember to update the "KHOROS_KEYWORDS" and "KHOROS_TOOLBOX" environment variables to point to the new locations of your customized files, as the Khoros system will continue to use the default files until you update these environment variables. Remember that you MUST customize the Khoros_Hosts file if you want to take advantage of the data transport and distributed computing capabilities of the Khoros system. The simple reason for this is that the default Khoros_Hosts file contains a list of our machines here at the University of New Mexico; obviously, this machine list will not be valid at your site.

```
#   ----------------------------------------------
#   This file must be sourced (source .khoros_env)
#   or included in your .login or .cshrc
#   ----------------------------------------------
#
#   This file contains default values for ALL
#   environmental variables used by KHOROS.

setenv KHOROS_HOME        directory_name
setenv KHOROS_MAIL        $USER
setenv KHOROS_LOG         $HOME/khoros.cmdlog
setenv KHOROS_VERBOSE         no
setenv KHOROS_KEYWORDS        $KHOROS_HOME/repos/Keywords
setenv KHOROS_TOOLBOX         $KHOROS_HOME/repos/Toolbox
setenv KHOROS_HOSTS       $KHOROS_HOME/repos/Khoros_Hosts
setenv TMPDIR             /usr/tmp
```

Important note: when you install patches, files in $KHOROS_HOME/repos will be overwritten. For this reason, if you have customized your "Keywords", "Toolbox", and/or "Khoros_Hosts" files for your personal use, we suggest that you move these files to another location outside the Khoros system.

B.2 MODIFICATIONS TO THE .cshrc FILE

For the Khoros system to work correctly, the Khoros environment file (describe above) must be sourced at login. This can be done by adding a source statement at the *top* of the ".cshrc" file. The Khoros binary directory must also be added to your search path. To indicate where Khoros X applications will be displayed, the DISPLAY environment variable must be set to your workstation. One way to accomplish these actions is to modify your `.cshrc` file with the following changes.

```
# Source the .khoros_env file to set environment variables:
# (This must be included before the "set path" command!)
source $HOME/.khoros_env

# Modify your path to include $KHOROS_HOME/bin:
set path=(. ~{$USER}/bin $KHOROS_HOME/bin ...)

# Set the X Window environment variable DISPLAY:
# (this is optional)
setenv DISPLAY workstation:0
```

B.3 XDEFAULTS

The ".Xdefaults" file is read by X Windows-based applications during startup to customize the appearance or characteristics of their windows. See the example in `$KHOROS_HOME/dotfiles/Xdefaults`. This is a very long .Xdefaults file; however, the majority of it is devoted to controlling the colors that appear on the various Khoros applications. You need not be concerned with this unless you would like to customize the colors for Khoros applications for your own use. If you would like to customize the colors, it is easiest to edit this file and do a global change. For example, if you use the *vi* visual editor, and want all subform buttons to appear in orange instead of royal blue, you can do:

```
:1,$ s/royal blue/orange/g
```

If you use the *emacs* visual editor, the global change can be made with:

```
<esc> x replace-string
```

Remember, any time you change your .Xdefaults file and you want to see that change reflected when you run X Windows-based applications, you must execute:

```
% xrdb {desired .Xdefaults file}
```

If you have problems when running Khoros on a monochrome screen, look at the `$KHOROS_HOME/dotfiles/Xdefaults.mono` file for examples of lines you might try changing, or simply use this Xdefaults file instead.

C. X WINDOWS

Since Khoros runs under the X Windows (X11R4) system, it is helpful to have an understanding of X Windows before attempting to use Khoros. For those that would like to learn more about X Windows, we recommend the "X Window System User's Guide" by Valerie Quercia and Tim O'Reilly. In addition to this book, O'Reilly and Associates, Inc. have published volumes on X Protocol, Xlib, the X Toolkit, and XView, some or all of which may prove valuable to the interested reader. A brief description of X Windows can be obtained with the Unix **man** command:

```
% man X
```

The details of managing the windows, such as interpreting mouse and keyboard commands, moving and resizing, are actually carried out by a program called a *window manager*. On many systems the window manager used is **twm**. Another popular window manager (used with SUN's) is **olwm**, or Open Windows window manager. Use the `% man` command to find out more about your window manager. Some window managers have automatic window placement, while others use manual window placement; some window managers "decorate" the tops of windows, while others do not. The operations that you must perform to resize a window, refresh the screen, and so forth are dictated by your window manager. If your site supports more than one window manager, you may have a choice as to which one is the most suitable for you. As the window manager will affect the operation of Khoros programs, you should be reasonably comfortable with your window manager before using the Khoros system. It should be noted that Khoros was developed primarily under the **twm** window manager; most Khoros programs have also been tested under **olwm**. *Never* attempt to run more than one window manager at the same time.

Once you have logged onto a system and are using X Windows, you may need to tell the system where you want Khoros images to be displayed. This is done by setting the environment variable DISPLAY. The command:

`% setenv DISPLAY` *workstation*:0

tells X Windows and Khoros to send images and graphics to the primary screen (screen 0) of the named *workstation*. If you have originally logged in to one machine, and then **rsh** or **rlogin** to a second machine on which you are going to be running Khoros applications, you must execute:

```
% xhost {second machine}
```

on the original machine before going to the second machine and executing the `setenv DISPLAY` command given above for the display of the second machine.

