Migration Toolbox Manual

Chapter 2

Source Code Conversion

Programs that convert Khoros 1.0 source code files to Khoros 2.0 files

Chapter 2 - Source Code Conversion

A. Introduction to Source Code Conversion Routines

This chapter explains the low level source code conversion routines that help users convert standard C or Khoros 1.0 code into ANSI compliant C code that uses the library calls in the khoros libraries. These routines include, kfix and mkproto.

A.1. kfix — *Update source code from khoros 1.0 library calls to 2.0 calls*

Object Information

Category:	*Migration Tools*
Subcategory:	*Code Migration*
Operator:	*Update Code*
Object Type:	*script*
In Cantata:	*No*

Description

This program converts source code written to Khoros 1.0 or to normal unix system calls to use the equivalent library calls that exist in Khoros 2.0. The user of the program can choose to fix some or all of the routine calls according to a several categories. The supported conversions are bsd library to posix library calls, Khoros 1.0 xvimage input data functions to Khoros 2.0 retro toolbox calls, vexpr to kexpr, vmath to kmath, Khoros 1.0 transport calls and unix file system calls to 2.0 transport and OS abstraction calls, and Khoros 1.0 string and general utility calls to Khoros 2.0 utility calls.

When it converts, it saves the original file as its previous name with a .old extension. Also, there are times when C syntax cannot be parsed easily with a regular expression parser, so it marks that code with '####' followed by the routine name in all capitol letters. Thus, you can just do a grep for '####' to find any code that must be converted by hand.

Note, most often, this shell script will be called from kdarwin while converting an old vroutine to a kroutine in the new object system.

Examples

This sectino has two examples of how kfix is used. These two examples show the two modes of operation available to kfix. In the first mode, it scans the current directory for source code files, and offers the user a menu allowing the user to choose which library conversions to perform.

The default operation of kfix is to scan the current directory for all files ending in .c, and to giving the user a menu of conversions that can be performed on each file. In this mode, it will also offer to convert the .h files. The following example illustrates how this mode of operation works.

```
% ls
config.c     ghost.h      internals.h  keynames.c   readprog.c

% kfix

NOTICE: '.h' files exist.  Do you want to convert them also? (y/n) y

Khoros 2.0 conversion script

This script will convert library calls for the following libraries, or all of the 2.0 libraries.

 1.  Convert bsd2posix library calls
 2.  Convert dataserv library calls
 3.  Convert kexpr library calls
 4.  Convert kmath library calls
 5.  Convert ktransport library calls
 6.  Convert kutils library calls
 7.  Convert all library calls

Please enter your selection: 7

 Converting config.c for bsd2posix library calls
 Converting config.c for dataserv library calls
 Converting config.c for kexpr library calls
 Converting config.c for kmath library calls
 Converting config.c for ktransport library calls
 Converting config.c for kutils library calls

 Converting keynames.c for bsd2posix library calls
 Converting keynames.c for dataserv library calls
 Converting keynames.c for kexpr library calls
 Converting keynames.c for kmath library calls
 Converting keynames.c for ktransport library calls
 Converting keynames.c for kutils library calls

 Converting readprog.c for bsd2posix library calls
 Converting readprog.c for dataserv library calls
 Converting readprog.c for kexpr library calls
 Converting readprog.c for kmath library calls
 Converting readprog.c for ktransport library calls
 Converting readprog.c for kutils library calls

 Converting ghost.h for bsd2posix library calls
 Converting ghost.h for dataserv library calls
 Converting ghost.h for kexpr library calls
 Converting ghost.h for kmath library calls
 Converting ghost.h for ktransport library calls
 Converting ghost.h for kutils library calls
```

```
 Converting internals.h for bsd2posix library calls
 Converting internals.h for dataserv library calls
 Converting internals.h for kexpr library calls
 Converting internals.h for kmath library calls
 Converting internals.h for ktransport library calls
 Converting internals.h for kutils library calls

% ls
config.c       ghost.h.old     keynames.c      readprog.c.old
config.c.old    internals.h     keynames.c.old
ghost.h        internals.h.old  readprog.c
```

This next example shows how to convert just one file.

```
% ls ˜joe/cfiles
myfile.c  other.c

% kfix ˜joe/cfiles/myfile.c

Khoros 2.0 conversion script

This script will convert all 2.0 library calls.


Converting /home/joe/cfiles/myfile.c for bsd2posix library calls
Converting /home/joe/cfiles/myfile.c for dataserv library calls
Converting /home/joe/cfiles/myfile.c for kexpr library calls
Converting /home/joe/cfiles/myfile.c for kmath library calls
Converting /home/joe/cfiles/myfile.c for ktransport library calls
Converting /home/joe/cfiles/myfile.c for kutils library calls

% ls ˜joe/cfiles
myfile.c     myfile.c.old  other.c
```

See Also

kdarwin(1), sed(1), csh(1)

Restrictions

This program uses unix regular expressions and the substitution capabilities of sed to implement the conversions.

References

None

A.2. mkproto — *Make prototypes for functions*

Object Information

Category:	*Migration Tools*
Subcategory:	*Code Migration*
Operator:	*Make Prototype*
Object Type:	*kroutine*
In Cantata:	*No*

Description

Mkproto takes a re-directed input from standard in of one C source code file, and produces as output (on the standard output stream) a list of function prototypes (a la ANSI) for the external functions defined in the given source files. This output, redirected to a file, is suitable for #include'ing in a C source file.

It can be executed as follows:

```
% mkproto {options} < file.c
```

or,

```
% mkproto {options} < file.c > file.h
```

The function definitions in the original source may be either "old-style" (in which case appropriate prototypes are generated for the functions) or "new-style" (in which the definition includes a prototype already).

A -n option causes the line number where each function was defined to be prepended to the prototype declaration as a comment.

A -s option causes prototypes to be generated for functions declard "static" as well as extern functions.

A -p option causes the prototypes emitted to be only readable by ANSI compilers. Normally, the prototypes are "macro-ized" so that compilers with __STDC__ not defined don't see them.

See Also

cc(1), lint(1)

Restrictions

Mkproto is easily confused by complicated declarations, such as

```
int ((*signal)())() { ...
```

or

    struct foo { int x, y; } foofunc() { ...

Float types are not properly promoted in old style definitions, i.e.

    int test(f) float f; { ...

should (because of the default type conversion rules) have prototype

    int test(double f);

rather than the incorrect

    int test(float f);

generated by mkproto.

Some programs may need to be run through the preprocessor before being run through mkproto. The -n option is unlikely to work as desired on the output of a preprocessor.

Typedef'd types aren't correctly promoted, e.g. for

    typedef schar char; int foo(x) schar x;...

mkproto incorrectly generates the prototype int foo(schar x) rather than the (correct) int foo(int x).

Functions named "inline" with no explicit type qualifiers are not recognized.

References

Original source from:

Eric R. Smith email:
Dept. of Mathematics ersmith@uwovax.uwo.ca
University of Western Ontario ersmith@uwovax.bitnet
London, Ont. Canada N6A 5B7
ph: (519) 661-3638

Command Line Arguments

```
Usage for mkproto: Make prototypes for functions
% mkproto
  [-n]       (boolean) put line numbers of declarations as comments
             (default = false)
  [-s]       (boolean) include declarations for static functions
             (default = false)
  [-v]       (boolean) include variable name in declaration
             (default = false)
  [-d]       (boolean) include declaration for the PROTO() definition
             (default = false)
  [-p]       (boolean) do not make header files readable by non-ANSI compiler
             (default = true)
  [-h]       (boolean) do file headers
             (default = true)
```

Table of Contents

This page left intentionally blank