Support Toolbox Manual

Chapter 7

Perl

The Perl Programming Language

Copyright © 1993, 1994 Khoral Research, Inc. All rights reserved.

Chapter 7 - Perl

A. Introduction

An ever-increasing number of software objects in Khoros are written using the *Perl* programming language, created by Larry Wall † The main reasons and advantages are:

- Perl contains a mixture of features from languages including awk, sed, csh, sh and C. It also includes direct support for networking, most system calls, and must of the C runtime library.
- Perl is freely available, runs on more machines than Khoros does, and is very stable.
- Perl has excellent support for text processing and regular expressions, which makes it well-suited for management tasks.

The following text is borrowed from the man page for Perl4:

> *Perl is an interpreted language optimized for scanning arbitrary text files, extracting information from those text files, and printing reports based on that information. It's also a good language for many system management tasks. The language is intended to be practical (easy to use, efficient, complete) rather than beautiful (tiny, elegant, minimal). It combines (in the author's opinion, anyway) some of the best features of C, sed, awk, and sh, so people familiar with those languages should have little difficulty with it. (Language historians will also note some vestiges of csh, Pascal, and even BASIC-PLUS.) Expression syntax corresponds quite closely to C expression syntax. Unlike most Unix utilities, Perl does not arbitrarily limit the size of your data--if you've got the memory, Perl can slurp in your whole file as a single string. Recursion is of unlimited depth. And the hash tables used by associative arrays grow as necessary to prevent degraded performance. Perl uses sophisticated pattern matching techniques to scan large amounts of data very quickly. Although optimized for scanning text, Perl can also deal with binary data, and can make dbm files look like associative arrays (where dbm is available). Setuid Perl scripts are safer than C programs through a dataflow tracing mechanism which prevents many stupid security holes. If you have a problem that would ordinarily use sed or awk or sh, but it exceeds their capabilities or must run a little faster, and you don't want to write the silly thing in C, then Perl may be for you. There are also translators to turn your sed and awk scripts into Perl scripts.*

A.1. Where To Get Perl

You can get Perl 4.036 from the following anonymous ftp archives.

Where to get Perl 4.036 via anonymous ftp

Archive Name	IP Address	Path
`ftp.khoros.unm.edu`	`198.59.155.28`	`/pub/perl/perl.4.0.36.tar.gz`

† who also created *rn*, and *patch*.

Where to get Perl 4.036 via anonymous ftp		
Archive Name	**IP Address**	**Path**
ftp.uu.net	192.48.96.9	/languages/perl/perl.tar.gz
ftp.cs.ruu.nl	131.211.80.17	/pub/PERL/perl.4.0.36.tar.gz
src.doc.ic.ac.uk	146.169.43.1	/packages/perl/perl-4.036.tar.gz

A.2. Books and Other Sources of Information

If you are new to Perl, then the best book to start with is:

> "*Learning Perl*", by Randal L. Schwartz (published by O'Reilly & Associates, ISBN 1-56592-042-2). Often referred to as the 'llama book', since there is an engraving of a llama on the front cover.

If you start using Perl seriously, then you should get the following book:

> "*Programming Perl*", by Larry Wall and Randal L. Schwartz (published by O'Reilly & Associates, ISBN 0-937175-61-1). Often referred to as the 'camel book', since there is an engraving of a camel on the front cover.

The second book comes with Johan Vromans' quick reference guide, which is also available via anonymous ftp from:

Where to get the Perl 4.036 Quick Reference Guide via anonymous ftp		
Archive Name	**IP Address**	**Path**
ftp.khoros.unm.edu	198.59.155.28	/pub/perl/perlref-4.036.1.tar.gz
ftp.uu.net	192.48.96.9	/languages/perl/perlref-4.036.1.tar.gz
ftp.cs.ruu.nl	131.211.80.17	/pub/DOC/perlref-4.036.1.tar.gz
src.doc.ic.ac.uk	146.169.43.1	/packages/perl/perlref/perlref-4.036.1.tar.gz

There is a newsgroup for perl, `comp.lang.perl`, which has a fairly high volume, but good signal-to-noise ratio.

A.3. Perl 5

Perl5 will have been released by the time you read this. Perl5 has a number of cool new featurs, including support for objects, and much improved support for extensions. A Khoros extension for Perl5 is in development, though more as a fun project at the moment. If you're interested in this, look at the following world-wide web page (eg with Chimera, Lynx or Mosaic):

```
http://www.khoros.unm.edu/staff/neilb/kperl
```

Or send email to `neilb@khoros.unm.edu`.

B. Software Objects in Perl

You can write perl software objects, of type *script* (script objects can also be written in *sh*, *csh*, or *ksh*). You can create an initial perl script object in one of two ways:

1. In craftsman, select a toolbox, bring up the *Create Object* subform, and select the object type *script*. The pane for script object creation contains toggles for selecting the scripting language.

2. The kgenobj program can be used from the command-line:

```
% kgenobj -tb <toolbox> -type script -lang perl -oname <object-name>
```

The newly created script object will contain some boiler-plate code, which works out where the support toolbox lives, and extends the list of directories searched for perl library files. This lets you include the **khoros** and **kutils** perl libraries, which are described below.

The Khoros perl libraries are fairly minimal in content at the moment, giving you the following:

- support for standard command-line argument parsing.
- getting simple toolbox information.
- spawning of remote processes.

Perl scripts can get information about toolboxes and software objects via kecho, details of which can be found on its man page. In the future there will be support for direct perl access of this information.

