X shell

Spencer Tipping

March 16, 2014

Contents

Part I

Language reference

Chapter 1

Similarities to TCL

Every xh value is a string. This includes functions, closures, lazy expressions, scope chains, call stacks, and heaps. Asserting string equivalence makes it possible to serialize any value losslessly, including a running xh process.[1]

Although the string equivalence is available, most operations have higher-level structure. For example, the `$` operator, which performs string interpolation, interpolates values in such a way that two things are true:

1. No interpolated value will be further interpolated (idempotence).
2. The interpolated value will be read as a single list element.

For example:

```
$ def bar bif
$ def foo "hi there \$bar!"
$ def baz $foo                      # no quoting necessary here by (2)
$ echo $baz
hi there $bar!                      # $bar unevaluated by (1)
$
```

This interpolation structure can be overridden by using one of three alternative forms of `$`:

```
$ def bar bif
$ def foo "hi there \$bar!"
$ echo $!foo                        # allow re-interpolation
hi there bif!
$ count [$foo]                      # single element
1
$ count [$@foo]                     # multiple elements
```

[1] Note that things like active socket connections and external processes will be proxied, however; xh can't migrate system-native things.

```
3
$ nth [$@!foo] 2                    # multiple and re-interpolation
bif!
$
```

All string values in xh programs are lifted into reader-safe quotations. This causes any "active" characters such as `$` to be prefixed with backslashes, a transformation you can mostly undo by using `$@!`. The only thing you can't undo is bracket balancing, which if undone would wreak havoc on your programs. You can see the effect of balancing by doing something like this:

```
$ def foo "[[[["
$ def bar [$@!foo]
$ echo $bar
[\[\[\[\[]
$
```

We can't get xh to create an unbalanced list through any series of rewriting operations, since the contract is that any active list characters are either positive and balanced, or escaped.

Chapter 2

Similarities to Lisp

xh is strongly based on the Lisp family of languages, most visibly in its homoiconicity.

Part II

Bootstrap implementation

Chapter 3

Self-replication

Listing 3.1 boot/xh-header

```
#!/usr/bin/env perl
BEGIN {
print STDERR q{
NOTE: Development image

If you see this note after installing the shell, it's probably because
you're running a version that has not yet rebuilt itself (maybe you got the
wrong file from the Git repo?). You can do this, but it will be really
slow and may use a lot of memory. There are two ways to fix this:

1. Download the standard image from http://spencertipping.com/xh
2. Have this image recompile itself by running xh.recompile-in-place (this
   will take some time because it stress-tests your Perl runtime)

Note also that bootstrapping requires Perl 5.14 or later, whereas running a
compiled image just requires Perl 5.10.

};
}

BEGIN {eval(our $xh_bootstrap = q{
# xh: the X shell | https://github.com/spencertipping/xh
# Copyright (C) 2014, Spencer Tipping
# Licensed under the terms of the MIT source code license

# For the benefit of HTML viewers (long story):
# <body style='display:none'>
# <script src='http://spencertipping.com/xh/page.js'></script>
use 5.014;
```

```
30 package xh;
31 our %modules;
32 our @module_ordering;
33
34 our %compilers = (pl => sub {
35   my $package = $_[0] =~ s/\./::/gr;
36   eval "{package ::$package;\n$_[1]\n}";
37   die "error compiling module $_[0]: $@" if $@;
38 });
39
40 sub defmodule {
41   my ($name, $code, @args) = @_;
42   chomp($modules{$name} = $code);
43   push @module_ordering, $name;
44   my ($base, $extension) = split /\.(\w+$)/, $name;
45   die "undefined module extension '$extension' for $name"
46     unless exists $compilers{$extension};
47   $compilers{$extension}->($base, $code, @args);
48 }
49
50 chomp($modules{bootstrap} = $::xh_bootstrap);
51 undef $::xh_bootstrap;
```

At this point we need a way to reproduce the image. Since the bootstrap code is already stored, we can just wrap it and each defined module into an appropriate `BEGIN` block.

Listing 3.2 boot/xh-header (continued)

```
1  sub image {
2    my @pieces = "#!/usr/bin/env perl";
3    push @pieces, "BEGIN {eval(our \$xh_bootstrap = <<'_')}",
4                  $modules{bootstrap},
5                  '_';
6    push @pieces, "BEGIN {xh::defmodule('$_', <<'_')}",
7                  $modules{$_},
8                  '_' for @module_ordering;
9    push @pieces, "xh::main::main;\n__DATA__";
10   join "\n", @pieces;
11 }
12 })}
```