-- --

Volume II

Khoros Programmer's Manual

Chapter 9

CONFIGURATION MANAGEMENT

Configuration Management for the Khoros System

Primary Author(s):

Tom Sauer & Charlie Gage

Printed: March 16, 1992

CONFIGURATION MANAGEMENT

A. INTRODUCTION

This chapter of the manual describes the source configuration of the Khoros system and how it is managed in a development environment. An important goal in the design of Khoros has been to create an environment that can change on a daily basis but yet not fall apart. (Surprisingly, it works most of the time.) The manager of the Khoros system will find this chapter useful, as it provides information about the organization of the Khoros system and on the utilities for installing and compiling the system.

Khoros is designed to work well in either a stand-alone environment or in a distributed environment. Several tools are provided to help manage the Khoros system more efficiently. Information is provided in this chapter on how to use the tools for different site configurations.

This chapter is organized into three sections. Section A describes the organization of the Khoros directory structure, and gives a brief explanation of the contents of each of the main directories in Khoros. Section B provides an overview of the Imake system necessary for installing and compiling Khoros, including explanations on the use of the **imkmf**, **makemake**, and **make** programs. This is followed by section C, which provides information on building and maintaining the Khoros srcmach tree for the support of multiple machine architectures.

A.1 ORGANIZATION OF THE KHOROS DIRECTORIES

An environment variable is used to define the path to the top level of the Khoros directory. This environment variable is termed **KHOROS_HOME**.

The Khoros directory structure is relatively complex because it is designed to support both the development of new code (i.e. a working environment) and the distribution of the system. The addition of new routines to Khoros is done in separate directory structure called a *toolbox*. The directory structure of a toolbox is similar to the directory structure of KHOROS_HOME. The *toolbox* area is designed for the addition of new routines provided by various users of Khoros outside of the main development team. Therefore all installation of new routines for Khoros should take place in the *toolbox* source tree, and not in the KHOROS_HOME source tree. For more information on creating and installing new routines in your own *toolbox*, see Volume II, Chapter 1, specifically you should read about *kraftsman, composer and kinstall..*

Please note that ANY ADDITIONS OR MODIFICATIONS TO THE KHOROS_HOME SOURCE DIRECTORY WILL NOT ALLOW REVISIONS OR PATCHES TO BE MADE TO THE KHOROS SYSTEM. THE KHOROS_HOME SOURCE TREE SHOULD NOT BE CHANGED!

While not imperative for those who simply intend to utilize the Khoros programs, it is most helpful for anyone who intends to add new programs to Khoros to acquire an understanding of the Khoros system directory structure.

The following table provides a brief description of the Khoros system directory structure.

KHOROS SYSTEM DIRECTORY STRUCTURE	
bin *	Location of Khoros binaries. It may be set up as a symbolic link to binmach.
binmach	Location of Khoros binaries if multiple machine architectures are supported. For example, in binmach/ there may be directories of bindecmips, binsun4, etc.
data	This directory contains sample data, plots and images.
doc *	This directory contains "pop-up" help for xvroutines. The files in doc are also used for creating the manual sections located in manual/.
dotfiles	The dotfiles directory provides the configuration files to run Khoros applications in a proper environment.
include *	System include files are located in this directory. See the descriptions in README.h for a brief description of each file.
lib *	Location of compiled libraries (.a's). It may be set up as a symbolic link to libmach.
libmach	Location of compiled libraries (.a's) if multiple machine architectures are supported. For example, in libmach/ there may be directories of libdecmips, libsun4, etc.
man *	Contains the man1 and man3 directories for "on-line" documentation on how to execute a program. (Accessed using **vman**.)
manual	This directory contains the documentation that is intended to be printed.
repos *	A repository of all Khoros default files, UIS files, configuration mach files, and all template files for the Imake system.
src *	The Khoros source tree is located here.
srcmach	This directory is used for symbolically linked source trees if multiple machine architectures are supported. For example, in srcmach/ there may be directories of srcdecmips, srcsun4, etc.

* These directories also present under {toolbox name}.

To run Khoros, a *minimal* system requires the contents of bin, repos, man, and doc. It is likely that you will also want a data directory for examples. If you plan on doing development within Khoros, you will also need the contents of lib, include, and src. More detailed information about the contents and organization of the data, repos, and src directories are provided in the sections that follow.

A.1.1 The Data Directory

The *data* directory contains images, kernels, look up tables, masks, and overlays that may be used as sample input data.

THE DATA DIRECTORY	
annotate	Annotation files for **editimage**, **xprism2** and **xprism3**.
bitmaps	A couple of X bitmaps created with **viff2xbm**, suitable for icons.
images	An assortment of VIFF (visualization/image file format) files.
kernels	VIFF images appropriate for convolution.
lut	An assortment of VIFF (visualization/image file format) files that have interesting color maps.
masks	Masks that can be used for the shape, clip and overlay functions in the interactive display programs.
morphokernel	VIFF images appropriate for morphology routines.
plot_functions	Equations and data that can be used in **xprism2** and **xprism3**.
sequences	Image time sequences in VIFF format.
signals	One dimensional real and complex VIFF format signals.
warp_coeffs	Example VIFF images that contain warping coefficients for **vwarp**.

A.1.2 The Repos Directory

The repos directory is a repository of required files. Primarily it contains the User Interface Specification (UIS) files for xvroutines.

THE REPOSITORY DIRECTORY	
Conventions	This directory contains template pane files for **cantata** and guidelines for creating a .pane file.
animate	The UIS file for **animate**.
annotation	The UIS file for annotation subforms that are used by **editimage**, **xprism2** and **xprism3**.
cantata	The UIS file for **cantata** is broken into subforms. Each subform file (.sub) refers to pane files (.pane). The cantata/subform subdirectory structure mimics that of the source directory structure.
composer	The UIS file for **composer**, and an example .prog file.
config	This directory contains the machine configuration and template files for the Imake system, OS and compiler specific installation scripts, and site specific mach files.

THE REPOSITORY DIRECTORY	
editimage	The UIS file for **editimage**.
fonts	Includes the various vector fonts used by **xprism2** and **xprism3**. These are required because the X fonts can not be resized or rotated.
preview	The UIS file for **preview**.
utils	This contains copyright templates and module header templates. **Ghostwriter** uses several of the files.
viewimage	The UIS file for **viewimage**.
warpimage	The UIS file for **warpimage**.
workspace	Several demo and test workspaces illustrating some of **cantata's** visual programming capabilities.
xprism	The UIS files for **xprism2** and **xprism3**.
journal	Journal playback files for demos of xvroutines.

A.1.3 The Source Directory

Within the *src* directory there are many levels of subdirectories. These subdirectories were chosen to categorize functions with similar functionality. For instance, you will find a *Lib* directory at the top of source and also in dspl, file_formats, num_analysis, remote_gis, vipl and xvroutines. The top level *Lib* contains libraries that are needed across the system. The lower level *Lib* directories contain source for libraries local to that functional category.

THE SOURCE DIRECTORY	
Lib	This directory contains several libraries that are required by the entire system.
dspl	Similar to vipl, except it contains one dimensional signal processing algorithms.
file_formats	This directory contains a collection of programs for converting to and from VIFF format.
num_analysis	This directory contains programs for matrix manipulations.
public_domain	This directory contains subdirectories which contain the public domain software which are used by Khoros.
remote_gis	This directory contains routines specific to remote sensing and GIS applications.
scripts	This directory contains subdirectories of csh script files used by Khoros.

THE SOURCE DIRECTORY	
tools	This directory contains subdirectories of programs used to manage the Khoros source tree.
vipl	This directory contains all the image processing driver programs. Within this directory, each subdirectory is dedicated to a a specific area of image processing. Below each of these is a subdirectory named for a specific program.
xvroutines	This directory contains all source for xvroutines. Below this is a Lib directory and a subdirectory for each program.
bootstrap	Programs necessary to boot the Khoros system. The programs in this directory must be compiled before anything else can be compiled.

To get a better understanding of how the *src* directory is arranged, the following diagram is provided to graphically illustrate the src directories and some of their subdirectories. Under *src* we have several subdirectories containing 1D and 2D data processing algorithms, file format conversion routines, and remote sensing/GIS routines. The categories were chosen to accommodate the broad spectrum of algorithms that exist in Khoros.

```
                     |-ghost----------
                     |-verror---------
        |-Lib--------|-vexpr----------
        |            |-vgparm---------
        |            |-*
        |
        |            |-Lib------------
        |            |-filter_1Dfreq--
        |-dspl-------|-filter_1Dtime--
        |            |-info-----------
        |            |-*
        |
        |            |-Lib------------
        |-file_formats--|-no_format------
        |            |-remote_gis-----
        |            |-standard-------
        |
        |-num_analysis--|-Lib------------
        |            |-matrix_algebra-
        |
        |            |-eispack--------
        |            |-imake----------
        |-public_domain-|-linpack- -------
        |            |-makedepend-----
        |            |-tiff-----------
|-src-|
        |            |-Lib------------
        |-remote_gis----|-surface--------
        |            |-vector---------
        |            |-warp-----------
        |
        |-scripts-------|-cantata--------
        |            |-tools----------
        |
        |            |-formatdoc------
        |            |-ghostcheck-----
        |-tools---------|-helppage-------
        |            |-imkmf----------
        |            |-*
        |
        |            |-Lib------------
        |            |-arith_binary---
        |-vipl----------|-arith_logical--
        |            |-arith_unary----
        |            |-*
        |
        |            |-Lib------------
        |            |-animate--------
        |-xvroutines----|-cantata--------
        |            |-composer-------
        |            |-*
                                  * - more directories follow
```

A.1.3.1 The *src/dspl* Directory

The table below describes the taxonomy for the 1D data processing algorithms.

