River Language Documentation

Release 0.1.0

Michael Wilson

January 31, 2015

Contents

1 Introduction **3**
1.1 Philosophy . . . 3
1.2 Snapshot or Overview . . . 3
1.3 Goals . . . 3

2 Basic Concepts **5**
2.1 Streaming . . . 5

3 Language Reference **7**
3.1 Syntax . . . 7
3.2 Data Model and Types . . . 7
3.3 Streams . . . 9

4 River Framework **11**
4.1 Applications . . . 11
4.2 Tasks . . . 11
4.3 Entry Points . . . 11

5 General Notes and Todos **13**

6 Indices and tables **15**

Contents:

CHAPTER 1

Introduction

This is an introduction

1.1 Philosophy

This document discusses the philosophy of River.

- Favor readable over short
- Use keywords in sentence-like syntax
- use visual clues and symbols to describe actions and objects
- Everything is an object
- Everything is reflectable and extensable
- Self-extendable language
- Everything is streamable
- Use decorators and meta to **describe** and **direct** data, not modify
- Should be one obvious, best way to do something
- Simple is better than complex
- Complex is better than complicated
- Visual wherever possible
- "Walk" data and streams

1.2 Snapshot or Overview

A basic overview in 5 minutes

1.3 Goals

- Excels at data manipulation and mining
- Self Extensable

- Data Aware
- Graph Aware
- Streaming
- Multiple entry points (async)
- Object oriented
- Non-linear. Starts at an entry point, then follows to other tasks or entries
- Include a powerful standard library * Event loop and hook system
- Extension of python, so can use native python libraries
- Both a language and a framework

CHAPTER 2

Basic Concepts

These are basic concepts like streaming, graphs, etc

2.1 Streaming

Pipes and such

CHAPTER 3

Language Reference

A basic handbook of all the awesome

3.1 Syntax

This document discusses the syntax of River.

3.1.1 Keywords

This is a list of keywords

-> Pipe a stream through a task is Checks the state, type, and visibility of a stream grab pulls the variable from a stream clone clones a stream variable to local scope entry defines an entry point task defines a task fork split a stream start l spring start a new stream merge merge two streams destroy destroys a stream

3.2 Data Model and Types

This is for data model and data types and such

3.2.1 Graphs

A new Graph object ca connect to a graph datastore via an adapter and load as an OGM

Graph friends:connect

OrientDb::graph_name options = credentials

Or map an existing object or set of tupples

Graph friends:map “Michael” “friends_with” “Nicole” “Nicole” “friends_with” “Sam”

Adapters work through a **blueprint** interface to query or execute gremlin, but you can also invoke blueprints directly

Graph friends:bluebrings { # Code

}

ALL adapters implement DataAccess and GraphDbAdapter

Altogether now:

Graph friends:connect OriendDb::users

bestFriends = friends.out("best").levels(3).walk()

A Sub H

Hello, yo!

3.2.2 Streams

This is for streams

3.2.3 Notes

- Assign a variable with the = operator.
- Assign a mutable variable without a type first `name = "michael"`
- Assign a type cast variable with a type first `Type name = "Michael"`
- Use a variable Type's creation method to create `Type name:creation_method`
- Cast a variable from one type to another. `(NewType) name = "Michael"`
- Object creation `Type name:new` or `name = new Type`

3.2.4 Data Types

These are the different data types

Primative (simple)

- Boolean
- String
- Integer
- Float

Collections

- Lists
- Tuples
- Dictionaries

Complex Data

- Graphs (working upon an adapter to connect to data persistance)
- Matrix or grid
- Document (Mongo-like)
- Tree (hierarchy)
- 3D matrix in 3D space
- Hash for security
- Times and dates

3.3 Streams

This describes streams, streaming and pipes

- Streams can be **open** or **closed**
- Streams can **expire** after a time
- Streams can be **local** (this session only) or **global** for the lifetime of the applications
- Streams can be **public** or **private** to this process

The idea is that you grab a variable from a stream (like a fish), modify it, and then send it along to the next task. You can also modify the entire stream.

You can:

- `open` / `close` streams and variables
- `clone`, `lock`, `unlock`, `alias` variables or streams
- **can check the:** *state (open, closed, public private, etc), *type (class), scope, contents (list of variables or streams), and value of streams

CHAPTER 4

River Framework

The framework and standard library for applications

4.1 Applications

Applications use wrappers of the core. For instance, http is one wrapper and the entry points would be url routes.

4.2 Tasks

Tasks modify streams and then pass them along

4.3 Entry Points

Entry points are regular tasks that are bound to the wrapper (http, cli, etc)

CHAPTER 5

General Notes and Todos

These are scratch notes. Where do they fit?

- Utility map to visually display application flow
- PSR-4 style autoloading
- use pip for package management?

CHAPTER 6

Indices and tables

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