D. USING KHOROS ROUTINES

D.1 COMMAND LINE USER INTERFACE

The Khoros routines can be invoked as individual programs from the command line, using a standard, Unix-like interface. By convention, all of the program options are preceded by a minus sign and a flag type. For example, the flag **-i** typically indicates an input file and **-o** an output file. Thus, the program to shrink the image contained in file *infile* by a factor of two and store the result in *outfile*, is invoked with:

```
% vshrink -i KHOROS_HOME/data/images/ball.xv -o outfile -s 2
```

To display an image file on the screen, you can use:

```
% putimage -i KHOROS_HOME/data/images/ball.xv
```

If you don't know how to use a program, you can get a brief description by typing the program name followed by the **-U** usage flag. This will produce a short description and a list of the required and optional arguments. For example:

```
% vshrink -U
```

produces

```
vshrink :

      Shrink an image by skipping pixels (subsampling)

      -i  input image  (infile)
      -o  output (shrunk image)  (outfile)

      [-s]  shrink scale factor (integer, > 0) [2]

      [-V] Gives the version for vshrink
      [-U] Gives the usage for vshrink
      [-P] Prompts for command line options
      [-A [file1]] Creates the answer file called vshrink.ans or file1
      [-a [file1]] Uses vshrink.ans or file1 as the answer file
```

If you would prefer to be prompted for the arguments one by one, use the **-P** flag. The **-A** option creates an answer file containing the parameters you have specified; referencing the program again with **-a** flag reads any parameters not specified on the command line from the answer file (useful if the same parameters are to be used repeatedly).

To get a detailed description of a Khoros routine, use the **vman** command, which is patterned after the Unix **man** command:

```
% vman vshrink
```

If you would like to know which routines are associated with a specific action or idea, use the **vman** command with the **-k** *keyword* option. Thus

```
% vman -k shrink
```

produces the listing

```
vshrink(1) - Shrink an image by skipping pixels (subsampling)
lvshrink(3) - Shrink an image by skipping pixels (subsampling)
```

which tells us that there is a program, **vshrink**, to shrink an image, and a library routine, *lvshrink()*, that may be called by a program to perform the shrinking operation.

Stdin can be used for any input file but must be specified by using a "-" for the file name. Stdout can also be used for any output file. In this way, the output of one program can be connected to the input of another using Unix pipes. Thus:

```
% vshrink -i image.xv -o - -s 2 | putimage -i -
```

invokes **vshrink** to shrink the image in *image.xv*, and passes the result to the display routine **putimage** to display it on the screen.

Stderr can be used for an ASCII output file but must be specified using "#" for the file name.

D.2 GRAPHICAL USER INTERFACE

Khoros includes several major application programs that always have a graphical user interface, in addition to their command line user interface. Among others, these include an animation/image sequencing program, an image display/manipulation package, an image warping application, an interactive imagery/elevation display package, a 2D plotting package, and a 3D plotting package. To begin use of these programs, you may want to skip forward to the section labelled, "APPLICATION DEMOS" to see demos of these applications. In addition, you might try the following commands as an introduction to these programs.

```
% animate -i $KHOROS_HOME/data/sequences/eye/eyes
```

To see the animation sequence of 60 frames (hit the button labeled ">>" to start, "Stop" to stop, "Quit" to end the program).

```
% editimage -i $KHOROS_HOME/data/images/feath.xv
```

To see an image displayed.

```
% xprism2 -i11 $KHOROS_HOME/data/plot_functions/example2.1
```

To plot a pre-defined set of data points in two dimensions.

```
% xprism3 -i11 $KHOROS_HOME/data/plot_functions/example3.0
```

To plot a pre-defined mathematical function in three dimensions.

D.3 VISUAL LANGUAGE INTERFACE

All the routines available in the Khoros image processing library (ipl), digital signal processing library (dspl), and file format conversion library can be accessed using the command line user interface (CLUI). However, most users prefer the graphical user interface (GUI) provided by the visual programming language, **cantata**. Image and signal processing tasks are often naturally expressed as block diagrams or data flow graphs. By providing a programming environment in a similar format, **cantata** assists in the rapid prototyping of applications in those fields.

To create a visual program using **cantata**, you begin with a blank **cantata** workspace. Selecting the desired Khoros routines to perform various tasks, you then create iconfied representations of the Khoros routines, called *glyphs*. Next, you connect the glyphs together with lines that go from one glyph's output arrow to another glyph's input arrow. This is the equivalent of running the first Khoros program from the command line, saving its output in a temporary file, using the temporary file as input to the next program, and so forth. A complete **cantata** workspace is a visual program made up of glyphs representing Khoros programs, with lines representing the flow of data between routines. The entire program can be run using a "Run" button located on the **cantata** form, or the program may be run step by step by clicking on the little "on/off" switches that appear at the upper left hand corner of each glyph representing an executable Khoros program.

At any time, a **cantata** workspace may be saved as an ascii file. Later, **cantata** may be run, restoring the saved workspace, so that you may continue with your visual program where you left off before.