1D DATA PROCESSING ORGANIZATION	
filter_1Dfreq	These routines operate on signals in the frequency domain.
filter_1Dtime	These routines operate on signals in the time domain.
info	These routines are designed to provide the user with information about a signal.
input_create1D	These routines generate a signal based on functions or user specifications.
linearop1D	A collection of routines that provide "classical" linear theory operations. They include convolution, correlation, etc.
modify_seq1D	A collection of routines designed to manipulate signals or portions of signals.
spectest1D	These routines provide several methods for estimating the spectral content of a signal.
transform1D	These routines are used to go to and from the Fourier Domain and perform operations in the Fourier Domain.

A.1.3.2 The *src/vipl* Directory

Because of historical reasons, *vipl* forms the base set of the data processing programs in Khoros. The categories of num_analysis, remote_gis, dspl, and file_formats have some dependencies on *vipl*.

The table below describes the taxonomy for the 2D data processing algorithms.

2D DATA PROCESSING ALGORITHM ORGANIZATION	
arith_binary	These routines use two images to produce a single output image. They also may use a masking image to gate what is used from one of the input images. The operations are arithmetic operations performed on a pixel by pixel basis from the two images or signals.
arith_logical	These routines perform logic operations on a pixel by pixel basis.
arith_unary	These routines use one image to produce a single output image. The operations performed are arithmetic point operations done on a pixel by pixel basis. Several of the operations may be used with an optional masking image or signal.
classify	These routines are used to perform supervised or unsupervised classification.

2D DATA PROCESSING ORGANIZATION	
convert_color	These routines use images with multiple bands representing a specific color to create images which are mapped into other color spaces.
convert_compr	Currently includes only a driver for existing compression algorithms. The purpose is to reduce the amount of data needed to store an image.
convert_data	These routines change the data storage size/type of VIFF image.
feature	These routines calculate various feature measurements of images.
filter_2Dfreq	These routines operate on images in the frequency domain.
filter_2Dspatl	These routines operate on images in the time domain.
filter_morph	These routines operate in the spatial domain using morphology filters.
geomanip	These routines alter the location of pixels in an image or signal by expanding, shrinking, rotating, etc.
histogram	These routines modify the histogram of an image or display the current histogram of an image.
info	A collection of routines designed to provide the user with information about an image.
input_create2D	These routines generate an image based on functions or user specifications.
map	These routines manipulate the color maps of an image.
output_dither	These routines convert images to a black pixel/white pixel image for use on monochrome screens or printers without pre-dithering algorithms.
output_print	These routines are used to format and send an image to a printer.
segment	These routines are intended to define regions of an image as a function of grey level pixel values.
subregion	These routines are designed to extract and manipulate subimages.
transform2D	These routines are used to go to and from the Fourier Domain and perform operations in the Fourier Domain.

A.1.3.3 The *src/file_formats* Directory

The following table describes the taxonomy for the file format conversion algorithms.

FILE FORMAT ORGANIZATION	
no_format	These routines are for converting ascii and raw images to VIFF format, and from raw to VIFF format.
remote_gis	These routines are for converting to and from the VIFF format to remote sensing and GIS file formats.
standard	These routines are for converting to and from the VIFF format to several other popular file formats.

A.1.3.4 The *src/remote_gis* Directory

The table below describes the taxonomy for the remote sensing and GIS algorithms.

REMOTE SENSING AND GIS ALGORITHM ORGANIZATION	
surface	A collection of routines designed to compute surface normals and correct the illumination gradient in an image.
vector	These routines manipulate the color maps and bands of dlg images.
warp	These routines are for performing image registration of images.

B. THE KHOROS IMAKE SYSTEM

The Khoros system is designed to be run on a variety of platforms and as such needs a common method which allows Khoros to be compiled and installed on different architectures. The method that was chosen uses *Imakefiles* and the utility **imake**, the same utility used by MIT with the X11 source. **Imake** allows architecture dependent *Makefiles* to be automatically generated from an *Imakefile* specification.

Khoros uses a series of *Imakefiles* which are created from a set of template files. Each *Imakefile* in turn includes a ''specification file'', that defines which **imake** rules are applied for making the resulting *Makefile*. The resulting architecture dependent *Makefile* is then used to compile and install programs, libraries, and shell scripts.

B.1 BACKGROUND

The **imake** and *Imakefile* environment created for MIT X11 was not intended for software development. It was primarily set up to help compile and install a 'final product', on many different hardware architectures. Khoros, on the other hand, uses **imake** and *Imakefiles* while in the development of the 'final product' allowing users and developers to extend Khoros. As a result, many modifications were necessary to allow Khoros to be able to use *Imakefiles*. These modifications necessitated the creation of a set of files similar in nature to the 'normal' **imake** definition and configuration files, but allow Khoros to use **imake** other ways To help manage the growth of Khoros during its development, **imkmf** and **makemake** were built to aid in the creation of *Imakefiles* and *Makefiles* in a consistent and easy manner. These tools allow a software developer to easily add to a 'final product' without having to spend an inordinate amount of time learning about **imake** and *Imakefiles*.

This manual section covers various aspects of how Khoros uses *Imakefiles* using the utility **imkmf**, how to install Khoros using the existing *Imakefiles*, and how to integrate new sources into Khoros or to move/rename existing sources.

This document does not go into any details on the specifics of how **imake** works, but enough information is given to compile and manage the Khoros **imake** system. For more details, it is suggested that the reader consult the following references.

"Configuration Management in the X Window System," Fulton, Jim (found in MIT X.V11R4 source in mit/doc/config/usenixws/paper.ms)

"An *Imake* Tutorial," Moraes, Mark, November 24, 1989 (found in MIT X.V11R4 source in contrib/doc/imake/imake.tex)

B.2 KHOROS IMAKE TOOLS: imake, makemake, imkmf, makedepend

Four main utility programs are used to manage the Khoros **imake** system. These programs include **imake**, **makemake**, **imkmf**, and **makedepend**. **Imake** and **makedepend** are low-level tools called by **makemake** and/or **imkmf**. **Makemake** and **imkmf** are used frequently to update and build both *Makefiles* and *Imakefiles*. For specific information on how to use these tools, consult the Khoros manpages makemake(1), imkmf(1), imake(1) and makedepend(1). The following paragraphs provide a brief introduction to each of these utility programs.

The utility **imake** is a low level configuration tool. **Imake** takes an Imakefile with a set of specifications defined in *KHOROS_HOME/repos/config/imake_conf* and generates a architecture specific *Makefile*. **Imake** uses the C preprocessor (cpp), taking advantage of its include-file and macro-processing facilities for the purpose of generating *Makefiles* suitable for **make**.

The Khoros utility **makemake** is a shell script that calls **imake** with all of the necessary arguments to successfully **make** or update a *Makefile* from an *Imakefile*. Currently, the **makemake** script contains the following command:

```
imake \
   -DKHOROS_TEMPLATE=\"Khoros.tmpl\"  \
   -DKHOROS_HOME=$KHOROS_HOME  \
   -D{toolbox_defines}  \
   -T$KHOROS_HOME/repos/config/imake_conf/Khoros.tmpl  \
   -I$KHOROS_HOME/repos/config/imake_conf  \
   -I{toolbox_includes}  \
   -I$KHOROS_HOME/repos/config/imake_tmpls
```

These arguments to **imake** specify *$KHOROS_HOME/repos/config/imake_conf/Khoros.tmpl* as the basic startup file which in turn includes all of the other necessary files, including the current directory's *Imakefile* (*./Imakefile*). **Imake** will first look in the directory *$KHOROS_HOME/repos/config/imake_conf* for the **imake** specification files. It will then search all the toolbox *repos/config/imake_conf* directories for the toolbox definition file. Figure 1 Section B.3.1 shows the files that **imake** includes in the process of making a *Makefile*.

The **imkmf** utility automatically generates an *Imakefile*, assuming that one does not already exist, or updates an already existing one to contain information sufficient to compare the files contained in the directory. If an *Imakefile* does not exist, **imkmf** copies a template, specified on the command line with '-type', into the current directory and then proceeds to update it. Revision is accomplished by reviewing of all the files in the current directory and updating any existing "definition" that **imkmf** uses. **imkmf** uses the template *Imakefiles* that can be found in $KHOROS_HOME/repos/config/imake_tmpls/. These template files include different specifications to compile a program (independent or dependent of X11, independent or dependent on Fortran), to create a library, to continue making in sub-directories, and to install shell scripts. All of these templates contain empty **make** definitions which are filled in by **imkmf**. The templates available are:

Imakefile.p — for programs which are independent of X11
Imakefile.pf — for programs which are dependent on Fortran and independent of X11
Imakefile.x — for programs which are dependent on X11
Imakefile.xf — for programs which are dependent on Fortran and X11
Imakefile.l — for libraries
Imakefile.d — for directories
Imakefile.s — for scripts

Finally, the **makedepend** utility is used to generate file dependencies for C source files in each directory after the *Makefile* has been built. Dependencies of C files upon objects can be statically listed in the *Imakefile*, but dependencies for header files (.h files) can not. Since different architectures organize dependencies for header files differently, these dependencies must be generated dynamically for each architecture. **Makedepend** is called automatically when a *Makefile* is generated or updated using **makemake**.

