Homespring-2003 Official Language Standard
Jeff Binder

§ *1 Introduction*

§ *1.0 Slogan*

"Because programming isn't like a river, but it damn well ought to be."

§ *1.1 Motivation*

One of the problems with current programming languages is that they're too abstract. Although they frequently use metaphors to exlain their concepts to users, these metaphors do not hold up very well in the long run. Enter Homespring, or Hatchery Oblivion through Marshy Energy from Snowmelt Powers Rapids Insulated but Not Great. It is also sometimes referred to as HOtMEfSPRIbNG.

§ *1.1.1 Revolution Information*

So what we have here is a new programming paradigm: Metaphore Oriented Programming, or MOP. MOP languages are built around a unified metaphor, and stick rigorously to its real-world properties and limits. This allows languages to be created that are both high-level and simple, offering exciting new abstractions and ideas that are familiar as they are powerful. As such, Homespring disposes of outmoded concepts such as classes, sequential execution, evaluation, assignment, binding, variables, numbers, and calculations.

§ *1.1.2 Consequences of Failure to Learn*

The Homespring language is the archetype of MOP, and it shows off all aspects of this revolutionary new concept. Learn it now or be left behind! Your current favorite language stands no chance! Now it is time to learn HOtMEfSPRIbNG, your next favorite language!

§ *2 Lexical Structure*

Before we get into the wonderful new concepts that you are impatiently awaiting, we must discuss Homespring's soon-to-be highly influential lexical structure.

§ *2.1 Tokens*

Homespring has exactly one (1) types of tokens: tokens. This simplicity will be greatly appreciated, once you try it. Tokens consist of zero (0) or a number greater than zero (¿ 0) of non-whitespace characters, separated by one (1) character of whitespace.

§ *2.1.1 Escaping*

§ *2.1.1.1 Tirade*

Many inferior languages include highly complex escape sequences and quoting rules. For example, how is one expected to remember that the 'n' in
n stands for 'newline', when the inept designers who thought of this could just as well have chosen 'e', 'w', or any of the other character in that word? Homespring's system is far superior, as well as intellectually stimlating.

§ *2.1.1.2 The Metacharacter*

Homespring offers one (1) metacharacter, namely, the period ('.'). To include a newline in a token, just use a period. To include a space in a token, just put a period before it. To include a period in a token, just put a space before it. The use of tabs is discouraged, as it is not possible in HOtMEfSPRIbNG.

§ *2.1.1.2.1 Paradox*

The sequences ' . ' and '. .' are required to cause a causality paradox in all conforming implementations. As such, there are no conforming implementations.

§ *2.1.2 Example*

Although Homespring's lexical rules are so simple that you don't need an example, one is provided any way as a service to our customers. The following seqence:

Hello,. World ..

is interpreted as:

```
(Hello, )()(World.
)
```

Note the conviniently easy to add blank token.

§ *2.2 Simplicity*

That's all there is to it, except for the fact that Homespring is helpfully case insensitive, which is mentioned in this section. But there is one exception: the token END causes watershed to ignore part of the tree. That's OK, though.

§ *3 Syntax*

Homespring disposes of the outdated notion of syntax, taking the burden of program design off the shoulders of the programmer, and putting it nowhere in particular.

§ *4 Innovations*

Now we can finally get to the exciting innovations you've been waiting for!

§ *4.1 The River Paradigm*

Homespring uses the paradigm of a river to create its astoundingly user-friendly semantics. Each program is a set of rivers that flows into the watershed (the screen). Information is carried by water (a priority queue), which flows from the springs (constants) through the network of rivers (which represents a red-black binary tree), to the watershed. Information input by the users also comes from the watershed, in the form of salmon (which represent string values) which swim upstream, using their sense of smell (represented by a string compare function, possibly implemented as a hash value of their contents or a precomputed deterministic finite automaton) to find their way to their home river (a terminal node of the tree).

§ *4.1 Spawning*

Once salmon reach the terminal nodes, they spawn, creating new salmon. All of the salmon then travel back to the watershed, appearing as user output.

§ *4.2 Program structure*

§ *4.2.1 Inferiority of Other Approaches*

In a bold and dynamic move, Homespring has ony a single structure which is used by all programs. In the traditional languages which you are now free from and will never have to use again, you would waste most of your valuable time creating a structure for your program which does what you want it to do. HOtMEfSPRIbNG liberates you from this, allowing you to spend your time in a few mega-productive fits of work, and the get back to slacking off. You see, with Homespring you simply use the language's built-in structure, and come up with a way to force it to do what you want. The superiority of the approach is so obvious that it need not be mentioned.