If you would like to see a simple, easy-to-understand example of a **cantata** workspaces, you might try executing the journal playback session of **cantata** mentioned in the section labelled, "APPLICATION DEMOS", or you might execute:

```
% cantata -restore $KHOROS_HOME/repos/workspace/examples/example1.Z
```

The objective of this first visual program is to enhance a dim image. We will go through the example step by step, so that you can follow along using **cantata** while reading this section. The input image is represented by the first glyph, labelled, "images". On this glyph, click on the button with the picture of the forms (meant to represent the graphical user interface). The glyph will open up into the corresponding graphical user interface; in this case, the Input Data subform with the Images pane displayed. You can see that the input image selected is "spine.xv". This will be the input image for our visual program. Now click on the button at the upper left hand corner of the subform, labelled "Glyph". The Input Data subform will turn back into a glyph, and go back to its place on the **cantata** workspace.

Notice that the "images" glyph's output is connected to two other glyphs: the "put_update" glyph and the "vconvert" glyph. This means that the spine.xv image will be serving as input to two Khoros routines: **putimage** and **vconvert** (usually, the glyph name is identical to the name of the program that it represents - however, the **putimage** program is a special case: it has two glyph representations, "put_update" and "put_multi", which are used differently). First, click on the "put_update" glyph's "run" button, which appears at the top right and has a picture of an electrical switch. This will run the **putimage** program with spine.xv as the input. Since spine.xv is not of data type BYTE, you will be prompted to convert the image. Click on "Cont" to indicate "Yes". The spine.xv image will then be converted to type BYTE and displayed by **putimage**. Notice that the image of the spine is very dim; most of its colors are on the dark end of the grey scale spectrum.

Now, go to the glyph labelled, "vconvert". It will also have spine.xv as its input, as indicated by the lines representing data flow. Click on its run button in order to call the **vconvert** program to convert the data in spine.xv to type BYTE in preparation for the next step.

The **vhstr** program will do a *histogram stretch* on the colors in the input image. That is, it will attempt to even out the colors in the image, so that instead of being all bunched together at the dark end of the grey scale spectrum, they will cover a larger number of colors ranging from black to white. To run the **vhstr** program on the converted spine.xv data, click on the "run" button of the "vhstr" glyph. To see the results of the histogram stretch, click on the "run" button of the last "put_update" glyph. This will call **putimage** once again to display the resulting image. Note that this image is much brighter than the original; vertibrae are easy to distinguish, and much more detail is apparent. So, our example visual program was a success!

Delete the images displayed by **putimage** by clicking the mouse inside them, or clicking the mouse on the "on/off" switch that appears to the upper right hand corner of the **put_update** glyph. If desired, you can repeat the entire process quickly by hitting the "RESET" button which appears to the left of the **cantata** form, and then clicking on the "RUN" button that appears right above it. The visual program will repeat itself all at once, not step by step as in the first try. Glyphs can be moved about by clicking on them with the mouse and moving them while holding the mouse button down.

By clicking on the "Workspace" subform button on the **cantata** form, you can set environment options for **cantata** - for instance, you may prefer small glyphs instead of large ones, you may wish to indicate when data is available at inputs and outputs, you may wish **cantata** to echo program execution to your tty, or you might want to use the interactive glyph placement option. These and other options may be set as desired.

There are two other example workspaces in the `$KHOROS_HOME/repos/workspace/examples` directory that you are welcome to try. We will not go over them here, but the README file in that directory will explain the objective of each workspace. You are encouraged to experiment with these examples of **cantata**.

For more information about the **cantata** program, you can execute `% vman cantata`; Chapter 2 of the Khoros User's Manual is recommended for further understanding of the **cantata** visual language.

E. APPLICATION DEMOS

E.1 SEEING A JOURNAL PLAYBACK

All Khoros *xvroutines*, including **animate, composer, editimage, viewimage, warpimage, xprism2**, and **xprism3** have a built-in journal record / journal playback capability that is provided via the graphical user interface. If you add a new xvroutine to Khoros, it will also have the journal record / playback ability (see Chapter 1 of the Khoros Programmer's Manual for more information on xvroutines and journal playback). This journal record/playback capability is available so that you may run the demos that are provided for you in `KHOROS_HOME/repos/journal`, and create your own journal sessions, if desired. Furthermore, the journal record / playback capability of these programs is what makes them viable target programs for the Khoros collaboration program, **concert** (see Chapter 9 of the Khoros User's Manual).

When playing back a journal session, a small "Journal Control" widget will appear. Since the journal playback session *grabs your keyboard*, this Journal Control widget reminds you of the simple control commands that the journal playback mechanism understands. The command keys will work regardless of where the mouse pointer is located when they are hit; remember, your keyboard has been grabbed, so you will not be allowed to provide keyboard input to any window that may currently be displayed until the journal recording is finished. The following are the three simple commands that journal playback understands:

h or H: Halts the journal playback session.

c or C: Continues the journal playback session.

q or Q: Quits the journal playback session, and allows you to continue execution of the application on your own, at whatever point the journal playback left off.