B.3 CONFIGURING THE KHOROS IMAKE SYSTEM

Before Khoros can be compiled on a platform, the *imake* system must be configured. The configuration consists of editing some of the *imake* include-macro files. Before discussing the specifics required to compile Khoros, an overview of the *imake* symbol and macro construct and the include-macro files is given. This will, hopefully, give one a clearer idea of how *imake* gathers the information necessary to build a *Makefile*. Finally, information specific to configuring the imake system is discussed.

B.3.1 Imake Symbol and Macro Construct

The **imake** system uses symbol and macro constructs to gather information necessary to generate a *Makefile*. The symbol and macro constructs are defined in the Khoros **imake** configuration files. The configuration files are discussed in the next subsection.

Imake passes its input through the C preprocessor (cpp) so that the definitions in the configuration files can take advantage of several features not present in **make**. The features include parameterized macros (using #define), file inclusion (using #include) and conditional testing (using #if, #ifdef and #ifndef).

The flexibility of the Khoros configuration files is achieved through the following construct, which defines a *symbol* only if it has not previously been defined:

```
#ifndef symbol
#define symbol
#endif
```

This construct allows any system, library, or build symbol to be given a default definition if none has previously supplied. This definition construct along with support for inclusion of platform, site and library specific files prior to the section in which the default symbols are defined, provides the ability to build a hierarchy for defining symbols. For example, the default C compiler symbol, *CcCmd*, occurs in the build section of the *Khoros.tmpl* file and is defined as:

```
#ifndef CcCmd
#define CcCmd cc
#endif
```

However, suppose you want to use *gcc* as the C compiler rather than *cc*. The *CcCmd* can be overridden by putting the following symbol definition in either the *Site.def* or *platform.cf* file.

```
#ifndef CcCmd
     #define CcCmd gcc
     #endif
```

The pattern followed in the the system/build section of the *Khoros.tmpl* file is that definitions for *cpp* symbols are listed first, followed by definitions for *make* variables. The strategy adopted is to associate a *cpp* symbol with a given *make* variable by equating the *make* variable to a given *cpp* symbol. The example below illustrates this concept:

```
#ifndef CcCmd
     #define CcCmd gcc
     #endif
```

```
        .
        .
        .
       CC=CcCmd
```

For a listing of all *cpp* symbols and **imake** rules, consult Vol 2, Appendix C, Imake Symbols and Variables. This appendix is helpful when either determining which *cpp* symbol to change/set for a correct compile and when porting to a new architecture.

B.3.2 *Imake* Configuration Files

To produce a *Makefile* from an *Imakefile*, the **imake** utility uses configuration files located in *$KHOROS_HOME/repos/config/imake_conf*. The following list briefly describes each configuration file:

Khoros.tmpl —	Master template file used to generate the Makefile
platform.cf —	Platform-specific definitions (the filename varies)
Site.def —	Site-specific definitions
Library.def —	Library specific definitions
Khoros.rules —	*cpp* macros to generate *make* rules
toolbox.def —	Toolbox-specific definitions (the filename varies)

Figure 1 below shows how the configuration files are included and processed by **imake** to produce a Makefile. The *Khoros.tmpl* file contains a #include line for each of the other four configuration files and for the *Imakefile*, and the *Imakefile* contains a #include line for the *toolbox*.def file. The *Khoros.tmpl* file also contains in-line sections for a header block section, a description of system characteristics, build definitions and *extra make* rules.

The *Khoros.tmpl* file contains all the default symbol definitions for the Khoros **imake** system. Symbols defined in the *platform.cf* file will override the same symbols in the *Site.def*, *Library.def* and *Khoros.tmpl* files. Subsequently, symbols defined in the *Site.def* file, which have not previously been defined in the *platform.cf* file will override the same symbols in the *Library.def* and *Khoros.tmpl* files. Any symbols defined in the *toolbox*.def file will override the same symbols in all the previously included imake configuration files. Note: the *Imakefile* should only contain **make** variables; it should not contain *cpp* symbols. A **make** variable defined in the *Imakefile* will override all previous declarations of that variable since the variable is made by assignment rather than defined.

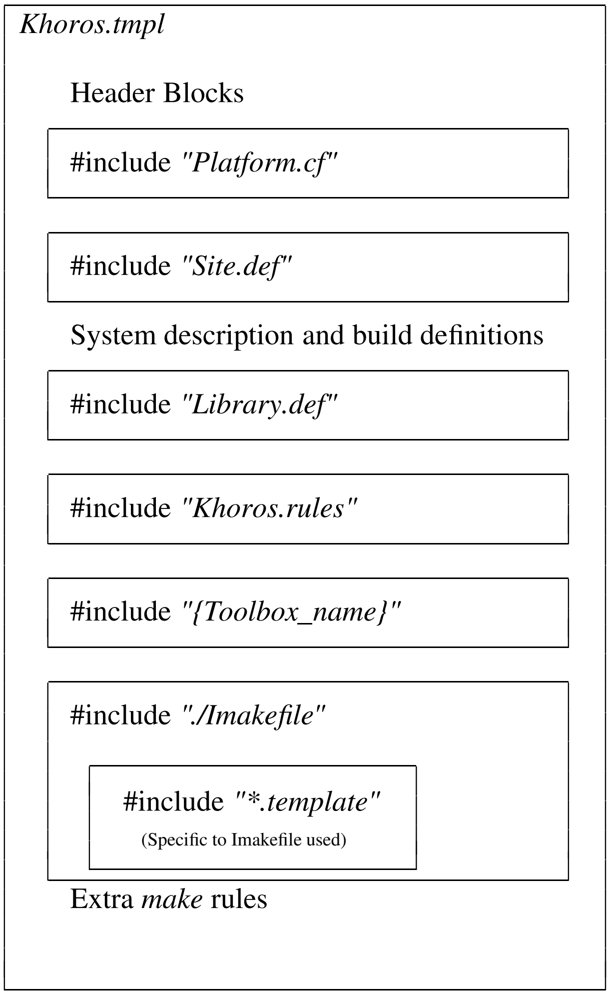

Figure 1. Khoros **imake** template structure

B.3.2.1 Header Block

The header block section of *Khoros.tmpl* determines the type of machine on which **imake** is being run. This will allow **imake** to include the proper *platform.cf* file when creating *Makefiles*. Most C preprocessors have unique symbols that indicate a given platform. For example, a SUN4's preprocessor defines the symbol "sun", where as the Apollo defines the symbol "apollo". If the C preprocessor does not define any such symbol, one can use the BOOTSTRAPFLAGS option with **imake** to define a unique C preprocessor symbol. **imake** is designed to pass the BOOTSTRAPFLAGS symbol to the C preprocessor for use. This is important for those trying to port Khoros to a new platform.

B.3.2.2 Platform.cf File

After the header block section has determined which platform-specific file to use, That file is then included by *Khoros.tmpl*. The platform-specific file includes symbol definitions necessary to build Khoros on a particular system. Overall these files are usually a conglomeration of symbol definitions, make variables and imake rules. This file must be edited before attempting to compile Khoros. Appendix C, Vol 2, "Imake Symbols and Variables" will aid in determining which symbol definitions need to be changed or added.

B.3.2.3 Site.def File

This file contains site-specific definitions. This file should reflect site-wide conventions. The *Site.def* file is a good place to override symbol defaults defined in the *Khoros.tmpl* file, at least in theory. This file works fine if Khoros is going to be installed and maintained on only one platform. If Khoros is going to be installed and/or maintained on multiple platforms, it is best make changes in the *platform.cf* file rather than the *Site.def* file. This *Site.def* is, in a sense, global; symbol defines for one platform might not be correct for a different platform.

B.3.2.4 System Description and Build Definitions

The inclusion of platform-specific and site-specific files in the *Khoros.tmpl* are followed by a section containing some default symbol definitions. This section will make sure that symbols, such as CcCmd, get defined that have not previously been defined. This section is also where **make** variables get equated to *cpp* symbols. This section of the *Khoros.tmpl* should not be modified; definitions should be overridden in the platform-specific and site-specific files.

B.3.2.5 Library.def File

The *Library.def* file contains symbol definitions and **make** variable equates for the Khoros libraries and programs. This file defines the system libraries, X11R4 libraries, Fortran libraries and Khoros libraries necessary to build Khoros. Most of the symbol definitions and **make** variables in this file do not have defaults in the *Khoros.tmpl* file. The *Library.def* file is included after the *Site.def* and *platform.cf* files.