§ *4.2.2 The Ideal Approach*

The tokens of a Homespring program are automatically formed into the ideal program struture, a network of rivers. To simplify things only 'nodes', points where two rivers come together, are considered. The tokens are therefore interpreted as a tree, with the first token as the root, and the rest added one branch at a time. Blank tokens are used to jump up in the tree. So by these simple rules, the program

```
a b c  d e   f g    h i
```

Is obviously parsed into this tree:

```
'a'
 'b'
  'c'
  'd'
   'e'
  'f'
   'g'
 'h'
  'i'
```

Remember that the outmoded concept of indentation is not present in Homespring, since two spaces does not have the same meaning as one space. This allows you to avoid worrying about program style and focus on what programmng is really about, the reproductive behavior of salmon.

A program with no tokens obviously can't be treated normally. Such a program will, as expected, print the message:

```
In Homespring, the null program is not a quine.
```

and exit.

§ *4.3 Superior Simplicity*

That's the basic structure of Homespring. One final advantage that must be mention in this standard, so that it can serve as a full specification of the language, is that it is very easy for an implementation of HOtMEfSPRIbNG to provide perfect errors, because, basically, every string of characters is a valid Homespring program.

§ *5 Reference*

The concepts of Homespring are so easy to handle, all that is needed is a feature-by-feature reference and a few examples of real-world programs, such as simple implmentation of the important GNU hello program, which prints 'Hello, World!'.

§ *5.1 Reference*

This is a reference of all the features that can be located on rivers. Each token represents a different feature. Here are all of the features, presented in the order of their inception.

§ *5.1.1 None of the below*

Tokens that are not equal to any of the features described below are 'constants', or springs. Springs can occur at junctions and at the beginnings of tributaries. Springs are the homes of salmon. Salmon always swim to their home springs, springs with the same text as the salmon. If the Salmon has no home spring or cannot reach its home spring, it takes the path closest to the top of the program.

Once a salmon reaches a spring, a new, young, identical salmon is created, and both swim back out to the watershed. The newly created salmon carries the text of the spring, and is first in the list of fish. The list switches order everytime the fish move. When looking for a spring, salmon will only swim upstream. Salmon become mature when they spawn.

§ *5.2.2 hatchery*

Hatcheries create mature salmon with the text 'homeless', which immidiately swim upstream along he leftmost path until they reach a spring. They only operate when supplied with electricity.

§ *5.2.3 hydro. power*

Creates electricity as long as water is flowing through it. If it is hit by a snowmelt, it is destroyed. Electricity is supplied to everything downstream.

§ *5.2.4 snowmelt*

Sends out a powerful flow that can destroy some functions once it hits them. Snowmelts are processed first, then water, then electricity, then salmon. Then it starts back with snowelts.

§ *5.2.5 shallows*

Mature salmon take two turns to pass through.

§ *5.2.6 rapids*

The inverse of shallows.

§ *5.2.7 append down*

Appends all salmon from the bottom branch to the end of each salmon coming from the top. Salmon from the bottom do not pass through.

§ *5.2.8 bear*

Eats mature salmon.

§ *5.2.9 force. field*

Blocks all water, icemelts, and, well, everything when electricity is supplied. Does not kill salmon, just keeps them from passing through.

§ *5.2.10 sense*

Blocks the flow of electricity when mature fish are present.

§ *5.2.11 clone*

Creates a copy of all fish that pass through, and sends them downstream. The copies are young.

§ *5.2.12 young. bear*

Eats every other mature fish. Young fish are moved to the beginning of the list, because they don't have to take the time to evade the bear.

§ *5.2.13 bird*

Eats young salmon.

§ *5.2.14 upstream. killing. device*

Kills everything in the node upstream towards the bottom when electrified.

§ *5.2.15 waterfall*

Blocks upstream salmon.

§ *5.2.16 universe*

Everything that exists. Can be destroyed by a snowmelt.

§ *5.2.17 powers*

Always generates power.

§ *5.2.18 marshy*

Slows down snowmelts like rapids do for young salmon.

§ *5.2.19 insulated*

Blocks power.