There are predefined application demos for all the major Khoros applications. All of these demos are run by executing the desired application with the [-jp {record file}.jp] option, where "jp" stands for "journal playback". In addition, any other options that were used when the journal file was created must also be used when the journal recording is played back. The following is a list of commands that may be run in order to see the predefined application demos that are provided with the Khoros 1.0 release. Journal record files are always named with the postfix, ".jp".

```
% animate -i $KHOROS_HOME/data/sequences/bush/bushes
          -jp $KHOROS_HOME/repos/journals/animate.jp

% cantata -jp  $KHOROS_HOME/repos/journals/cantata.jp

% composer -jp  $KHOROS_HOME/repos/journals/composer.jp

% editimage -jp $KHOROS_HOME/repos/journals/editimage.jp

% viewimage -i $KHOROS_HOME/data/images/feath.eye
            -e  $KHOROS_HOME/data/images/flow.eye
            -jp  $KHOROS_HOME/repos/journals/viewimage.jp

% warpimage -jp  $KHOROS_HOME/repos/journals/warpimage.jp

% xprism2 -jp $KHOROS_HOME/repos/journals/xprism2.jp
```

```
% xprism3 -jp $KHOROS_HOME/repos/journals/xprism3.jp
```

In some cases, the fact that journal playback grabs the keyboard may present a problem. If you need to have control of the keyboard while journal playback is running, you must use the [-jg 0] or [-jg false] (where "jg" stands for "journal grab") option in addition to the [-jp {journal file}] option. When this option is used, the Journal Control Widget will not appear. It is mainly used for debugging purposes by the Khoros group, but some people may find it useful.

E.2 MAKING A JOURNAL RECORDING

You may wish to create your own application demos using the journal recording mechanism that is built in to all Khoros xvroutines. The journal recording / playback mechanism is machine independent, so if you want to record a session on one machine and play it back on a differing architecture, this will not pose a problem. If, however, the journal recording was made with a particular font, and is played back with a different font, the graphical user interface may have trouble sizing itself properly. While this mars the physical appearance of the application being played back, it will not harm the functionality of the recording.

To make a journal recording of a Khoros xvroutine, simply use the [-jr {record file}] option that is available with all Khoros xvroutines. For example, say that we would like to make a journal record file of **xprism**. We would execute the following:

```
% xprism2 -jr my_xprism2_demo.jp
```

This will create the journal recording file, "my_xprism2_demo.jp". To play back the recorded session, we execute:

```
% xprism2 -jp my_xprism2_demo.jp
```

Remember that any other options you used when recording the session *must also be used when playing it back*. Therefore, if you record a session with **editimage** as:

```
% editimage -i $KHOROS_HOME/data/images/lizard.xv  -jr editimage_demo.jp
```

You *MUST* play back the session with:

```
% editimage -i $KHOROS_HOME/data/images/lizard.xv  -jp editimage_demo.jp
```

Attempting, for example, to execute a journal playback of **editimage** without the input file that the journal recording was made with:

```
% editimage -jp editimage_demo.jp
```

Will not work! We cannot predict exactly what the results will be, but we can guarantee that they will not be what you recorded. For this reason, when making journal recording sessions, it is recommended that you use as few command line arguments as possible, thus making it less likely to forget what they were when you play back the session.

F. MAKING BUG REPORTS

As with any system approaching the size of the Khoros system, you can expect to find some bugs. We already are aware of several bugs; these can be found in $KHOROS_HOME/repos/BugList. If you find a bug in the Khoros system, please look through this file first to see if it is one of the bugs that we already have on our list. We *strongly* encourage you to report any bugs that you may find that are not listed in the BugList. ALL BUGS are to be reported via *email*, to `khoros-bugs@chama.eece.unm.edu`. NOTE: BUG REPORTS ONLY are to be mailed to this address. If you have comments, questions, or other feedback, please see the next section. There is a bug-reporting template, `$KHOROS_HOME/repos/BugReport`, that we hope you will use. If you do not use the template to report the bug, *please include the following information:*

1) The program or library routine containing the bug
2) A clear and concise description of the problem. If the problem was experienced when using **cantata**, a saved workspace that shows the bug is very helpful, along with the explanation.
3) Step-by-Step instructions for someone else to follow in order to reproduce the bug
4) The machine architecture that you were on when you experienced the bug
5) The operating system that you were using when you experienced the bug
6) The version of X Windows that is installed on your system (if not X11R4)
7) The window manager that you were using at the time

Please remember that all the above information is imperative for a bug report, so that members of the Khoros group will be able to re-produce the bug and fix it.

G. QUESTIONS? COMMENTS? COMPLIMENTS? COMPLAINTS?

For general discussion purposes, getting answers to questions that are not addressed in the Khoros Manuals, writing comments, or giving constructive criticism, we have an open-format (unmoderated) mailing group. The name of this mailing group is `khoros@chama.eece.unm.edu`, and was created due to popular demand. In addition, it helps us to separate actual bug reports (which, as indicated above, should go to `khoros-bugs@chama.eece.unm.edu`) from other mail.

If you would like to become a member of the Khoros mailing list, send your request to `khoros-request@chama.eece.unm.edu`. *PLEASE:* only requests to get on or off the Khoros mailing list should go to this address. Other mail is very likely to get lost or be deleted.

Once a member of the Khoros mailing list, feel free to write to `khoros@chama.eece.unm.edu`. If you have a question, and it is not answered in the Khoros Manuals or Khoros online documentation, you are invited to mail your question to this address. If the Khoros group cannot answer your question immediately, it is likely that another Khoros user may be able to (there are well over 300 people on the mailing list already). Constructive criticism, suggestions, or open-format discussions are also welcome at `khoros@chama.eece.unm.edu`. Of course, everyone is invited to join the mailing list, but if you use the Khoros system often, we especially encourage you to join.