B.3.2.6 Khoros.rules File

This file contains the *cpp* macros used to generate **make** rules for the *Makefiles*. This file should not be modified. If rule changes need to be made, the *platform.cf* file is the place to make changes. Consult the *sgi.cf* file for an example.

B.3.2.7 toolbox.def File

This file contains the symbol definitions for a particular toolbox. All the symbol definitions specified in this file will override the same symbol definition that was previously defined. This file exists in each toolbox in the repos/config/imake_conf directory. The **imkmf** program will automatically include this file. If the "-toolbox" **imkmf** option is not used on the command line, the default toolbox definition file for KHOROS_HOME will be used. It is located in
$KHOROS_HOME/repos/config/imake_conf/Toolbox.def. If the "-toolbox {toolbox_name}" option is specified, the
$TOOLBOX_NAME/repos/config/imake_conf/{toolbox_name}.def file will be used. For more

information on setting up and managing toolboxes, consult Vol 2, Chapter 1, specifically read about **kraftsman, composer and kinstall**.

B.3.2.8 ./Imakefile File

The last file included in the *Khoros.tmpl* file is the *Imakefile* from the current directory. The Khoros *Imakefiles* define the sources, objects, header files and necessary libraries needed to compile. *Imakefiles* should always be generated using the progarm **imkmf**.

B.3.2.9 Extra Make Rules

The last section of the *Khoros.tmpl* file defines some of the common **make** targets such as "clean", "tags", etc. These definitions should not be modified.

B.3.3 Khoros *Imakefile* Templates

The Khoros **imake** system uses template *Imakefiles* when generating *Imakefiles*. The utility **imkmf** uses the template files in *$KHOROS_HOME/repos/config/imake_tmpls* to make an *Imakefile*.

All of the *Imakefile* templates, besides containing the **make** definitions to be filled in by **imkmf**, contain ''include'' references to other files that contain specifications which are common to all of the *Imakefile* templates. Templates *Imakefile.p* and *Imakefile.x* include the file *Prog.template*; *Imakefile.pf* and *Imakefile.xf* include the file *FProg.template*; *Imakefile.l* includes the file *Lib.template*; *Imakefile.d* includes the file *Dir.template* and *Imakefile.s* includes the file *Scrpt.template*. Note, these files should never have to be touched, changes to the imake system should be made through the *platform.cf* files, the *Site.def* file and the *Library.def* file located in *$KHOROS_HOME/repos/config/imake_conf.*

The **.template* files contain the **imake** specifications instead of the *Imakefile* templates themselves because if a ''rule'' needs to be added or modified which changes the way *Imakefiles* work, for example compiling a library, then only a single file, in this case *Lib.template*, needs to be updated instead of all library *Imakefiles*.

B.4 KHOROS INSTALLATION CONFIGURATION

Before the installation can begin, there are three files that need to be checked for correctness to insure proper installation of Khoros. All three files are located in *$KHOROS_HOME/repos/config/imake_conf* and contain symbol definitions to override default specifications.

The first file, *platform.cf*, where *platform* is the architecture of your machine. For example, use the file "sun.cf" if you are on a Sun, use the file "ultrix.cf" if you are on a Decmips, use the file "sgi.cf" if you are on a SGI machine. The *platform.cf* file contains machine specific symbol defines. If is important to set the symbols to the correct value. For example, if your system does not have a Fortran compiler, then set the symbol "HasFortran" to NO, if your system does not support Saber-C, set the symbol HasSaberC to NO.

The next file that needs to be modified is the *Site.def* file. It is suggested that the file not be changed. It is better to copy the symbols that need changing into the *platform.cf* file, and make the changes in the *platform.cf* file. The reason for this is because if you are going to support Khoros on more than one architecture the *platform.cf* will override the *Site.def* file. The *Site.def* file is a "global" file where as the *platform.cf* is local to a particular platform.

The symbols in the *Site.def* file define where the Khoros binaries, libraries, man pages, and source directories exist. Also, this file contains some of the common symbols that need to be overridden from the defaults. Note, many of the symbols in the *Site.def* file will appear in the *platform.cf* file.

The last file that needs to be modified is the *Library.def* file. This file contains the symbol defines for which libraries are necessary to link against. The only symbols that should be changed in this file are *LibDir, SysLibs, XLibs, ForSysLibs, and ExtraLibs*.

B.4.1 Multiple Architectures

If you plan to support Khoros on multiple architectures, then all changes to the **imake** system should be made in the *platform.cf* file. The *Site.def* and *Library.def* file act as "global" files for symbol defines, whereas the *platform.cf* file is local to a particular architecture. Again, any symbol that is defined in the *platform.cf* file will override the same symbol in the *Site.def* and *Library.def* file.

B.4.2 Commonly Asked Questions

Below are a few commonly asked questions about the **imake** configuration files. We suggest that the symbol defines discussed below are put in the *platform.cf* file.

Q: X11 directories are not in the "standard" place on my machine, how do I specify where the X11 directories exist?

A: define the symbol "XLibDir" to the directory where the X11 libraries exist

```
#define XLibDir /usr/local/lib/X11
```

define the symbol "XIncludes" to the directory where the X11 include directory exists.

```
#define XIncludes -I/usr/local/include
```

Q: Fortran libraries are in /usr/lang rather than in /usr/lib, how do I change where to look for Fortran libraries?

A: define the symbol "FortranLibDir" to the directory where the Fortran libraries exist

```
#define FortranLibDir /usr/lang
```

Q: I need to add some other libraries to compile against, how do I add these extra libraries?

A: define the symbol "ExtraLibs" to the directory where the libraries exist

```
#define ExtraLibs /myhome/foobar
```

Q: How do I make the libraries load with a different option than the C programs load with?

A: define the symbol "LibraryLdCmd" to the command that you want the libraries to load with and the "LdCmd" to the command that you want the C programs to load with.

```
#define LibraryLdCmd ld
#define LdCmd cc
```

Q: How do I specify different compile options for libraries and C programs.

A: define the symbol "LibraryCCOptions" to the options that you want the libraries to compile with and the "DefaultCCOptions" to the options that you want the C programs to compile with.

```
#define LibraryCCOptions -q -P
#define DefaultCCOptions -float
```

Q: How do I specify different compile options for Fortan libraries and Fortran object files.

A: define the symbol "LibraryFCOptions" to the options that you want the libraries or other Fortran object files to compile with.

```
#define LibraryFCOptions -O
```

Although many more questions about configuring the **imake** system have been asked, this set of questions and answers should give one the feel of how to make changes to the **imake** configuration files. It is strongly suggested that the document Vol 2, Appendix C, "Imake Symbols and Variables" be consulted when making changes to the **imake** symbols.

B.5 ADDING FILES TO KHOROS

At this point it is assumed that Khoros has already been successfully installed, and now other data processing programs or applications are to be added to enhance it. These enhancements may range from simply adding a new algorithm to an existing library, to something a little more difficult, such as adding a new library, program or shell script. NOTE, KHOROS SOURCE SHOULD NOT BE MODIFIED. MODIFICATION OF KHOROS SOURCE WILL PREVENT THE ABILITY TO UPDATE THE SOURCE WITH LATER PATCHES. New routines, libraries and scripts should be installed in a Khoros toolbox, or kept in the users local area.

In all cases the *Imakefiles* are managed by the **imkmf** utility which either creates *Imakefiles* if none exist, or updates them to contain current information concerning the files in existence in the current directory.

It is strongly suggested that the Khoros utilities for installing and managing the Khoros source tree and the Khoros toolboxes be used rather than performing the tasks manually, as outlined below. Please consult the "Writing Programs" documentation Vol 2, Chapter 1, and the Configuration Management documentation Vol 2, Chapter 9, Section A before installing a new routine. The utilities that may be helpful for managing Khoros and Khoros toolbox source include: **kraftsman, kmakeall, kinstall, kdinstall, krmprog, ksrcconf, and klnfile**. Use **vman** to obtain more information on these tools.

B.6 UPDATING *Imakefiles*

If all the *Imakefiles* in a subdirectory need to be updated or changed to reflect changes made to the *Imakefile* templates, probably as a result of adding a new library (see EXAMPLE 2), then all that needs to be done is to cd into the directory and run **make AllImakefiles**. The *Imakefiles* have built into them the necessary information so that **imkmf** gets called with the correct arguments.

If the *Imakefiles* need to be completely regenerated, the command **make GenAllImakefiles**, will remove the existing *Imakefiles* and recreate them. The difference between **make AllImakefiles** and **make GenAllImakefiles** is **make AllImakefiles** just updates existing *Imakefiles* while **make GenAllImakefiles** recreates them. Normally **make GenAllImakefiles** is not needed.