§ *5.2.20 upstream sense*

Like sense, but only works for upstream fish.

§ *5.2.21 downstream sense*

Like sense, but only works for downstream fish.

§ *5.2.22 evaporates*

Blocks water and snowment when powered.

§ *5.2.23 youth fountain*

Makes all fish young.

§ *5.2.24 oblivion*

Changes fish to null fish when powered. Destroyable by snowmelt.

§ *5.2.25 pump*

Fish can only enter this node when it is powered.

§ *5.2.26 range sense*

Blocks electricity when any fish is here or upstream.

§ *5.2.27 fear*

Doesn't allow fish to enter when powered.

§ *5.2.28 reverse up*

Send fish coming down from the down direction up the up direction.

§ *5.2.29 reverse down*

Send fish coming down from the up direction up the down direction.

§ *5.2.30 time*

Opposite of youth fountain.

§ *5.2.31 lock*

Keeps downstream salmon from entering when powered.

§ *5.2.32 inverse lock*

Keeps downstream salmon from entering when not powered.

§ *5.2.33 young sense*

Sense but for young salmon.

§ *5.2.34 switch*

Requires mature salmon to let electricity through.

§ *5.2.35 young switch*

Requires young salmon to let electricity through.

§ *5.2.36 narrows*

Only one salmon can be present.

§ *5.2.37 append up*

Appends all salmon from the bottom to the end of salmon coming from the top. Salmon from the bottom do not pass through.

§ *5.2.38 young range sense*

Range sense but for young salmon.

§ *5.2.39 net*

Only young salmon can enter.

§ *5.2.40 force down*

Like reverse down, except upstream salmon can't go down.

§ *5.2.41 force up*

Like reverse up, except upstream salmon can't go up.

§ *5.2.42 spawn*

Makes all fish upstream spawn when supplied with electricity.

§ *5.2.43 power invert*

Supplies electricity when no electricity is supplied. Can be smashed.

§ *5.2.44 current*

Only mature salmon can enter.

§ *5.2.45 bridge*

Becomes blockage once destroyed by snowmelt.

§ *5.2.46 split*

Splits a salmon into its individual character salmon.

§ *5.2.47 range switch*

Like a switch, but also like a range sense.

§ *5.2.48 young range switch*

Like a range switch, but for young salmon.

§ *5.2.49 (null)*

Nothing. These tokens can only be created when spaces can not move the current position up the tree any more. They are also autmatically created to balance the tree. Salmon will not enter these when swimming upstream unless they have a specific destination.

§ *6 Examples*

The first example program is the simplest useful Homespring program:

This program is similar to the cat utility, but it doesn't print the newlines like cat irrationally does. Here is a version of the inferior old cat utility:

```
.
```

Here are several possible implementations of the important and useful UNIX utility 'hello'. This is the simplest possible one:

```
Universe␣bear␣hatchery␣Hello.␣World!.
␣Powers␣␣␣marshy␣marshy␣snowmelt
```

This is the same program written in professional style, with a more cohesive sentence structure:

```
Universe␣of␣bear␣hatchery␣says␣Hello.␣World!.
␣It␣␣␣powers␣␣␣␣␣the␣marshy␣things;
the␣power␣of␣the␣snowmelt␣overrides.
```

Here's the alternative, more complicated and less efficient perferred method:

```
Universe␣of␣marshy␣force.␣Field␣sense
shallows␣the␣hatchery␣saying␣Hello,.␣World!.
␣Hydro.␣Power␣spring␣␣sometimes;␣snowmelt
␣␣␣␣␣␣powers␣␣␣snowmelt␣always.
```

This is the somewhat less common but still often useful, "Hi. What's your name? Hi, xxx!" program.

```
Universe␣marshy␣now.␣The␣marshy␣stuff␣evaporates␣downstream.␣Sense␣rapids
upstream.␣Killing.␣Device␣downstream.␣Sense␣shallows␣and␣say␣Hi,.
␣␣␣That␣powers␣the␣␣␣␣␣force.␣Field␣sense␣shallows␣hatchery␣power.
Hi␣..␣What's.␣your.␣name?.
␣␣Hydro.␣Power␣spring␣␣when␣snowmelt␣then␣␣␣␣␣␣␣powers
␣␣␣␣insulated␣bear␣hatchery␣!.
␣Powers␣felt;␣␣␣␣␣␣␣powers␣feel␣␣␣␣␣snowmelt␣themselves.
```

