# Our Editor: TextScape

Leland Miller
Michael Kell

## Abstraction: The "Law and Order" Editor Principle

- Who?
- What?
- When?
- Where?
- Why?

## Goals

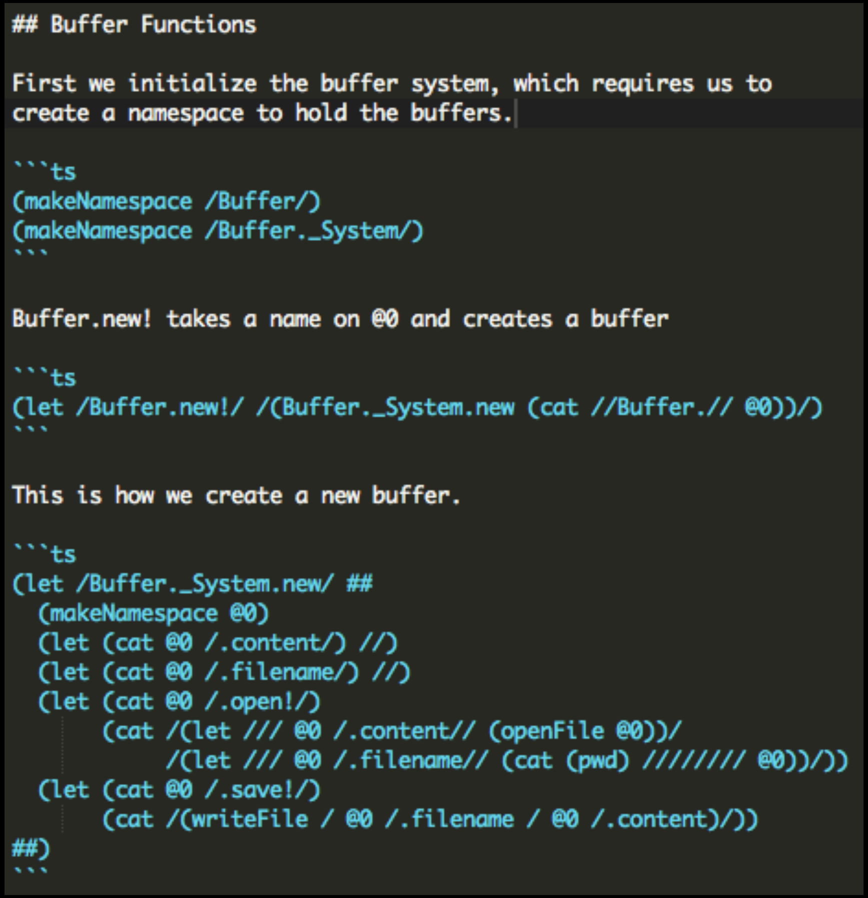
- Write an interpreter that implements our DSL
- Build a basic text editor that uses our interpreter
- Writing our own editor as an exercise to help us better understand their complexity
- Writing our own interpreter in Haskell to better understand functional programming

## Structure

- Haskell : interpreter
- Ruby : server
- JavaScript : editor

# Our Language: TextScape

- Similar syntax to LISP
- Source code embedded in Markdown files
- Everything is a namespace, or a variable
  - All variables are Strings
  - User functions are strings evaluated at runtime
  - We also used internal functions and lists



~~~
## Buffer Functions

First we initialize the buffer system, which requires us to
create a namespace to hold the buffers.

```ts
(makeNamespace /Buffer/)
(makeNamespace /Buffer._System/)
```

Buffer.new! takes a name on @0 and creates a buffer

```ts
(let /Buffer.new!/ /(Buffer._System.new (cat //Buffer.// @0))/)
```

This is how we create a new buffer.

```ts
(let /Buffer._System.new/ ##
  (makeNamespace @0)
  (let (cat @0 /.content/) //)
  (let (cat @0 /.filename/) //)
  (let (cat @0 /.open!/)
       (cat /(let /// @0 /.content// (openFile @0))/
            /(let /// @0 /.filename// (cat (pwd) //////// @0))/))
  (let (cat @0 /.save!/)
       (cat /(writeFile / @0 /.filename / @0 /.content)/))
##)
```
~~~

## TextScape: Example Syntax

- This defines a function named “sayHello!”
- “let” and “cat” are symbols
- Statements are in parentheses
  - Parentheses delimit lists where the first element is function, and all other elements are arguments
- “/” and “#” delimit literals
- “@0” will be replaced with the first unnamed argument

```
(let /sayHello!/ #
     (cat /Hello, / @0 /!/)
#)


>>(sayHello! /Leland/)
Hello, Leland!
```