B.7 WORKING WITH *Makefiles*

A user should never have to edit a *Makefile* to change a **make** definition (for example to change the '-O' option to '-g'). Changes can be made by either modifying the *Imakefile* and then remaking the *Makefile*, by running **make Makefile**, or by using **make CDEBUGFLAGS=-g**. The user has access to several **make** definitions to help accomplish this task. Some of the more common definitions are listed in Table 1.

make definitions	Default	comments
CDEBUGFLAGS	-O	Passed to all compiles. If a 'debug' compile is needed, just change this to be '-g'. This definition is used in defining CFLAGS, which is used when compiling C files, and LDOPTIONS, which is used when loading (linking) a program, *a.out* for example.
STD_INCLUDES	-I.	*$KHOROS_HOME/include* is already included elsewhere. This field should only be added to in the rare case where an include file is needed and is not in either of '.' or *$KHOROS_HOME/include*. This definition is used in defining CFLAGS which is used when compiling C files.
YFLAGS		Flags passed to **yacc**
LFLAGS		Flags passed to **lex**
EXTRA_LOAD_FLAGS		When compiling a program, this field can be used to pass additional flags to the loader during the program load phase, for example any additional libraries that need to be linked in. This definition is used when loading (linking) a program, *a.out* for example.

Table 1. Some common **make** definitions the user may change in the *Imakefile* or on the **make** command line.

B.8 EXAMPLES

In this section, several references are made to Khoros utilities which are used to help manage the system. For a complete description of these utilities and their exact use consult Section A of this chapter. **kmakeall** is basically equivalent to **make** except that **kmakeall rsh**es to other systems to **make** Khoros on different architectures. **ksrcconf** creates symbolically linked source trees allowing different architectures to be compiled using the same source without having to have multiple copies of the source. It is strongly suggested that the Khoros manual pages for these utilities be consulted to provide a detailed description of what these utilities do.

There are also references made to 'srcmach' trees which are what **ksrcconf** creates. For example, in $KHOROS_HOME/srcmach there might contain directories **srcdecmips** and **srcsun4** which are source trees, symbolic links to the real source, created to allow compiles on DEC Mips and Sun 4 based machines respectively.

The rest of this section contains examples of how to add new routines or programs into a Khoros *toolbox* (see Volume 2, Chapter 1 for instructions on using **kraftsman** to create and maintain a toolbox). The examples that follow will assume that a toolbox has been previously created and the name of the toolbox is CONTRIB_TOOLBOX.

B.8.1 Adding a New Routine to an Existing Library

The following steps show how to install a new routine, source file, into an already existing library. For example, assume that a routine called *vfoo* (file *vfoo.c*) is to be added to the library *libcontrib.a* which is located in the toolbox called CONTRIB_TOOLBOX. Do not forget to specify the -toolbox option when using **ksrcconf** and **kmakeall**.

1) Install the new file, *vfoo.c*, in the library directory, with either the **mv** or **cp** command.

2) Run **imkmf** in the library directory, to update the *Imakefile* so that it contains a reference to the newly added file *vfoo.c*.

3) Run **makemake** to update the *Makefile* to reflect the changes made to the *Imakefile* in step 2 above.

4) If there are no 'srcmach' trees, go to step 5b else continue with step 5a.

5a) Run **ksrcconf -toolbox contrib_toolbox** to create the symbolic links for the new file in the 'srcmach' trees.

6a) Run **kmakeall -toolbox contrib_toolbox Makefile** to update the *Makefiles* in the 'srcmach' trees.

7a) Run **kmakeall -toolbox contrib_toolbox depend** to add the file dependencies to the *Makefiles* in the 'srcmach' trees. Do not combine this step with step 6a above, i.e. **kmakeall -toolbox contrib_toolbox Makefile depend**, because the *Makefile* that is used for the 'depend' would be the old one and not the new one which contains a reference to the newly added file *vfoo.c*.

8a) Run **kmakeall -toolbox contrib_toolbox install** to compile the new file in the 'srcmach' trees and to install the library in the library directory. Do not combine this step with step 7a above, i.e. **kmakeall -toolbox contrib_toolbox depend install**, because the *Makefile* that would be used for the 'install' is the old one and not the one with the dependencies added.

Adding the routine *vfoo* to the library is finished.

5b) Run **make depend** to add the file dependencies to the *Makefile*. Do not combine this step with step 3 above, i.e. **make Makefile depend**, because the *Makefile* that is used for the 'depend' would be the old one and not the new one which contains a reference to the newly added file *vfoo.c*.

6b) Run **make install** to compile the new file into the library and to install the library in the library directory. Do not combine this step with step 5b above, i.e. **make depend install**, because the *Makefile* that is used for the 'install' would be the old one and not the one with the dependencies added.

Adding the routine *vfoo* to the library is finished.

B.8.2 Creating a New Library

The following steps show how to install a new library into the "CONTRIB_TOOLBOX" toolbox. For example, assume that the library *libfubar.a* is to be added to your toolbox and is to be placed in *$CONTRIB_TOOLBOX/src/Lib*. For clarity, the directory name should be the same as the library name. For this example since the library will be *libfubar.a*, the directory name should be *fubar*.

1) Make a directory *fubar* in *$CONTRIB_TOOLBOX/src/Lib*.

2) Install the files that will make up the new library into the *fubar* directory with either the **mv** or **cp** command.

3) cd into the new directory, *fubar*.

4) Run **imkmf -type lib -name fubar -toolbox contrib_toolbox** to create the *Imakefile*. The *libname* argument should be the name of the library, which in this example is *fubar*.

5) Run **makemake** to create the first *Makefile*.

6) cd .. (into *$CONTRIB_TOOLBOX/src/Lib*).

7) Run **imkmf** to update the *Imakefile* in *$CONTRIB_TOOLBOX/src/Lib* so that it contains a reference to the new directory *fubar*.

8) Run **make Makefile** to update the *Makefile* to reflect changes made to the *Imakefile* in step 7 above.

9) If there are no 'srcmach' trees, go to step 11b else continue with step 11a.

10a) Run **ksrcconf -toolbox contrib_toolbox** to create the symbolic links for the new directory in the 'srcmach' trees.

11a) Run **kmakeall -toolbox contrib_toolbox Makefile** to update the *Makefile* in the 'srcmach' trees.

12a) cd into the new directory, *fubar*.

13a) Run **kmakeall -toolbox contrib_toolbox install** to make, in any 'srcmach' tree, the new library and to install it.

14a) Go to step 15.

10b) cd into the new directory, *fubar*.

11b) Run **make install** to make the new library and to install it. Do not combine this step with step 11b above, i.e. **make depend install**, because the *Makefile* that is used for the 'install' is the old one and not the one with the dependencies added.

Continue with step 15.

15) Now that the library has been compiled and installed, the toolbox imake definition file needs to be updated so that programs will know about the new library.

cd into *$CONTRIB_TOOLBOX/repos/config/imake_conf*.

Edit the file *contrib_toolbox.def* adding `-lfoobar` to the `TOOLBOX_LIBRARIES` line, and `$(LIBDIR)/libfoobar.a` to the `TOOLBOX_DEP_LIBRARIES` line.
Note, if the *fubar* library is a fortran library then modify the `FOR_TOOLBOX_LIBRARIES` and `FOR_TOOLBOX_DEP_LIBRARIES` lines rather than the `TOOLBOX_LIBRARIES` and `TOOLBOX_DEP_LIBRARIES` lines. If the *fubar* library makes calls or references any of the X11 libraries then modify the `X_TOOLBOX_LIBRARIES` and `X_TOOLBOX_DEP_LIBRARIES` lines rather than the `TOOLBOX_LIBRARIES` and `TOOLBOX_DEP_LIBRARIES` lines.

17) At this point, the addition of library *fubar* to the toolbox is complete; allowing programs to know about it. Before compiling a program which needs to link against the new library, the *make Makefile* command must be run on the programs *Makefile*.

B.8.3 Creating a New Program (independent of X11)

The following steps show how to install a new program, independent of X11, into a Khoros toolbox. For example, assume that the program, *vbar*, is a routine and falls under the *arith* category. Also assume that this program needs to call the library routine *lvbar*. For clarity, the directory name should be the same as the program name. Note, Khoros provides a tool, **kinstall**, for installing new routines. For more information on **kinstall** consult kinstall(1) and Volume 2, Chapter 1 - Writing Programs. The method outlined below is for manual installation of a routine, we recommend that you use **kinstall**.

There are two parts to installing a program. The first is adding the routine *lvbar* to the library located in *$CONTRIB_TOOLBOX/src/Lib/bar* and then installing the modified library. The procedure for doing this is described above in the above example. The second part is installing the ''main'' part of the program. The procedure for doing this is described in the steps below.

It should be noted that it is very likely that the library module source will be located in the same directory with the ''main'' source of the program, requiring that steps 1, 2 and 3 below be completed before the library module can be added to the library, as described in the above example.

1) Create the directory *vbar* in *$CONTRIB_TOOLBOX/src/arith*. Note, if the *arith* directory does not exist, you must first create the *arith* directory then create the *vbar* directory. In the *arith* directory you must create an Imakefile using **imkmf -type dir -toolbox contrib_toolbox** and a Makefile using **makemake**. If you

created the *arith* directory, then `cd ..` and run Run **imkmf** to update the *Imakefile* in *$CONTRIB_TOOLBOX/src* and **make Makefile** to update the *Makefile*.

2) Install the files for the new program in the *vbar* directory with either the **mv** or **cp** command.

3) cd into the new directory, *vbar*.

