climate

Release 0.2.0

May 11, 2014

CHAPTER 1

Prompt Chains

Climate is designed around a chainable interface that allows you to design complicated interactions. The concept of a prompt chain, however, does not refer to the *chainability* of the functions though.

A prompt chain is way of programmatically defining a question / response sequence.

To be completed

1.1 Forking Prompts

To be completed

CHAPTER 2

REPL Interface

Node's repl is great and offers a lot of great functionality, that said it does have some limitations. Depending on the type of application that you are writing, you may find the REPL implemented in climate a better fit.

2.1 Creating a REPL

Creating a REPL with climate is really simple. The example below show's a trivial example:

```
var climate = require('climate');

climate
  .repl('say hi>')
  .command('hi', function(input) {
    var message;

    if (! input) {
      message = 'Hello, my name is Bob';
    }
    else if (input.toLowerCase() == 'bob') {
      message = 'You remembered my name, awesome!';
    }
      else {
        message = 'My name isn\'t ' + input + ', it\'s Bob';
      }

    climate.out(message + '\n');
  });
```

This example simply displays a prompt `say hi` which responds with varying results when you enter "hi", "hi Bob" or "hi something else". Currently the REPL is case sensitive with commands so "HI Bob" will not work.

2.2 Prompting for Data within a REPL

Within Climate it's possible to create sub-instances which divert from the current prompt chain. This is particularly useful when using REPL, as a REPL is essentially a non-incrementing *prompt chain*.

Consider the following example:

```
var climate = require('..');

climate
  .repl('say hi>')
  .command('hi', function(input) {
    var message;

    if (! input) {
      message = 'Hello, my name is Bob';
      climate.fork()
      .prompt('What\'s your name?')
      .receive('*', function(name) {
        climate.out('Hey there ' + name + '!!\n');
      });
    }
    else if (input.toLowerCase() == 'bob') {
      message = 'You remembered my name, awesome!';
    }
      else {
        message = 'My name isn\'t ' + input + ', it\'s Bob';
      }

    climate.out(message + '\n');
  });
```

When the repl receives the text `hi`, it prompts for additional information. It does this by *forking* a new prompt. This fork creates a new *prompt chain* and pauses the currently executing chain. Once the new fork has been completed / resolved, the previously active chain is resumed and interaction continues with that chain.

2.3 Loading Commands from Command Files

While it's simple enough to wire up a few commands as shown in the previous examples, when it comes to writing more complicated command line applications with a wide variety of commands then it definitely better to provide a little more structure to your application.

For this we can use the `loadActions` method of the repl:

```
var climate = require('..');
var path = require('path');

climate
  .repl('testrepl>')
  .loadActions(path.resolve(__dirname, 'actions'));
```

You can see here that no actual command logic in the file above, but rather it is implemented in separate command (or action) files stored in the referenced actions directory. An example of one of those files is shown below:

```
module.exports = function(input) {
    console.log('And now I\'m sliding... wheeee :)');
};
```

Using this technique provides some structure to your application which will generally make it easier for people to contribute to and extend your work.

CHAPTER 3

Input Filters

To be completed

CHAPTER 4

Indices and tables

- *genindex*
- *modindex*
- *search*