Migration Toolbox Manual

Chapter 1

Introduction

Introduction to Migration Toolbox

Chapter 1 - Introduction

A. Introduction

The migration toolbox provides tools and documentation to aid in the conversion of Khoros 1.0 programs, workspaces, and toolboxes into the Khoros 2.0 software system. This toolbox also includes tools to update pre-release versions of Khoros software or data.

The migration toolbox depends on the toolboxes BOOTSTRAP and DATASERV in order to function. These toolboxes must be installed before compiling the migration toolbox.

The table below provides a complete list of utilities that exist within the migration toolbox. These programs will be explained in more detail in the chapters following this introduction.

Toolbox GUI Operators: migration			
Category Subcategory	**Operator**	**Description**	**Executable**
Migration Tools	* Make Prototype	Make prototypes for functions	`mkproto`
Code Migration	* Update Code	Update source code from khoros 1.0 library calls to 2.0 calls	`kfix`
Migration Tools Object Migration	* DB conversion	Convert from a pre-Khoros 2.0 kdbm file to a 2.0 kdbm file	`dbcnvrt`
Migration Tools	* Fix Pane File	Update Pane UIS File to Khoros 2.0 Conventions	`kfixpane`
UIS Migration	* Update UIS File	Updates an old Khoros 1.0 Patch 5 UIS file to a new Khoros 2.0 UIS file.	`uisupdate`
Natural Evolution	* Migrate 1.0 vroutine	A program to evolve 1.0 programs to 2.0 objects	`kdarwin`

It is important to note, that there are specific circumstances that dictate when you should use these programs. When converting a Khoros 1.0 vroutine, the only program you will directly run is kdarwin. This program does most of the work necessary to convert a Khoros 1.0 vroutine to a Khoros 2.0 software object, and it makes calls to uisupdate, kfixpane, mkproto, and kfix. Thus, for vroutine conversion, it is important to know what all the programs do, but you will only directly execute kdarwin.

For xvroutines, the way you work in Khoros 2.0 has significantly changed since Khoros 1.0, so no direct conversion program is possible. In this case, you should use uisupdate, mkproto, and kfix to convert the GUI, make function prototypes, and fix non-generated source code. Nearly all of the Khoros 1.0 X library calls will have to be converted to the Khoros 2.0 equivalent to take advantage of the widget abstraction (See Programming Services Manual Volume III).

A.1. The Migration Process

The first part of this introduction has given a brief overview of the tools that exist to assist for converting your Khoros 1.0 code to Khoros 2.0. This section deals with the process of migrating code to take full advantage of the Khoros 2.0 infrastructure. This process begins with the use of the migration tools, discussed above and in great deal in the following chapters, to move into the Khoros 2.0 development environment quickly.

The quickest way to begin using your Khoros 1.0 environment in Khoros 2.0 is to use uisupdate and kfixpane to convert the .pane files of you're Khoros 1.0 routines into usable 2.0 panes. You can create a number of pane objects in a toolbox and use the converted panes with -R lines that point to old 1.0 binaries in cantata. This is the quickest and least painful way to use 1.0 data processing algorithms in Khoros 2.0. Unfortunately, this implies that you must have a Khoros 1.0 source tree around to support those 1.0 binaries, and it all must be maintained separately from the Khoros 2.0 pane objects. Thus, this method is good for a quick startup to Khoros 2.0, but it is recommended that you convert the code to Khoros 2.0 as soon as possible.

To accomplish this next step, we provide the kdarwin routine. This routine takes a Khoros 1.0 vroutine's prog and pane file and generate a complete Khoros 2.0 software object with source code and documentation. All the routines in the RETRO toolbox are examples of routines that have been converted to Khoros 2.0 using kdarwin. This used of the kdarwin routine is explained in more detail in Chapter 4 of this manual. With the vroutine converted to a real Khoros 2.0 kroutine object, it can use some of the advantages of the Khoros 2.0 infrastructure. Namely, the *readimage* routine uses data services to read input images, and then forces the data into an xvimage structure. This can be a lossy conversion if the underlying data file has data that will not fit into an xvimage; however, it allows the kroutine to read all the data formats that are supported by Khoros via the data object abstraction. Once all the Khoros 1.0 vroutines have been converted into Khoros 2.0 kroutines and they are competely supported by the Khoros 2.0 development tools, you can begin the final phase of migration.

Routines converted with kdarwin have most of the same limitations as the Khoros 1.0 vroutines. Namely they cannot operate on large data sets and often are very limited in the data types they can operate on, requiring that many conversions be performed on the data between operators in the visual language. Hence, the final stage of migration is to rewrite the kroutine to take full advantage of data services and the polymorphic data model. In this manner both visual programmers and toolbox programmers can use the data processing routines you create. Routines created with kdarwin have library functions that will only accept pointers to xvimage structures, rather than the kobject pointer is used by the Khoros 2.0 infrastructure. This makes it very difficult to combine library calls from the new 2.0 libraries, and the calls created by kdarwin.

Finally, as you have no doubt noticed, there is very little support for the conversion of Khoros 1.0 xvroutines to Khoros 2.0 xvroutines. There are several reasons for this. First, the paradigm has changed considerably since Khoros 1.0, and most often there is no direct conversion from the Khoros 1.0 xv, Xt, and Xlib calls in the xvroutine to the new Khoros 2.0 widget abstraction. Next, the code generator has been changed to allow regeneration of source code without losing the user written code stubs. Moving the Khoros 1.0 code stubs to the new structure must be done by hand anyway, since it is not obvious where these sections will be. Finally, in Khoros 1.0, reuse of xv code meant stripping the code out of a preexisting xvroutine and modifying it to your needs. Under Khoros 2.0, these commonly used bits of code were extracted and generalized into Xt Object classes in the xvisual library. Since the API for interacting with these visual objects is completely different from what was available in Khoros 1.0, practical conversion tools beyond a program to migrate the form file was not possible.

Table of Contents

This page left intentionally blank