   If there is a library module associated with the ''main'' source, then it needs to be installed in the required library as described in the above example before continueing. In our example, the source for *lvbar* needs to be installed in library *$CONTRIB_TOOLBOX/src/Lib/bar*.

4) Run **imkmf -type prog -name vbar -toolbox contrib_toolbox** to create the *Imakefile*. Note, if the program makes a call to Fortran or any Fortran function is used, use **imkmf -type fprog -name vbar -toolbox contrib_toolbox** to create a Fortran *Imakefile*. The *progname* argument should be the name of the program, which in this example is *vbar*.

5) Run **makemake** to create the *Makefile*.

6) cd .. (in *$CONTRIB_TOOLBOX/src/arith*).

7) Run **imkmf** to update the *Imakefile* in *$CONTRIB_TOOLBOX/src/arith* so that it contains a reference to the new directory *vbar*.

8) Run **make Makefile** to update the *Makefile* to reflect changes made to the *Imakefile*.

9) If there are no 'srcmach' trees, go to step 10b else continue with step 10a.

10a) Run **ksrcconf -toolbox contrib_toolbox** to create the symbolic links for the new directory in the 'srcmach' trees.

11a) Run **kmakeall -toolbox contrib_toolbox Makefile** to update the *Makefile* in the 'srcmach' trees.

12a) cd into the new directory, *vbar*.

13a) Run **kmakeall -toolbox contrib_toolbox install** to make, in any 'srcmach' tree, the new program and to install it.

Adding the routine *vbar* to *arith* is finished.

10b) cd into the new directory, *vbar*.

11b) Run **make depend** to create the file dependences in the *Makefile*.

12b) Run **make install** to compile the new files and to install the new program. Do not combine this step with step 11b above, i.e. **make depend install**, because the *Makefile* used for the 'install' would be the old one and not the one with the dependencies added.

Adding the routine *vbar* to *arith* is finished.

B.8.4 Creating a New Program (dependent on X11)

The steps involved in adding a new program, dependent on X11, is exactly the same as described in the above example with the exception that step 4) needs to be:

4) Run **imkmf -type xprog -name vbar -toolbox contrib_toolbox** to create the *Imakefile*. The *xprogname* argument should be the name of the X11 program. If the routine is dependent on Fortran then use **imkmf -type fxprog -name xprogname -toolbox contrib_toolbox**.

B.8.5 Creating a New Shell Script

The steps involved in adding a new shell script are exactly the same as described above in the "Creating a New Program" section with the exception that step 4) needs to be:

4) Run **imkmf -type script -toolbox contrib_toolbox -name** *scriptname* to create the *Imakefile*. The *scriptname* argument should be the name of the 'main' script, assuming that several scripts are needed to help the 'main' script.

All script files should end with the suffix ".csh".

B.8.6 Moving Files within a Khoros Toolbox

If the only things being moved are files, no directories, then the following steps need to be taken to insure that the *Imakefiles* located in the 'original' and 'destination' directories are up to date. For example, assume that files *lvclip.c* and *lvtranslat.c* need to be moved from *$CONTRIB_TOOLBOX/src/vipl/Lib* to *$CONTRIB_TOOLBOX/src/dspl/Lib*.

In the steps below, if there are no 'srcmach' trees, then just use **make** for the make command otherwise use **make** followed by **kmakeall -toolbox contrib_toolbox**.

1) cd into the 'original' directory, in our example *$CONTRIB_TOOLBOX/src/vipl/Lib*.

2) Move the required files from the current directory to *$CONTRIB_TOOLBOX/src/dspl/Lib*, for example: `mv lvclip.c lvtranslat.c $CONTRIB_TOOLBOX/src/dspl/Lib`.

3) Remove the object files (*lvclip.o* and *lvtranslat.o*) associated with the files that were moved. This can be done either running **make clean**, which removes all objects, or they can be removed by hand, for example: `rm lvclip.o lvtranslat.o`.

4) Run **imkmf** to update the *Imakefile* (to remove references to the moved files, *lvclip.c*, *lvtranslat.c*, *lvclip.o*, and *lvtranslat.o*, from the SRCS, OBJS, FSRCS, FOBJS, HEADERS lists).

5) Run **make Makefile** to update the *Makefile* to reflect the changes made to the *Imakefile* in step 4 above.

6) Run **make depend** to add the file dependencies to the *Makefile*. Do not combine this step with step 5 above, i.e. **make Makefile depend**, because the *Makefile* that is used for the 'depend' would be the old one and not the new one which has the references to the moved files removed.

7) Run **make install** to make sure moving the files did not cause any **make** problems. Do not combine this step with step 6 above, i.e. **make depend install**, because the *Makefile* that is used for the 'install' would be the old one and not the one with the dependencies added.

8) cd into the 'destination' directory, in our example *$CONTRIB_TOOLBOX/src/dspl/Lib*.

9) Run **imkmf** to update the *Imakefile* (to add references to the moved files, *lvclip.c*, *lvtranslat.c*, *lvclip.o*, and *lvtranslat.o*, to the SRCS, OBJS, FSRCS, FOBJS,

HEADERS lists).

10) Run **make Makefile** to update the *Makefile* to reflect the changes made to the *Imakefile* in step 9 above.
11) Run **make depend** to add the file dependencies to the *Makefile*. Do not combine this step with step 10 above, i.e. **make Makefile depend**, because the *Makefile* that is used for the 'depend' would be the old one and not the new one which has references to the moved files inserted.
12) Run **make install** to make sure moving the files did not create any unresolved dependencies. Do not combine this step with step 11 above, i.e. **make depend install**, because the *Makefile* that is used for the 'install' would be the old one and not the one with the dependencies added.
13) Moving the files, *lvclip.c* and *lvtranslat.c*, is finished.

B.8.7 Moving a Directory within a Khoros Toolbox

If a directory is all that is being moved, no individual files, then the following steps need to be taken to insure that the *Imakefile* located in the 'original' and 'destination' directories are up to date. For example, assume that the directory *vadd* needs to be moved from *$CONTRIB_TOOLBOX/src/vipl/arith_binary* to *$CONTRIB_TOOLBOX/src/vipl/arith_unary*.

In the steps below, if there are no 'srcmach' trees, then just use **make** for the make command otherwise use both **make** and **kmakeall -toolbox contrib_toolbox**. Again, it is best to use the Khoros utilities, **kdinstall**, **krmprog**, and **kinstall** to remove or move either programs within the Khoros environment. The method outlined below should be followed when directories are being moved manually.

1) cd into the 'original' directory, *$CONTRIB_TOOLBOX/src/vipl/arith_binary*.
2) Move the required directory, *vadd* from the current directory to *$CONTRIB_TOOLBOX/src/vipl/arith_unary*, for example:

   ```
   mv vadd $CONTRIB_TOOLBOX/src/vipl/arith_unary.
   ```
3) Edit the configuration file, *vadd.conf*, to reflect the change of location of the sources.
4) Run **imkmf** to update the *Imakefile* (to remove the reference to the moved directory, *arith_binary*, from the SUBDIRS list).
5) Run **make Makefile** to update the 'original' directory *Makefile* to reflect the changes made to the *Imakefile* in step 3 above.
6) cd into the 'destination' directory, *$CONTRIB_TOOLBOX/src/vipl/arith_unary*.
7) Run **imkmf** to update the *Imakefile* (to add a reference to the moved directory, *arith_unary*, to the SUBDIRS list).
8) Run **make Makefile** to update the *Makefile* to reflect the changes made to the *Imakefile* in step 6 above.
9) Run **make install** to make sure moving the directory did not create any unresolved dependencies.
10) If the name of the directory moved was changed, for example from *vadd* to *addition*, then all files associated with the old name need to be removed by hand; i.e. remove *$CONTRIB_TOOLBOX/bin/vadd*. If the directory moved was a library, for example *$CONTRIB_TOOLBOX/src/Lib/vrast*, and the name was changed from *libvrast.a* to *libvraster.a*, then the old library would need to be removed by hand;

i.e. remove *$CONTRIB_TOOLBOX/lib/libvrast.a*.

11) Moving a directory is finished.

C. MAINTAINING THE KHOROS SRCMACH TREE FOR MULTIPLE ARCHITECTURES

This section of the manual provides information for setting up and maintaining Khoros for multiple architectures. Information in this section applies to both KHOROS_HOME and Khoros toolboxes. Briefly, the steps necessary for configuring Khoros for multiple architectures includes the following:

(1) - Create the libmach directories for each architecture to be supported.

(2) - Create the binmach directories for each architecture to be supported.

(3) - Modify or create the mach file(s) in KHOROS_HOME/repos/config/src_conf or in {toolbox}/repos/config/src_conf for the appropriate paths and machine names for each architecture.

(4) - Create the srcmach tree for each architecture. This consists of creating a directory structure that mimics that of the Khoros source tree, and making symbolic links from the srcmach tree to the Khoros source tree.

(5) - Compile the system for each architecture.

The remainder of this document provides more detailed information on each step outlined above. Several tool programs are provided, which help in creating directories and making links to files in source. The use of these programs is explained in the sections that follow.

Khoros should be installed for a single architecture prior to establishing a multiple architecture system. This will ensure that all programs compile clean and that there are no configuration problems. The *Installation Guide* provides the user with the necessary information for setting up Khoros for a single architecture.

