
Micro Tiling Documentation

Aurélien Moisson-franckhauser, Johyn Papin, Quentin Boyer

Mar 21, 2019

Contents:

1	Brainfuck Interpreter	1
2	Database	3
3	Ensicoín Library	5
4	Golfer: a Gopher Server	7
5	Micro Tiling Client	9
5.1	Library usage	9
5.2	Command line usage	9
6	a_pi	11
6.1	Segment generation	11
6.2	Flask Application	11
7	Milllllll	13
7.1	Module for Rotating points	13
7.2	Milllllll Reference	13
8	Unitator	15
8.1	Unitator Service	15
8.2	Clipping Reference	15
9	Translator	17
9.1	Translator Reference	17
9.2	Translator Service	17
10	Cruxingator	19
10.1	SMTP Server	19
10.2	Segment Splitting	19
11	Solidator	21
11.1	Master	21
11.2	Subprograms	22
12	REST API Usage	23
12.1		23

13 Indices and tables	27
Python Module Index	29

CHAPTER 1

Brainfuck Interpreter

```
class bf_interpreter.Interpret (file_name, mem_size=65536, get_input=<function Inter-  
                                pret.<lambda>>>)  
    Interpret a brainfuck code, Loads file_name in a memory of size mem_size, loads the inputs with get_input  
  
    get_output (size=0)  
        If size is 0 gets the current output, else waits for the output size to be size  
  
    step ()  
        Execute an instruction
```


CHAPTER 2

Database

`database.open_db()`

Opens the database

`database.update_state(db, new_state, job_id)`

Update the state in *db* for the job *job_id* if it is less than *new_state*

CHAPTER 3

Ensicoins Library

Wrapper around ensicoinscli

`ensicoins.generate_keys()`

Generates a pair of (**private_key**, **public_key**)

`ensicoins.send_to` (*value*, *outpoint_hash*, *outpoint_index*, *privkey_from*, *spent_output_value*, *pubkey_to*,
flags, *uid*)

Sends a transaction

`ensicoins.wait_for_flag` (*flag*)

Blocks until a transaction is issued containing *flag*

`ensicoins.wait_for_pubkey` (*pubKey*)

Blocks until a transaction is issued by *pubKey*

CHAPTER 4

Golfer: a Gopher Server

`gopher.golfer.get_entries()`
returns `[(DirEntry)]` as defined by the gopher protocol

`gopher.golfer.get_selectors()`
returns a list of selectors for the known files

`gopher.golfer.main()`
Starts the gopher server

5.1 Library usage

Module to generate mosaics using micro-tiling

`client_library.client.generate_mosaic(host, on_invoice)`
Wraps **manage_state** to return a result, uses *on_invoice* when needing to pay

`client_library.client.get_address(host, job_id)`
Gets the ensicoin address needing fees for *job*

`client_library.client.launch_job(host)`
Create a job for *host* and returns the **job_id**

`client_library.client.manage_state(host, result)`
Iterator on the completion of a job generation at *host*,
Writes the mosaic in the list *result*

5.2 Command line usage

A command line usage is defined for convenience, however it needs progressbar2.

It is used by invoking `client.py` with arguments the REST api address for micro-tiling and an ensicoin address used to pay

This is the module responsible for generating random segments using the digits of pi.

6.1 Segment generation

Generation of a bounded segment

`a_pi.segment_generator.random_segment(x_max, y_max)`

Creates a segment with each endpoint x, y such that:

$$x \in [0; x_{max}], y \in [0; y_{max}]$$

6.2 Flask Application

Flask `a_pi` executing an action on each digit of pi sent

`a_pi.app.action(db, job_id, job_current)`

Advances the state of the job `job_id` using the state `job_current` as initial state and executes an action, here `segment generation`

`a_pi.app.close_db(e=None)`

Closes the DB

`a_pi.app.create_app(test_config=None)`

Generates the Flask application

`a_pi.app.get_db()`

Get the db associated with the application

`a_pi.app.terminate(db, job_id, mill_stub)`

Finishes the job `job_id` and forwards it to the `mill_stub`

This service is rotating segments on a gRPC call

7.1 Module for Rotating points

`Millllllllll.rotate.decode(s)`

Takes six bytes *s* and returns a **point**

`Millllllllll.rotate.encode(vect)`

Takes a **point** *vect* and converts it to six bytes

`Millllllllll.rotate.go_through_brainfuck(file_name, point, use_python=False)`

passes *points* through the brainfuck file *file_name*

`Millllllllll.rotate.mirror_and_turn(points, use_python=False)`

points need to be tuples or lists of floats

`Millllllllll.rotate.mirror_and_turn_segments(segments, use_python=False)`

segments need to be tuples or lists of **points** (will be returned as tuples), **points** need to be tuples or lists of floats

7.2 Milllllllll Reference

gRPC server to rotate segments

class `Millllllllll.millllllllll.MillServicer`

Serves a Milllllllll gRPC server

Turn (*request, context*)

Turns segments contained in the RPC message *request*

`Millllllllll.millllllllll.segment_to_tuple(segment)`

Transforms a *segment* from a dict to a tuple

`Millllllllll.mllllllllll.serve()`

Start the Millllllll server

`Millllllllll.mllllllllll.write(job,job_id)`

Write a *job* to a gopher served directory while notifying a golfer server of the *job_id*

This services clips segments fetched from a gopher server into the unit square, then writes them to firefox

8.1 Unitator Service

`Unitator.unitator.listen()`
Start a Unitator server listening for notifications of a golfer server

`Unitator.unitator.recv_data(conn)`
Receive all data from a socket *conn*

`Unitator.unitator.send(data, host, port)`
Sends *data* using a gopher connection on *host:port*

