Seed Tutorial

1 Prerequisites

- Go `http://golang.org`
- Gnuplot
- Seed `https://github.com/nathankerr/seed`
- A web browser (tested in Safari, Chrome, and Firefox)
- Some knowledge of command lines, go, javascript, html, and services.

2 Getting Started

Make a new directory to work in:

```
mkdir seed-tutorial; cd seed-tutorial
```

Create a file, kvs.seed, with the following contents (don't include the line numbers).

```
1 table kvstate [key] => [value]
```

This file defines the core of a key-value store service. Line 1 describes a table called kvstate with a single key column called key and a single value column called value.

Start the service by running:

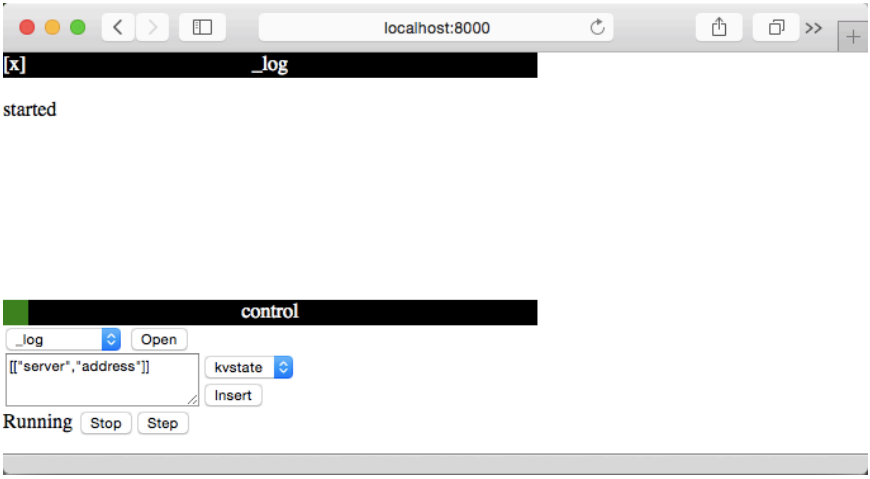
```
seed -sleep 0.5s -monitor :8000 -execute kvs.seed
```

-sleep 0.5s will sleep half a second in between time steps. This slows things down enough so that we can see what is happening in the service instance. A sleep value is needed when using the monitor to limit the data the web browser must process.

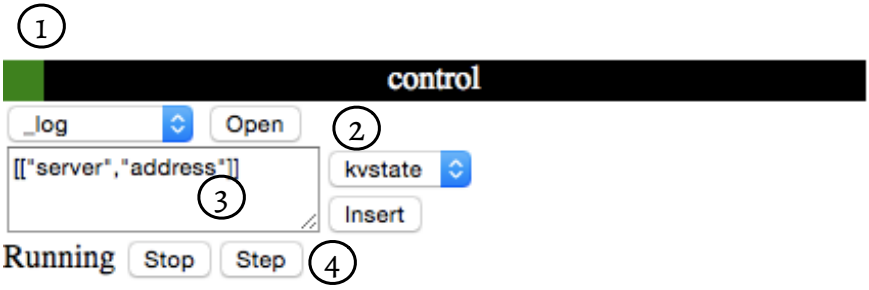
-monitor :8000 starts the debugging monitor on port 8000. To restrict access to localhost, use -monitor localhost:8000.

-execute starts the service using the built in interpreter.

The monitor can be accessed at `http://localhost:8000` using your favorite web browser:



The debug monitor presents a view of the state of the service instance. The control area has controls for controlling the monitor.



1. Green shows the monitor is connected; red disconnected.

2. Access to available views. Select the desired view and press open.

3. Insert data into a collection. Data is encoded in JSON as an array of arrays. "server" and "address" are special values that are translated into the server and client addresses respectfully.

4. Execution control. Start and stop the service, and step through the parts of the time step.

The rest of the monitor area is for the views. Click on the name of the view to put it in the main area (on the left, where _log is); [x] to close the view. Views beginning with underscores are special.

_log The monitor log

_service The service that is running (expressed in Seed)

_graph The flow graph of the running service. Rule numbers in the graph correspond to those in _service.

_time The time spent on the last timestep

Views not beginning with underscores correspond to collections (e.g., tables, inputs, outputs) in the service.

3 KVS Put Operation

```
1 table kvstate [key] => [value]
2
3 input kvput [key] => [value]
4 kvstate <+- [kvput.key, kvput.value]
```

```
seed -sleep 0.5s -monitor :8000 -execute kvs.seed
```

Insert stuff into kvput and watch how it flows to kvstate.