Before proceeding to set up a system for multiple architectures, the user should be aware of some of the possible configurations that may arise on a networked system of machines. As described in Section A of this chapter, the directory structure of Khoros includes src, repos, lib, include, bin, and data. To support multiple architectures, you would also have srcmach, libmach, and binmach directories which would contain subdirectories for each architecture. Because disk storage is a precious commodity, you might choose to store the Khoros source directories on a different disk from that of the binaries and libraries. Note that the environment variable, *KHOROS_HOME*, points to the top level of the Khoros src directory.

There are three common scenarios for setting up Khoros for multiple architectures on a networked system. One scenario is to have the srcmach, bins, and libs on a local partition and the src directory on another machine that is mounted on the local partition. Another scenario is to have the binaries and libraries on a local partition, and the src and srcmach on a remote disk that is mounted on the local partition. The last scenario is to have the src, srcmach, binaries, and libraries on a remote disk that is mounted locally. The following diagram depicts these three scenarios:

```
Scenario 1 - src located on a remote disk

     Machine A                    Machine B
      (local)                      (remote)
     ----------     mounted       ----------
     |        |<----------------| khoros  |
     |--------|                 |         |
     | srcmach|                 |         |
     |  bins  |                 |         |
     |  libs  |                 |         |
     ----------                   ----------

Scenario 2 - src and srcmach located on a remote disk

     Machine A                    Machine B
      (local)                      (remote)
     ----------     mounted       ----------
     |        |<----------------| khoros  |
     |        |<----------------| srcmach|
     |--------|                 |         |
     |  bins  |                 |         |
     |  libs  |                 |         |
     ----------                   ----------

Scenario 3 - src, srcmach, bins, and libs located on a remote disk

     Machine A                    Machine B
      (local)                      (remote)
     ----------     mounted       ----------
     |        |<----------------| khoros  |
     |        |<----------------| srcmach|
     |        |<----------------|  bins   |
     |        |<----------------|  libs   |
     |        |                 |         |
     ----------                   ----------

* khoros - includes the directories of src, include, repos, and data.
```

After it is determined which scenario best fits a particular site, the next step is to create the directories for the *libmach* and *binmach*. We will now provide a procedure for doing this.

The first step in setting up Khoros for multiple architectures is to create a *libmach* directory that will have a separate subdirectory for each architecture to be supported. For example, if both dec and sun architectures are supported, then *libdecmips* and *libsun4* directories are created beneath the *libmach* directory. Each of these directories will contain the compiled libraries for each supported architecture. The directory *lib* is just a symbolic link to the libmach directory for the local host machine architecture. This is accomplished by creating a */localhost/khoros* directory and creating a link between the *lib* directory and the libmach directory on each machine. Thus, accessing the directory *lib* on a sun4 machine will actually get you the libmach directory for the sun4 architecture, *libsun4*. This step must be performed for each machine architecture to be supported.

This step can best be illustrated by an example: (on a sun4 machine)

```
% mkdir /localhost/khoros
```

```
% ln -s ~khoros/libmach/libsun4 lib
% cd khoros
% ln -s /localhost/khoros/lib lib
```

The next step is to create a *binmach* directory that will contain the compiled libraries for each architecture. Similar to the *libmach* directory discussed above, a *binmach* directory is created with a separate subdirectory for each architecture. Using the same example as above, separate subdirectories will be created for *bindecmips* and *binsun4*. Each of these directories will contain the compiled binaries for each supported architecture. Similar to the *lib* directory discussed above, the *bin* directory is just a symbolic link to the local host libmach for that machine architecture. This step must be performed for each machine architecture to be supported. A symbolic link can be set up for the local host in the following manner:

```
% mkdir /localhost/khoros
% ln -s ~khoros/binmach/binsun4 bin
% cd khoros
% ln -s /localhost/khoros/bin bin
```

The following diagram shows a portion of the Khoros directory structure for support of multiple architectures. It illustrates the organization of the *libmach* and *binmach* directories as described above would appear if the local host machine is a decmips.

```
/localhost/khoros/bin@  -> binmach/bindecmips

/localhost/khoros/lib@  -> libmach/libdecmips


             |--bin@ -> /localhost/khoros/bin
             |
             |                  |-bindecmips----
             |--binmach---------|
             |                  |-binsun4-------
             |
../khoros---|
             |
             |                  |-libdecmips----
             |--libmach---------|
             |                  |-libsun4-------
             |
             |--lib@ -> /localhost/khoros/lib
```

This is followed by creating a srcmach tree for each architecture, creating symbolic links to the files in the Khoros source tree, and compiling for each architecture. Creating a srcmach tree consists of creating a directory structure that mimics the directory structure of the Khoros source tree. This is followed by creating symbolic links from the srcmach tree to each of the files in the Khoros source tree. The final step is to compile the system for each of the architectures.

There are several programs that can be used to help create symbolically linked source trees when your site supports multiple machine architectures. Two programs that can be used to create a symbolically linked source tree are **klndir** and **ksrcconf**. **Klndir** is a low level tool used to create a symbolically linked source tree for a particular architecture, whereas **ksrcconf** can be used to create multiple source trees at once. **Ksrcconf** will take care of creating the source tree directories for each architecture and make the links to the source files. **Kmakeall** is a tool to ease the process of compiling the source for each architecture. **Ksrcconf** and **kmakeall** both depend on *mach* files, whereas **klndir** does not. Each of these tools will be discussed in the sections that follow.

The concepts described above were applied to KHOROS_HOME. These same concepts can be applied to Khoros toolboxes. Since many of the tool programs in Khoros make use of *mach* files, we will begin by discussing the creation and use of mach files.

C.1 MACH FILES AND MULTIPLE ARCHITECTURES

Ksrcconf and **kmakeall** use a *mach* file to sort out the correct paths to the top level of source for each machine. A mach file contains information specific for a particular installation site. It contains the name of the Khoros user, the names of the machines, and the proper paths to the source and srcmach directories for each of the machine architectures that source will be compiled on. These routines also require that an ".rhosts.bk" file be located in the top level directory of KHOROS_USER, for access to the machine names and login IDs.

Note that a mach file must be setup even if you are not going to support multiple architectures. That is you need a mach file even if you are working with a single architectures.

The five entries in the mach file are described below:

KHOROS_USER	specifies the name of the user. This is used to ensure proper ownership and permissions of files, required for rsh's, and will mail to the user the results of compilations and installations. NOTE, This is a required field.
LOCAL_SRC_TOP	specifies the complete path to the top of source for the machine that is used in the development of the Khoros system. (See explanation below). NOTE, this is a required field.
KHOROS_MACHINES	specifies the name of the machines that source will be compiled on. This field only needs to be specified if multiple architectures are to be supported.
KHOROS_MACH_DIR	specifies the complete paths to the srcmach directories for each machine named in the KHOROS_MACHINES entry. This field only needs to be specified if multiple architectures are to be supported.
KHOROS_SRC_TOP	specifies the complete paths to the top of source for each machine named in the KHOROS_MACHINES entry. This field only needs to be specified if multiple architectures are to be supported.

`$KHOROS_HOME/repos/config/src_conf` is the location for KHOROS_HOME mach files. If your site has one or more machines with the same paths to the Khoros source tree, then one mach file may be used to specify the machine information for Khoros home. A mach file must also be created for those sites supporting Khoros toolboxes. Each toolbox will contain its own mach file(s), located in `{toolbox}/repos/config/src_conf`. Consult Volume 2, Chapter 1 for information on creating and managing toolboxes using **kraftsman**. Several examples of different mach files are presented in the examples that follow.

The paths for $KHOROS_HOME and and the top level of toolboxes must be correct for your site. The user should be aware of aliases for "pwd". For example, if you have "pwd" aliased to "cwd", then you may get an incomplete path if your site is using *automount* or if your partitions are symbolically linked so that Khoros is available in a different location on your disk. To avoid this problem, the user may want to unalias the "pwd" command, or use "wd" to get the correct path to the top of source.

One other problem that may occur, is that the hostnames for the machines at your site may not be qualified with an extension. For example, if your site has a machine name of "mymachine", it may be qualified with an extension, such as "mymachine.xyz.edu". You should verify the correct hostname of all machines at your site by logging on to the machine and entering the command, `hostname`. This will return the correct hostname for that machine, and this is what should be entered in KHOROS_MACHINES field of the mach file.

The following is an example of a mach file for Khoros that supports two different machine architectures:

set KHOROS_USER = (khoros)
set KHOROS_MACHINES = (dec.xyz.edu sun.xyz.edu)
set KHOROS_MACH_DIR = ($KHOROS_HOME/srcmach/srcdecmips $KHOROS_HOME/src
mach/srcsun4)
set KHOROS_SRC_TOP = ($KHOROS_HOME/src $KHOROS_HOME/src)
set LOCAL_SRC_TOP = (/local/khoros/src)

If the full paths to the top of the Khoros source tree are different for each of the machines at a site, then a mach file must be created for each of the machines. To create a mach file for a particular site, one needs to list the machine names for each of the architectures that are supported. The path to the srcmach directories for each machine must also be provided in the KHOROS_MACH_DIR field. In addition, the path to the top of the source for each of the machines must also be provided. Note that there is a one-to-one correspondence for each entry in the fields, KHOROS_MACHINES, KHOROS_MACH_DIR, and KHOROS_SRC_TOP.