The Khoros mailing list sometimes does get alot of traffic; some users may not want to receive the amount of mail that is frequently associated with the Khoros mailing list. For this reason, we have recently started our own news group, **Comp.soft-sys.khoros**. If you are a reader of network news, we encourage you to participate in our newsgroup. Note that all letters to khoros@chama.eece.unm.edu are forwarded to Comp.soft-sys.khoros; (the opposite is not true) thus, the Comp.soft-sys.khoros news group offers an alternative way of keeping up with the Khoros community if you do not wish to be a part of the mailing list.

H. ADDITIONAL BACKGROUND

For most first-time Khoros users, the previous sections will provide enough information to get started using Khoros. However, if you were a user of Khoros Beta and are already familiar with the system, or if you simply would like to know more about additional features that can be used with the major Khoros applications, the following sections will provide additional information that you can take advantage of to customize your environment, and use the Khoros software more efficiently.

H.1 THE FILE BROWSER AND KEYWORDS CAPABILITY

The Khoros Browser provides an alternative to typing in filenames for input from an InputFile selection that appears on the graphical user of any Khoros program. The Khoros Browser can be used in two ways: as a file/directory browser, or as a browser for keywords. When the Khoros browser is used as a file browser, one uses the mouse to move about the directory structure to find the desired input file; when it is used as a keyword browser, one uses the mouse to move about the keyword structure to find the desired file.

USING THE FILE BROWSER

The default mode for the Khoros Browser is to act as a File Browser. The first item in the list of the file browser is the word, "-- keywords --". By selecting this item, you change the Khoros browser to the second mode, which is the Keyword Browser; we will talk about this in the next section. The rest of the list is the contents of your local directory; to enter an input file that is located in your local directory, simply select the desired filename. The browser will go away, and the parameter box of the InputFile selection from which it originated will be filled in with the selected filename. Using the file browser, you may also move about the directory structure by clicking on "../" to move up in the directory structure, or by clicking on the name of the desired directory to move down in the directory structure. When the item you select from the browser is the name of a file, the entire path to that file prepended to the filename will become your input file; the file browser will go away, and the file you selected will appear in the InputFile selection.

Alternatively, one may also type the entire directory path directory into the text box at the bottom of the file browser, following the directory path with a carriage return. The file browser will immediately update the list with the contents of the specified directory. Notice that the directory box at the bottom of the file browser continually updates to the current pathname, so one always knows where the current directory is.

USING THE KEYWORD BROWSER

By selecting the first item of the file browser, "-- keywords --", you cause the Khoros Browser to change from being a file browser to being a keywords browser. The keywords browser provides a convenient shorthand for long pathnames to input files; furthermore, once the controlling Keywords file is set up, one need not remember the location of an input file - only it's keyword.

The keywords used by the Khoros system are determined by the Keywords file, located in KHOROS_HOME/repos/Keywords. Take a moment now to look at this file. Each entry in the Keywords file consists of a single line of text: the keyword or keyword set, followed by the path that the shorthand represents. The first entries are single keywords that are shorthand notations for very frequently used files. For instance, ":feath" is a shorthand notation for "KHOROS_HOME/data/images/feath.254.xv". Once this entry is established in the Keywords file, one may use ":feath" anywhere an input file is needed, in either a command line user interface or a graphical user interface. For instance, one might type:

```
% editimage -i :feath
```

instead of:

```
% editimage -i KHOROS_HOME/data/images/feath.xv
```

Or, on the Input/Output pane of editimage, one might enter ":feath" in the parameter box labeled, "Khoros image file as INPUT:". Note that when *typed by the user,* a single keyword MUST be preceded by the ":" symbol to distinguish it from an actual file in one's local directory.

Finally, one might use the Keyword browser, and select "feath" from the Keywords list. Again, the result is to input the image of the woman with the feathered hat.

Keywords may be sub-divided as desired. For instance, in the `KHOROS_HOME/repos/Keywords` file, "plot3D:example3.0" is defined to be a shorthand for `KHOROS_HOME/data/plot_functions/example3.0`. Subdividing keywords allows us to conceptually group input files as desired; in this case, the "plot3D" group has 5 elements -- example3.0 through example3.4. In addition, subdivision of keywords causes the Keywords Browser to mimic the operation of the File Browser.

The meaning of ":" following a keyword in the Keywords Browser is equivalent to a "/" following a filename of a file in the File browser; ie, when you click on this item, the current list will be replaced by a list of those items that are subordinate to the first item. Our Keywords file defines the "plot2D" keyword as having 5 subordinate keywords: example2.0 - example2.4. Thus, when we click on the "Plot2D:" entry of the Keywords Browser, we are presented with a list of those 5 subordinate keywords. Now, clicking on "example2.4" causes the Keyword Browser to expand the Keyword "path" to "plot2D:example2.4", which in turn is defined in the Keywords file as the shorthand representing `KHOROS_HOME/data/plot_functions/example2.4`.