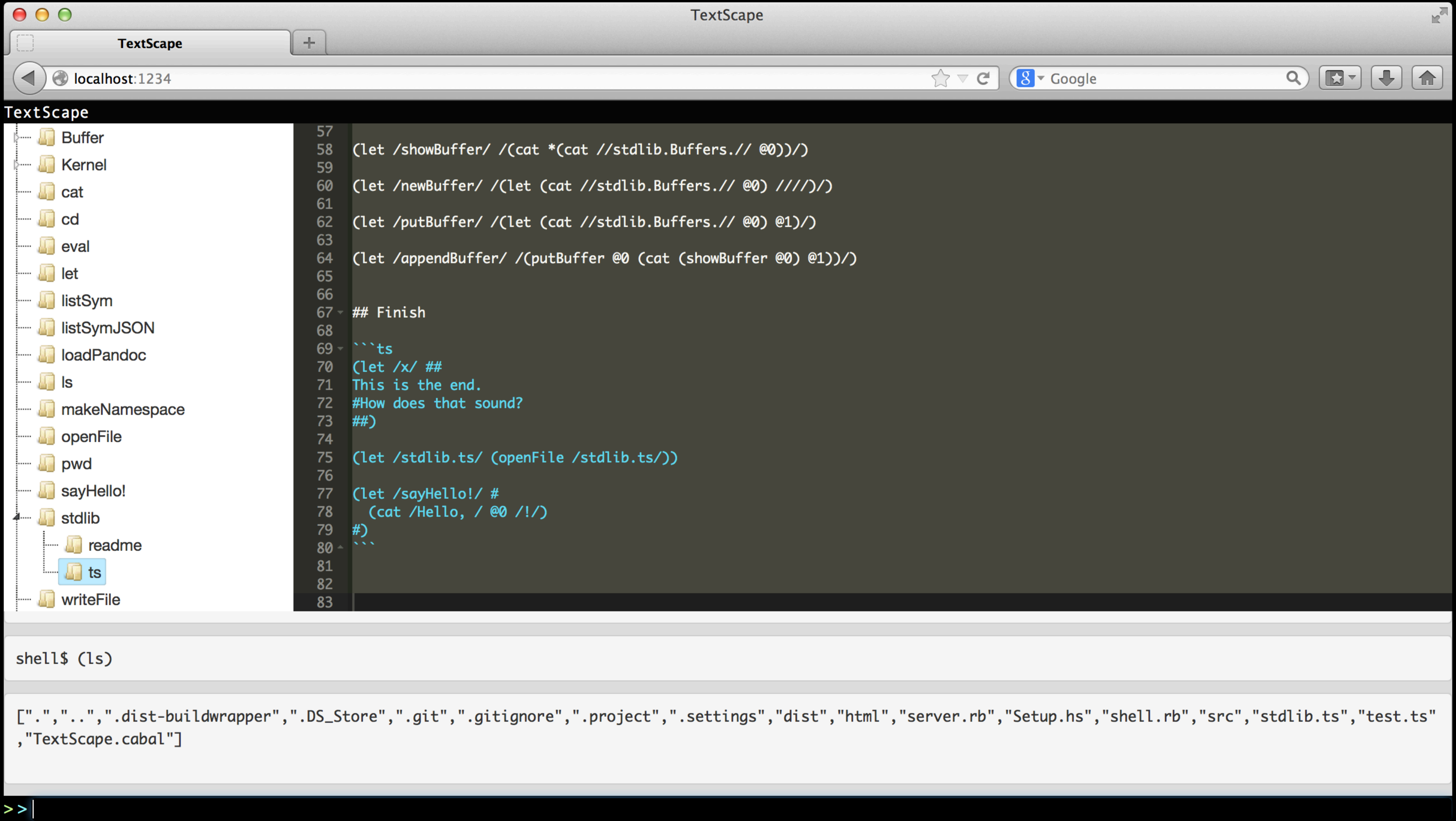
## Interpreter

- Written in Haskell
- Provides REPL environment
- Implements TextScape language

## Server

- Simple Ruby wrapper script
- Creates a server using WEBrick library
  - Allows for an interface for the interpreter
  - Serves a static file directory

# Editor



TextScape

~~~
(let /showBuffer/ /(cat *(cat //stdlib.Buffers.// @0))/)

(let /newBuffer/ /(let (cat //stdlib.Buffers.// @0) ////)/)

(let /putBuffer/ /(let (cat //stdlib.Buffers.// @0) @1)/)

(let /appendBuffer/ /(putBuffer @0 (cat (showBuffer @0) @1))/)


## Finish

```ts
(let /x/ ##
This is the end.
#How does that sound?
##)

(let /stdlib.ts/ (openFile /stdlib.ts/))

(let /sayHello!/ #
  (cat /Hello, / @0 /!/)
#)
```
~~~

```
shell$ (ls)
```

```
[".","..",".dist-buildwrapper",".DS_Store",".git",".gitignore",".project",".settings","dist","html","server.rb","Setup.hs","shell.rb","src","stdlib.ts","test.ts"
,"TextScape.cabal"]
```

```
>>
```

# What We Learned

- Editors are incredibly complex
- Most of our time was spent on the interpreter, not the editor
- No matter the implementation, every editor has its pluses and minuses
- The creation of any editor forces creators to always make tradeoffs

Fin