This program tests whether the user knows what six times four is, and get this: the *program* knows what six times four is!

```
Universe␣alive␣with␣youth.␣Fountain␣bear␣Marshy
evaporates␣downstream.␣Sense␣rapids
upstream.␣Killing.␣Device␣downstream.␣Sense␣shallows␣you.␣lie!.
␣Powers␣␣␣force.␣Field␣sense␣shallows␣the␣hatchery␣but
what's.␣six.␣times.␣four?.
␣␣Hydro.␣Power␣spring␣␣with␣snowmelt␣which␣has
␣␣␣␣␣␣␣powers␣enough.
␣␣␣␣␣␣␣␣It␣powers␣␣␣␣snowmelt␣at␣least.
␣␣␣␣␣␣␣Marshy␣lock␣upstream.␣Sense␣bear␣now.
24␣␣powers␣drive␣␣␣snowmelt␣away.
␣␣␣Insulated␣bear␣hatchery␣time,␣rightyo!.
␣HYDRO.␣Power␣spring␣␣with␣snowmelt␣first.
```

This extremely powerful program can actually add two arbitrary digits together, in only twenty seconds or so on a fast machine!:

```
Universe␣is␣marshy␣but␣evaporates␣downstream.␣Sense␣the␣rapids␣reverse.␣Down
bridge␣is␣now␣marsh:
Marshy␣marshy␣marshy␣marshy␣marshy␣marshy␣marshy␣marshy␣marshy␣marshy␣now.
All␣evaporates␣downstream.␣Sense
the␣rapids␣now:
Rapids␣rapids␣rapids␣rapids␣rapids␣rapids␣rapids␣rapids␣sensed.
Ugh␣+.
␣Take␣powers␣from␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣snowmelt␣␣therefore;
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣the␣␣␣current␣time␣is␣of␣youth.␣Fountain␣is␣young.␣Bear␣cannot
reverse.␣Down␣inverse.␣Lock␣young.␣Switch␣young.␣Range.␣Switch␣clone␣to␣the
switch␣itself.␣Now␣inverse.␣Lock␣narrows␣down:
␣␣␣␣Powers
␣␣␣␣␣␣␣to␣␣␣append.␣Up␣go␣all␣young.␣Bear␣time␣evaporates
then.␣Therefore:
Spawn␣power.␣Invert␣evaporates␣it.␣Down␣force.␣Down␣reverse.␣Down␣net.␣The
␣net␣reverses␣force.
Now␣try:
Add␣add␣add␣add␣add␣add␣add␣now.
It␣is␣not␣possible;␣now␣count:
0.
1.
2.
3.
4.
5.
6.
7.
8.
9.
10.
11.
12.
13.
14.
15.
16.
17.
18+.
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣You␣␣␣can␣␣␣now␣␣␣pump
in␣reverse.␣Down␣lock␣goes;␣narrows␣lock␣down:
Inverse.␣Lock␣young.␣Range.␣Sense␣0n␣1n␣2n␣3n␣4n␣5n␣6n␣7n␣8n␣9n
␣␣␣␣␣␣␣␣␣␣Powers␣␣␣␣␣␣␣␣␣lock␣time␣now.
Inverse.␣Lock␣young.␣Range.␣Sense␣0n␣1n␣2n␣3n␣4n␣5n␣6n␣7n␣8n␣9n
␣␣␣␣␣␣␣␣␣␣Powers␣␣␣␣␣␣␣␣␣␣␣␣snowmelt␣␣␣now.
␣␣␣␣Powers
␣␣␣␣␣␣all:
Bear␣hatchery␣n
␣powers
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣insulated␣bear␣hatchery␣?.
␣Hydro.␣Power␣spring␣as
␣snowmelt␣␣␣␣␣␣␣␣␣powers␣␣␣snowmelt␣␣then,␣and␣disengage.
HYDRO!!
```

This program is the language's name. It prints a bunch of various stuff:

```
Hatchery
Oblivion␣through
Marshy
Energy␣from
Snowmelt
Powers
Rapids
Insulated␣but
Not
Great
```

You can see that Homespring programs have a very poetic and expressive quality. Although it is said that artists must suffer for their work, this does not apply to HOtMEfSPRIbNG as suffering is not incuded among its features. Writing programs in any one of the flawed 'other' languages is a painful and disturbing ordeal that is best avoided at all costs.