The next time the Khoros Browser is used, it "remembers" that the last operation was to use the Keywords Browser, under the "Plot2D" keyword category, so it goes back to that "location" - again, we are presented with the list of example2.0 - example2.4. By clicking on "<==" (the equivalent of "../" in the File Browser) we go "up" to the original Keywords list. We may now choose another keyword ("moon", for instance, is the shorthand to

KHOROS_HOME/data/images/moon.xv) or we may again click on "<==". Since we are now at the "top" level of the Keyword Browser, there is no where else to go, except back to the File Browser.

While complicated to explain, the use of Keywords is very trivial after only a little practice; users are encouraged to take advantage of this convenient and time-saving shorthand. Furthermore, the Keywords file found in KHOROS_HOME/repos is only meant as a model; the user is encouraged to create their own Keywords file, to personalize the operation of keywords. When creating your own Keywords file, be sure to follow the syntax rules carefully:

1) all single keywords must be preceded with a ":".
2) keywords may be subdivided as many times as desired but remember: if you subdivide your keywords too much, your shorthand is no longer short!
3) each keyword must be followed by some white space, and then its full path definition.
4) each keyword/path pair must appear on a single line
5) comments are preceded by a "#" in the first column.

Now that you have created your Keywords file, you must tell the khoros environment to use it. You do this by going to the .khoros_env file in KHOROS_HOME/.khoros_env and changing the line that says,

```
setenv KHOROS_KEYWORDS            $KHOROS_HOME/repos/Keywords
```

to:

```
setenv KHOROS_KEYWORDS            $USER/Keywords file
```

Note that the .khoros_env file must be sourced (source .khoros_env) or included in your .login or .cshrc file. From now on, you can use your own keywords to your own best advantage.

H.2 DATA TRANSPORT / DISTRIBUTED COMPUTING

"Data transport" refers to the method used to transfer data between processes. Data transport mechanisms can be local or remote. Local transport mechanisms include shared memory, files, pipes, and streams; remote transport mechanisms include sockets, tli, and rpc. With the use of remote data transport, the ability to get input from and output to remote machines is implemented, and distributed processing is made possible.

"Distributed processing" means the ability to specify remote machines on which to execute individual Khoros programs. The capability to do distributed processing is implemented via employment of the remote data transport mechanisms. With distributed processing, one needs a method to execute jobs remotely, as well as a mechanism to transport data back & forth from the remote machine. A remote daemon, "phantomd", is started on the remote machine, takes requests to execute a job, and transports data involved with that job using the remote data transport mechanisms.

Note that Patch 4 to Khoros implements a "beta" version of data transport / distributed processing; the capability will be refined, improved and extended in future Khoros releases. One limitation of this first version of data transport / distributed processing involves whether or not a transport mechanism provides *permanent data storage*. By their nature, some transport mechanisms provide permanent data storage, while others do not. Permanent data storage is offered by transport mechanisms such as shared memory and files; it is not provided by sockets, pipes, or streams. When permanent data storage is not available, the scheduler used by *cantata* will be unable to execute a network of glyphs unless it starts *at the very beginning*, where initial input is provided in a permanent form. Thus, when using distributed processing, which is implemented via a non-permanent data transport mechanism (sockets) you must click on the "RESET" button and run the network completely through every time. We realize that this limitation is inconvenient and restrictive; in future versions of Khoros, the scheduler used by cantata will be improved so as to eliminate this problem.

The transport mechanisms that are (at least partially) supported by individual Khoros vroutines are:

shm: Shared Memory (local transport / permanent storage)

file: Standard Unix file (local transport / permanent storage)

pipe: Standard Pipes (local transport / no permanent storage)

stream:
Named stream/pipe (local transport / no permanent storage)

socket:
Socket transport (remote transport / no permanent storage)

Transport mechanisms used by the cantata visual language are:

shm: Shared Memory (local transport / permanent storage)

file Standard Unix file (local transport / permanent storage)

socketSocket transport (remote transport / no permanent storage)

In the future, other transport mechanisms that may be implemented include:

sunvm:
Sun's Virtual memory (permanent storage)

tli: Sys V Transport Layer Interface

sunrpc:
Sun's remote procedure call

apollorpc:
Apollo's remote procedure call

machvm:
Mach's Virtual Memory

Your computer architecture will dictate which, if any, of these transport mechanisms are available for your use. Note that data transport and distributed processing will not be available on some computer architectures. Also be aware that this "beta" version of data transport in cantata does have limited capability; some transport mechanisms may not always be used even when most appropriate for the situation.

The data transport and distributed processing capability may be taken advantage of either from the cantata visual language, or from individual command line executions of Khoros programs. However, before distributed processing may be utilized, the KHOROS_HOSTS environment variable must be set properly, and the ".rhosts" or ".rhosts.bk" file must be made available at the top level directory of the account on which you are running Khoros. Also, to enable distributed processing, a Khoros "phantom" daemon must be started on each machine which is to execute jobs remotely.

The KHOROS_HOSTS environment variable simply specifies a file (by default, ".khoros_hosts") in which all computers which are available for remote data transport are listed.

In addition to setting the KHOROS_HOSTS environment variable properly, it is essential that you have a ".rhosts" file set up properly before you can use the distributed processing capabilities of Khoros. The ".rhosts" file contains a list of host computers. This file allows a user who has an account on the local machine to log in from a remote host without suppling a password; more importantly with respect to the use of distributed processing, it allows the user to start a process on that remote machine. To simplify matters, it is easiest to set up the ".rhosts" file and the ".khoros_hosts" file at the same time. They will look identical, except that the ".rhosts" file will have the addition of the applicable login name appearing after each machine name. Note: some sites do not allow ".rhosts", so you can use ".rhosts.bk", and the "phantom" daemon will move ".rhosts.bk" to ".rhosts" temporarily.