The field LOCAL_SRC_TOP specifies the path to the top of source. For some sites, this path may be different for each of the machines that source will be compiled on. If the paths to source are different for each machine, then it is necessary to create a mach file for each machine. For example, if your site supports two machine architectures with different paths to the top of source, then a mach file should be created for each architecture. In addition, the paths to the top of the Khoros source tree must be checked for any networked machine that is used for developing or managing Khoros. If the path to the top of the Khoros source tree is different, then a separate mach file must also be created for use with that particular machine.

The programs that use mach files check the current machine name and will use the appropriate mach file, if one exists for that machine. For example, if your current machine name is sun.xyz.edu, a search is made in the repos/config/src_conf directory for a mach file named "default_mach.sun.xyz.edu" for KHOROS_HOME and "{toolbox}_mf.sun.xyz.edu" for a toolbox. If the mach file is not found, then the search will continue in the repos/config/src_conf directory for "default_mach" for KHOROS_HOME and "{toolbox}_mf for a toolbox. If a mach file is not found, the program will terminate and a mach file will have to be created.

The next example illustrates a site supporting two architectures when the paths to the top of Khoros source are different for each of the machines. In this case two mach files are required. The two mach files are named using the machine name as the extension. If the machine names are dec.xyz.edu and sun.xyz.edu, then the mach files would be named, default_mach.dec.xyz.edu and default_mach.sun.xyz.edu. Note, this example applies to the mach files in KHOROS_HOME, however this concept can be applied to the mach files in a Khoros Toolbox. Examples of these two files are presented below:

```
#  default_mach.dec.xyz.edu

set KHOROS_USER = (khoros)
set KHOROS_MACHINES = (dec.xyz.edu  sun.xyz.edu)
set KHOROS_MACH_DIR = ($KHOROS_HOME/srcmach/srcdecmips  $KHOROS_HOME/src
mach/srcsun4)
set KHOROS_SRC_TOP = ($KHOROS_HOME/src $KHOROS_HOME/src)
set LOCAL_SRC_TOP = (/local/khoros/src)
```

```
#  default_mach.sun.xyz.edu

set KHOROS_USER = (khoros)
set KHOROS_MACHINES = (dec.xyz.edu  sun.xyz.edu)
set KHOROS_MACH_DIR = ($KHOROS_HOME/srcmach/srcdecmips  $KHOROS_HOME/src
mach/srcsun4)
set KHOROS_SRC_TOP = ($KHOROS_HOME/src $KHOROS_HOME/src)
set LOCAL_SRC_TOP = (/foo/local/khoros/src)
```

Notice that the paths for the two machines are different for LOCAL_SRC_TOP. Since the tools make use of these paths to install and configure srcmach trees, it is important to recognize these differences for successful maintenance of the Khoros source tree.

The next example shows how a mach file may be set up for support of a Khoros toolbox srcmach for two different architectures. In this example, let's assume that the paths to the top of Khoros source are the same. This enables the use of one mach file to define the paths for both architectures. We will call the Khoros toolbox "CONTRIB_TOOLBOX." An example of a mach file for a Khoros toolbox is presented below:

```
#  contrib_toolbox_mf

set KHOROS_USER = (spanky)
set KHOROS_MACHINES = (dec.xyz.edu  sun.xyz.edu)
set KHOROS_MACH_DIR = ($CONTRIB_TOOLBOX/srcmach/srcdecmips  $CONTRIB_TOOLBOX
RIB/srcmach/srcsun4)
set KHOROS_SRC_TOP = ($CONTRIB_TOOLBOX/src $CONTRIB_TOOLBOX/src)
set LOCAL_SRC_TOP = (/usr/users/spanky/contrib_toolbox/src)
```

The following sections discuss each of the tool programs involved in building and maintaining the Khoros srcmach tree for multiple architectures.

C.2 CREATE A SYMBOLIC LINKED SOURCE TREE - KLNDIR

Klndir is a program for creating directories in a srcmach tree and creating symbolic links to the files in the corresponding directories in the Khoros source tree. **Klndir** first checks to see if the destination directories in the srcmach tree exist. If they do not exist, then it creates the directories and makes the symbolic links to the *.c, *.h, *.f source files. It copies over the source Makefiles, and makes symbolic links to the Imakefiles.

The usage for **klndir** is as follows:

```
 % klndir -source xxxx -dest yyyy
```

where,
xxxx is the absolute path to a directory in source from /.

yyyy is the absolute path to a directory in the srcmach from /.

The user specifies the FULL directory path to the Khoros source tree and the destination srcmach tree. The Khoros source tree is the location of the source files for Khoros. The destination source tree is a new source tree containing symbolic links to the source (*.c, *.h, and *.f files), and is referred to as a srcmach tree. The user may retrieve the man page for **klndir** for a complete description and usage information.

The following example demonstrates the usage for **klndir**:

```
% klndir -source /foo/khoros/src -dest /foo/khoros/srcmach/foomach
```

This will make links from the khoros source tree to the foomachine source tree.

C.3 CREATE MULTIPLE SYMBOLIC LINKED SOURCE TREES - KSRCCONF

Ksrcconf is a more powerful and higher level tool than **klndir**, since it can create multiple srcmach trees at once. Since **Ksrcconf** calls **klndir** for each directory in the srcmach directory, it is the preferred tool to use when creating srcmach trees for multiple machine architectures. By default, **ksrcconf** will attempt to create symbolic links relative to the present working directory. **Ksrcconf** assumes that there is a directory called srcmach. In the srcmach directory are subdirectories for each machine architecture, and for each of these subdirectories it will update and create the symbolic links. The paths to each subdirectory of the srcmach directory are specified in the *mach file*.

The usage for **ksrcconf** is as follows:

```
 % ksrcconf [-source xxxx] [-toolbox yyyy]
```

where,
xxxx is the source directory

yyyy is the name of the toolbox you are working in. If you

you are working in KHOROS_HOME, do not specify the
the -toolbox option.

A complete description and program usage may be obtained by retrieving the man page for **ksrcconf**.

The following examples may be helpful:

```
% cd $KHOROS_HOME/src/vipl/arith_binary
% ksrcconf
```

OR

```
% ksrcconf -source $KHOROS_HOME/src/vipl/arith_binary
```

Here is an example of setting up the symbolic links for all of $KHOROS_HOME/src:

```
% cd $KHOROS_HOME/src
% ksrcconf
```

The next example shows how to configure the srcmach directory for a Khoros toolbox. We will call the toolbox "contrib_toolbox".

cd to the source directory in the "contrib_toolbox" toolbox. Execute:

```
% ksrcconf -toolbox contrib_toolbox
```

ksrcconf will look in $CONTRIB_TOOLBOX/repos/config/src_conf for the mach file.

C.4 COMPILING MULTIPLE ARCHITECTURES - KMAKEALL

Once the srcmach trees are in place and the bin and lib directories for the multiple architectures have been created, **kmakeall** can be used to compile all of the architectures at once. **Kmakeall** is a shell script to assist in the use of **make** on a distributed computer system. By default, **kmakeall** will attempt to make source relative to the present working directory.

The usage for **kmakeall** is as follows:

```
% kmakeall [-toolbox] [-f full_path | -r relative_path]
                [make options]
```

where,

[-toolbox] the Toolbox name
[-f] is the full path
[-r] is relative path to the top level src directory

The user can specify the FULL directory path name (-f) to the source, or the relative path name (-r) to the src directory. A complete description and usage of **kmakeall** can be obtained at any time by retrieving the man page for **kmakeall**.

After getting the defined make objects and options, **kmakeall** attempts to call *rsh* to each machine and call **make** in the appropriate $KHOROS_HOME/srcmach/ directory. **Kmakeall** uses *rsh*, and since ".rhosts" files should not be left lying around, **kmakeall** copies the

$KHOROS_USER/.rhosts.bk to $KHOROS_USER/.rhosts before calling *rsh* and then removes the ".rhosts" file before exiting. Since the *rsh*'s are put in the background, **kmakeall** needs to sleep for about a minute to allow *rsh*'s to be connected before removing the ".rhosts" file.

Here is an example of making and installing **vadd** $KHOROS_HOME on all machines:

```
% cd $KHOROS_HOME/src/vipl/arith_binary/vadd
% kmakeall install
```

Here is an example of making and installing all of $KHOROS_HOME/src/file_formats on all machines:

```
% cd $KHOROS_HOME/src/file_formats
% ksrcconf
% kmakeall install
```

Note that on a busy network and/or a slow server, you may experience timeout problems. Therefore one should be cognizant of the network load when compiling.

The following example illustrates how to compile all of the source in a Khoros toolbox. We will call the toolbox CONTRIB_TOOLBOX

```
% cd CONTRIB_TOOLBOX/src
% kmakeall install -toolbox contrib_toolbox
```

CONTENTS

Chapter 9 - CONFIGURATION MANAGEMENT