To find out what script objects are already in the system, use the following command:

```
% kfindobj -type script
```

Most of them will be in perl.

C. khoros.pl — The Khoros Perl Library

The Khoros perl library currently provides a standard mechanism for describing and parsing command-line arguments, and a few utility functions.

To use the khoros perl library routines you have to include the following statement in your script:

```
require 'khoros.pl';
```

C.1. &khoros'initialize()

This function is passed three arguments:

```
&khoros'initialize('SUPPORT', $whatis, @cli_args);
```

The first argument is the name of the toolbox containing this script. The second argument is a short one-line description of the object, which is used when printing the usage statement. The last argument is an array which contains details of the command-line switches for the script. The array is best formatted as a table, as illustrated with the following, from kdoc2html:

```
@cli_args =
(
 'i',           'filename',      'documentation file to convert',
 'o',           'filename',      'file to write output to',
 '[verbose]',   '',              'verbose mode -- tell you what i´m doing'
);
```

For each command-line switch, there are three entries in the array, which are, in order:

1. The name of the switch (for example the 'i' above is for a `-i` switch). Square brackets around the switch name signifies an optional switch.

2. An optional argument specifier for the switch. An empty string ('') means that the switch has no argument. Square brackets around the identifier signify an optional argument.

3. A short description of the switch, used for the command-line usage statement.

In addition to the switches specified in your array, the perl library adds in some of the standard command-line switches available with most khoros programs. For example, every perl script which uses the khoros perl routines supports the `-U` switch, which requests a usage statement. Running the command:

```
% kdoc2html -U
```

results in the following output

```
kdoc2html:  generate html from khoros manual

usage:
    kdoc2html -i filename -o filename [-verbose] [-V] [-U]

where:
    -i filename  documentation file to convert
    -o filename  file to write output to
    [-verbose]   verbose mode -- tell you what i'm doing
    [-V]         display version of kdoc2html
    [-U]         display usage for kdoc2html
```

C.2. &khoros'parse_args()

This function parses command-line arguments according to a specification previously passed to `khoros'initialize()`. This is typically invoked as follows:

```
&khoros'parse_args(@ARGV);
```

Any trailing arguments from the command are returned by the function:

```
@leftovers = &khoros'parse_args(@ARGV);
```

If the command-line was successfully parsed, then two associative arrays can be queried:

khoros'seenswitch
is used to check whether a switch was seen. For example, referring to the kdoc2html example above, to check whether the `-verbose` switch was specified, you might write:

```
&khoros'parse_args(@ARGV);
if ($khoros'seenswitch{'verbose'})
{
    # do something here ...
}
```

khoros'argval
holds arguments for those switches which take them. For kdoc2html example, there is a `-i` switch, for specifying the input file name.

```
&khoros'parse_args(@ARGV);
$input = $khoros'argval{'i'};
print "Converting file $input\n";
```

C.3. &khoros'get_tb_name()

This function first checks to see whether the script supports a `-tb` switch, and if so whether it appeared on the command-line. If it did, then the toolbox name specified on the command-line is returned. If not, then we check to see if there is an Imakefile in the current directory, and if there is, we grab the toolbox name from there. This lets perl scripts default to working on the toolbox containing the current directory:

```
$toolbox = &khoros'get_tb_name();
```

C.4. &khoros'toolbox_path()

his function takes a toolbox name and returns the path to the top of the toolbox:

```
$tbpath = $khoros'toolbox_path($toolbox);
```

D. kutils.pl — Khoros Perl Utilities

D.1. &spawn()

This function is used to spawn one or more commands onto a possibly remote host.

```
&spawn($host, @commands);
```

The first argument is a hostname, and remaining arguments are command-strings, which will be invoked on the specified host. The commands will always be run from the *bourne shell*. If the host specified is the local host, then the commands will be run from within a bourne shell. For example, to make something in a specific directory on a remote machine, you might use:

```
&spawn($host,
        "cd /full/path",
        "make",
        "mail -s done fred");
```

This command uses *rsh*, and assumes you have a .rhosts file (see `enableRemoteShell()`).

Note: this function fork()'s and exec()'s, not waiting for the process to finish. See the description for function `WaitForWorkers()`.

D.2. &WaitForWorkers()

This function blocks until all processes started via `spawn()` have finished.

D.3. &enableRemoteShell()

This is a simplistic routine which makes a soft-link from a file `.rhosts.bk` to file `.rhosts`, to enable *rsh*. It is assumed that `.rhosts.bk` contains entries for any remote machine you are likely to use.

D.4. &disableRemoteShell()

This function removes your `.rhosts`, and is usually used in conjunction with the function `enableRemoteShell()`.

D.5. &DateStamp()

This function returns a string which contains the data and time in a standardized format. It is useful if you have scripts which generate log files.

This page left intentionally blank

Table of Contents

A. Introduction 7-1
A.1. Where To Get Perl 7-1
A.2. Books and Other Sources of Information 7-2
A.3. Perl 5 7-2
B. Software Objects in Perl 7-3
C. khoros.pl — The Khoros Perl Library 7-3
C.1. &khoros'initialize() 7-4
C.2. &khoros'parse_args() 7-5
C.3. &khoros'get_tb_name() 7-5
C.4. &khoros'toolbox_path() 7-5
D. kutils.pl — Khoros Perl Utilities 7-6
D.1. &spawn() 7-6
D.2. &WaitForWorkers() 7-6
D.3. &enableRemoteShell() 7-6
D.4. &disableRemoteShell() 7-6
D.5. &DateStamp() 7-7

This page left intentionally blank