Once the KHOROS_HOSTS environment variable is set, and the ".rhosts" file is added to the top level of your account, there are two ways you may use the data transport / distributed processing capabilities of Khoros. The first way is via the **cantata** visual language.

When data transport / distributed processing is to be used via **cantata**, you must first use the "Workspace" pulldown menu to map the "Attributes" subform. On the bottom left of the "Attributes" subform is the option, "Enable Remote Execution". Set this logical to "Yes". Now, all glyphs representing Khoros programs will appear with a network symbol in the middle. Clicking on this network symbol will bring up a list of machines. This list of machines is called the network browser, and the contents of the network browser is dictated by the machines that you listed earlier in your ".khoros_hosts" file (or other file specified by KHOROS_HOSTS). If there is not a ".khoros_hosts" file, the initial list of hosts will only include the local machine. You can interactively add machine names via the network browser. Selecting the desired machine will cause that program to be run on the specified machine.

Before programs can be executed on remote machines, a daemon must be started on that remote machine to run Khoros programs on behalf of the local machine. Cantata will attempt to start these daemons for you, automatically; however, if cantata has problems starting these daemons, you may have to start them yourself. When data transport / distributed processing is to be used via the command line, it is always necessary to start the distributed processing daemons on the remote machines that are to be used. Starting the daemons is a simple process; it can be done in one of two ways. The first way is to log on to each machine on which programs will be executed, and run the **phantomd** program directly. Suppose we are on machine A, and will be using the distributed processing capability to execute programs on machines B, C, and D. Before attempting to execute programs on the three remote machines, we must log on to each of machines B, C, and D, and execute:

```
 % phantomd
```

The second way to start the distributed processing daemons on the desired remote machines is to use the **phantom** shell script on the local machine, once for each remote machine that is to be used for program execution. In this case, we would execute, on machine A:

```
% phantom machineB
% phantom machineC
% phantom machineD
```

Note that when distributed processing is to be used via **cantata**, **cantata** will remote shell to the machines in question, and execute the **phantomd** program for you. However, if **cantata** fails for some reason to start the daemons properly, you will have to use one of the two above-mentioned methods to start the daemons on the desired remote machines yourself. In any case, it is required that both the KHOROS_HOSTS environment variable and the ".rhosts" file are set properly before the daemons can be started. Daemons started in this way have a 1/2 hour time-out limit set. Thus, if more than one half hour elapses before distributed processing is used, you must re-start the daemons again. If you would like to change the time-out limit, you may use the [-timeout *time*] option to either **phantom** or **phantomd**, where *time* is provided in minutes.

There are two conventions which dictate what type of data transport mechanism is to be used, and whether a routine should be executed on a remote machine. If no data transport mechanism is explicitly specified, the Khoros transport and distribution routines will negotiate the transport mechanism automatically. However, if you would like to dictate the data transport mechanism yourself, then you must follow the two transport specification conventions. The first convention is used to specify a data transport mechanism:

identifier=token

Where *identifier* is one of the data transport mechanisms listed earlier, such as "shm", "file" or "socket".

The *token* is the type of identifier for that transport. For a file, it is simply the filename. For shared memory, it is the shared memory key. For a pipe, it is the input & output file descriptors, as in "pipe=[3,4]". For a socket, it is the number of the socket, as in "socket=5430".

The second convention is used to specify a remote machine for distributed processing; it specifies where a token is located. For instance, if you want to retrieve a regular file from a remote machine, then the syntax is:

file=filename@machine

or

filename@machine

This causes the Khoros I/O routines to look thru the internal list of data transport mechanisms and select the first available remote transport mechanism. In the currently available list, this will be "socket". If a specific remote transport mechanism is desired, then the following should be used:

filename@socket=machine

Multiple machine routing is also allowed. This means that if you cannot directly access a file from your local machine, but you can access the file via the machine "gateway", then you may use the following syntax to retrieve the file:

filename@machine@gateway

Almost all combinations are allowed, but it is important to remember that *local transport mechanisms CANNOT be used with distributed processing*. For instance, the following specification will result in an error:

filename@stream=machine

H.3 USE OF DIFFERENT FONTS & COLORS

There are three ways to specify fonts with Khoros application programs, and one way to specify colors for use with Khoros applications. Let us discuss fonts first. A specific font may be specified for the entire graphical user interface of a Khoros xvroutine by using the [-fn] option that is available with all Khoros xvroutines, or by specifying the desired font in the .Xdefaults file.