4 Put Client

The server is the same as the previous section:

```
1 table kvstate [key] => [value]
2
3 input kvput [key] => [value]
4 kvstate <+- [kvput.key, kvput.value]
```

When starting the service, add the network interface by using the networkg transformation:

```
seed -sleep 0.5s -monitor :8000 -execute
   -transformations network kvs.seed
```

The client is a static web page (index.html):

```
<!DOCTYPE html>
<html>
<head>
<title>Put Client</title>
<meta charset="utf-8" />
<script>
  var websocket, address, connected, server
  var key, value

  function onClose() {
    connected.style.backgroundColor = "red"
  }

  function send() {
    try {
      websocket.send(JSON.stringify({
        Operation: "<~",
        Collection: "kvput",
        Data: [[server, key.value, value.value]]
      }))
    } catch (e) {
      alert(e)
    }
  }

  function init() {
    connected = document.getElementById("connected")

    key = document.getElementById("key")
    value = document.getElementById("value")

    // connect to the monitor server
    server = "ws://" + window.location.hostname +
       ":3000/wsjson"
    websocket = new WebSocket(server);
    websocket.onclose = onClose;

    // set uniq id (address)
    websocket.onopen = function() {
      address = Math.random().toString(36).substr(2)
      try {
        websocket.send(JSON.stringify(address))
      } catch (e) {
```

```
43            alert(e)
44          }
45
46          connected.style.backgroundColor = "green"
47        }
48      }
49
50      window.addEventListener("load", init, false)
51    </script>
52    </head>
53    <body>
54    <div id="connected" style="width:20px; height:20px;
          background-color:red"> </div>
55    <div>
56      <form onsubmit="return false;">
57        <input type="text" id="key" value="Key" />
58        <input type="text" id="value" value="Value" />
59        <input type="submit" value="Put"
            onclick="send()"/>
60      </form>
61    </div>
62    </body>
63    </html>
```

Because web browsers don't like allowing websocket access from pages loaded from the local file system, we need a simple web server:

```
1   package main
2
3   import (
4     "fmt"
5     "net/http"
6   )
7
8   const ADDR = ":4000"
9
10  func main() {
11    http.Handle("/", http.FileServer(http.Dir(".")))
12
13    fmt.Println("Starting Put client on", ADDR)
14    err := http.ListenAndServe(ADDR, nil)
15    if err != nil {
16      println(err.Error())
17    }
```

```
18 | }
```

Start the client web server by:

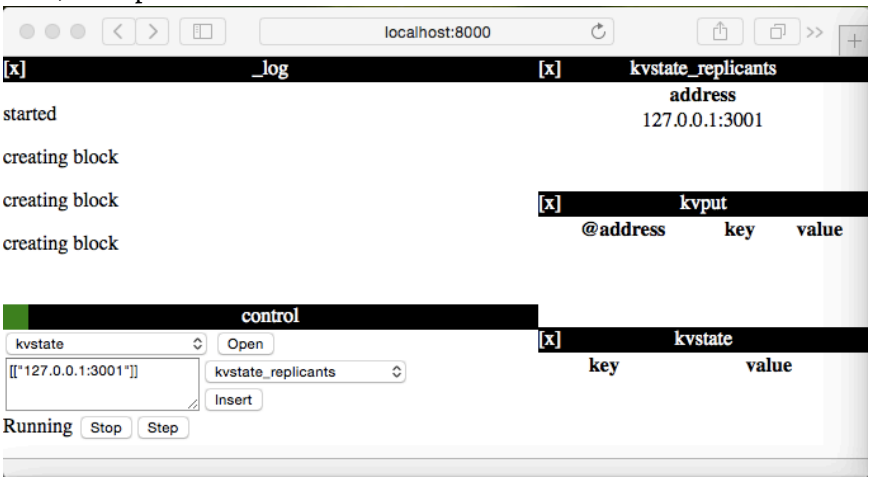
```
go run server.go
```

5 Replication

Start the first replica:

```
seed -sleep 0.5s -monitor :8000 -execute
    -transformations "network replicate" -address
    :3000 kvs.seed
```

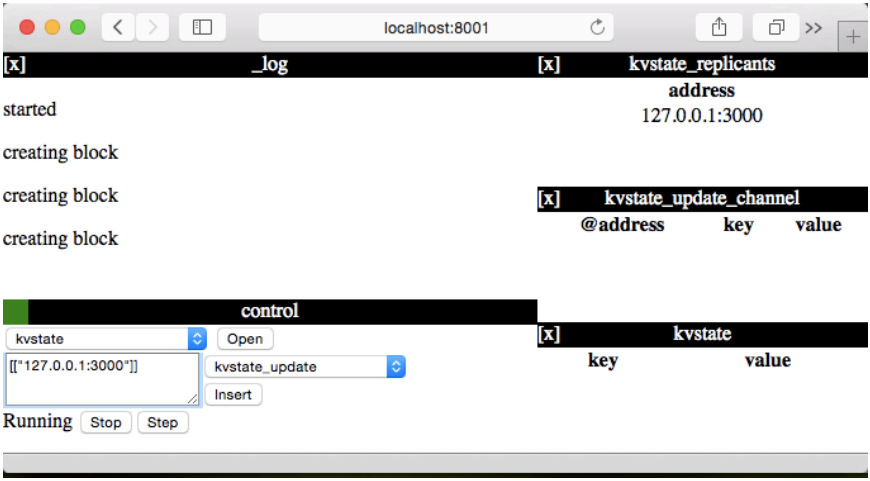
Insert [["127.0.0.1:3001"]] into :8000's kvstate_replicants
Also, set up the views as follows:



Start the second replica:

```
seed -sleep 0.5s -monitor :8001 -execute
    -transformations "network replicate" -address
    :3001 kvs.seed
```

Insert [["127.0.0.1:3000"]] into :8001's kvstate_replicants
Also, set up the views as follows:



Start a client (will connect to :3000):

```
go run server.go
```

Put some value into the client and watch as it first hits the first replia and then the second.

Note that due to the transformations, no changes were needed to kvs.seed or the client to enable replication. However, while the client will only connect to one of the replicas, this could be resolved by using a load balancer.