H.3.1 CHANGING THE FONT FOR AN ENTIRE APPLICATION

The first two ways of changing the font, which will change the font for ALL fonts in **preview**'s graphical user interface, for instance, to kana14:

```
% preview -fn kana14
```

Alternatively, you may edit your .Xdefaults file, and add the line:

```
preview.font:    kana14
```

If you want *all* Khoros xvroutines to appear in a particular font, you may use a wildcard for the font specified in the .Xdefaults file, for example:

```
*.font:    fixed
```

This will make the graphical user interface of ALL Khoros xvroutines appear in the fixed font. Unfortunately, it will also make all other X Windows applications come up with a fixed font. At this point, there is no easy solution to this problem, but we are working on a better approach for Khoros 2.0. Now, if you want specific programs to appear in a diffent font, you can add each one specifically. For example, suppose you want all Khoros routines to appear in the fixed font except **xprism3**, which you want to see in 8x13bold. You would follow the line above with:

```
xprism3.font:    8x13bold
```

This line will over-ride the previous wildcard statement for the xprism3 font. Alternatively, you could have specified on the command line:

```
% xprism3 -fn 8x13bold
```

Since the [-fn] option always over-rides any specifications found in the Xdefaults file. NOTE: we were unable to get the SGI architecture to get specificattions from the .Xdefaults file; if you are running Khoros on an SGI, you will have to always use the [-fn] option to change fonts.

H.3.2 CHANGING THE FONTS & COLORS WITHIN AN APPLICATION

As a true understanding of this procedure is more complicated, beginning Khoros users are advised to save the detailed explanation in the latter part of this section until later; one font is generally enough for an application. In addition, the .Xdefaults file provided with Khoros already provides different border colors for different types of buttons for the graphical user interface of Khoros applications. If these colors do not suit you, you may easily change them with a global change editor command, without knowing the details that make them work. You might, for example, edit the .Xdefaults file and change all occurances of "sea green" to "light green". Since this .Xdefaults file *only* uses the color "sea green" to specify the border color of a Help button, doing a global change from "sea green" to "light green" will result in all Help buttons being displayed with borders of light green. If you use the *vi* visual editor, the global change is made with:

```
:1,$ s/sea green/light green/g
```

If you use the *emacs* visual editor, which is more interactive, you can do a global change by:

```
<esc> @
```

Note that the colors supported by your display should be listed in `/usr/lib/X11/rgb.txt`. Some systems may have the **xcolors** program, which displays the available colors with labels, and is very helpful in deciding on a good combination of colors. Remember that changes made to your .Xdefaults file will not take effect until you execute:

```
% xrdb .Xdefaults
```

Some more creative users may want to specify a different font for different parts of the graphical user interface, or a different color for different parts of the graphical user interface that are not addressed in the .Xdefaults file distributed with Khoros. Specifying fonts for different items in the graphical user interface follows the same procedure as for specifying different colors. This process is tedious, but not very difficult. The tricky thing is that you must know the entire, complete name of each widget in the graphical user interface that You wish to change the font of. This information can be deduced from the UIS file for the Khoros application that you are interested in.

Let's learn with an example of changing fonts within an application. Suppose you want to run **preview**, and have it come up with the help button in kana14, the quit button in lucidasans-10, the "Revisit" and "Edit" action buttons in 6x13bold, and the text of the input file parameter in lucidasans-14. Copy the .Xdefaults file found in `KHOROS_HOME/dotfiles`. First, you must look at the *.form file for **preview**, because you will need the names that are specified there. Do:

```
% more KHOROS_HOME/repos/preview/preview.form
```

Here, look for the "Form_Name" specified at the end of the [-F] line, which is "UI_view". The "Subform_Name" specified at the end of the [-M] line is "show". You must recognise that **preview**'s user interface has only a single subform. There is no master form, as in **editimage**, or **xprism2**. So, in the .Xdefaults file, you will refer to the **preview** subform by its full name, which will be X_form_Y_subform, where X = Form_Name and Y = Subform_Name. So you know that you must begin lines in the .Xdefaults file as follows:

```
UI_view_form_show_subform*
```

Now, you want to specify fonts for the help and quit buttons. Help and quit buttons are always named %d_help_button and %d_quit_button respectively, where %d is the count of the number of help and quit buttons on that pane, starting at zero. There is only one help button and one quit button on the preview subform, so that means you put in the .Xdefaults file:

```
UI_view_form_show_subform*0_help_button:  kana14
UI_view_form_show_subform*0_quit_button:  lucidasans-10
```

Now, let's see what else we have to work with. In the preview.form file, there are two pane action buttons, specified by [-a] lines. Their variable names are "revisit" and "edit". Pane action buttons will be named x_pane_action_button, where x = variable name. Therefore, add to the .Xdefaults file:

```
UI_view_form_show_subform*revisit_pane_action_button*font:
6x13bold
```

```
UI_view_form_show_subform*edit_pane_action_button*font:  6x13bold
```

Finally, there is an input file selection. The backplane widget for InputFile selections is named "w_input_back" where w is the variable name at the end of the [-I] line - in this case "i". So the last line must read,

```
UI_view_form_show_subform*i_input_back*font:  lucidasans-14
```

Now, you're all set. Write the file, and do:

```
% xrdb .Xdefaults
```

If your fonts do not change properly, look for a line that looks like, "`preview.font: fixed`. Remember that this will over-ride any other specifications that you might have; you must delete this line in order for the above procedure to work.

Colors are changed exactly the same as fonts are, except that instead of referring to "*font", you refer to "*borderColor". We have found that adding colors to the graphical user interface is most effective when it is the color of the border that is changed. However, you might also want to experiment with "*foreground" and "*background" to achieve interesting effects.

CONTENTS

Chapter 1 - GETTING STARTED