`Unitator.unitator.unit(segments, job_id)`
Clip *segments* of the job *job_id* into a unit square and forwards them

`Unitator.unitator.write(job_id, text)`
Writes a *text* by an html form

8.2 Clipping Reference

Module clipping lines in the unit square

`Unitator.clipping.clip_right(lines)`
Clip *lines* with a vertical line to the right at $x = 1$

`Unitator.clipping.clip_unit_square(lines)`
Clip *lines* to fit in the unit square

class `Unitator.clipping.p(x, y)`
Class defining a two dimensional point in space

class `Unitator.clipping.segment` (*a*, *b*)
Class representing a segment by the two endpoints

`Unitator.clipping.turn_right` (*lines*)
Rotates *lines* by 90 degrees

This translates and replicates segments from a inotify event and sends them by mail

9.1 Translator Reference

`Translator.translator.decode(s)`

Takes six bytes *s* and returns a **point** (tuple of coordinates)

`Translator.translator.decode_nine(s)`

Decode nine **points** from *s*

`Translator.translator.encode(vect)`

Takes a *vect* and converts it to six bytes

`Translator.translator.go_through_brainfuck(file_name, point, use_python=False)`

Passes *point* through the brainfuck file *file_name* Returns a list of translated points

`Translator.translator.translate_segments(segments, use_python=False)`

Translate and copies each segment from the list *segments* in all eight directions

9.2 Translator Service

`Translator.app.create_app(test_config=None)`

Launches the flask web application

`Translator.app.encrypt(string, gpg)`

Encrypt the *string* in pgp encoded using the context *gpg*

`Translator.app.send(host, segments, job_id, gpg)`

Sends the encoded *segments* to the next service by smtp through *host* encrypted using the *gpg* context

`Translator.app.translation(segments, job_id, gpg)`

Creates replicas of the segments in the eight directions

This service cuts all segments such that no two segments cross, and publishes the result in an ensicoin transaction

10.1 SMTP Server

```
class Cruxingator.smtp_serv.Handler
    Handles SMTP requests

    handle_DATA (server, session, envelope)
        Handles the content of a mail

    handle_RCPT (server, session, envelope, address, rcpt_options)
        Refuse all mails not in @micro-tiling.tk

Cruxingator.smtp_serv.decrypt (string)
    Decrypts the string using gpg
```

10.2 Segment Splitting

```
class Cruxingator.split.Point (pos)
    Points in two dimensions

    get_pos (precision=0.0003)
        Get a rounded position at precision level

    merge (other)
        Tells two points are equivalent

class Cruxingator.split.Segment (a, b)
    Segment between two endpoints

    intersect (other)
        Calculate the intersection point between self and other
```

```
class Cruxingator.split.Vect (x,y)
    Two dimensional Vector

    norm2 ()
        Squared norm of the vector

Cruxingator.split.cut (segments)
    Cuts the segments such that no two segments cross

Cruxingator.split.find_intersections (segments)
    Find all intersections between segments

Cruxingator.split.generate_id_tuple (points)
    Generates a tuple for each point (id, x, y, [id of linked])

Cruxingator.split.pairs (l)
    Iterate on pairs of segments
```


This service works in two parts: a master program managing subprograms

11.1 Master

signification of messages to solidator : r : the point is ready. d : there is one less point of degree 1 alive. e : a point process has ended.

class Solidator.solidator.**Point** (*pos*)
A two dimensional point whose *pos* is a **Vect**

get_pos (*precision=1e-10*)
Returns a position rounded up to *precision*

merge (*other*)
Merges with another **Point**

class Solidator.solidator.**Segment** (*a, b*)
A segment of two vectors

class Solidator.solidator.**Vect** (*x, y*)
A two dimensional vector

Solidator.solidator.**create_points** (*point_list*)
Takes in a list of [(*point_id*, *pos_x*, *pos_y*, [*list of neighbours' ids*]),...] and output a list of **Point**

Solidator.solidator.**open_process** (*point*)
Opens a process representing a *point* tells it how many neighbours it should expect and its *Point id*

Solidator.solidator.**remove_deg_1** (*points, job_id, multiprocessing=True*)
Write an svg with only closed polygons displayed

11.2 Subprograms

processes are sent the IDs of their neighbours at start once they have them, they start telling others if they have a degree of 1 and they listen for neighbours signaling they have a degree of 1

signification of messages to point_processes :

e [message coming from main meaning it is now time to write] the result.

d <neighbour_id> [the neighbouring point with neighbour_id] had only one neighbour left 'alive' and is therefore now 'dead'.

a : death acknowledge means the neighbour now knows the point is 'dead'

A <neighbour_id> <neighbour_x_pos> <neighbour_y_pos> [the] corresponding neighbour is still 'alive' (sent after the neighbour received a message 'e').

D [a neighbour is 'dead' and no line should be displayed between the] two points.

`Solidator.point_process.debug` (*string*, *own_id*)
write a *string* in stderr and add the point's *own_id* for clarity

`Solidator.point_process.main` ()
Waits for processes to die around you to check if you can survive

`Solidator.point_process.read_neighbours` (*amount*)
Reads *amount* neighbours from **stdin**

`Solidator.point_process.read_position` ()
Reads a line from **stdin** and interprets it as a position

`Solidator.point_process.signal_death` (*neighbour_messagers*, *own_id*)
Tells all neighbours of *own_id* death by *neighbour_messagers*' fifo files

`Solidator.point_process.svg_coord` (*x*)
Returns the coordinate in svg space for *x*

`Solidator.point_process.svg_output` (*message*, *own_id*, *position*, *res*)
Outputs to *res* the svg line from *position* to the neighbour indicated in *message* if *own_id* is smaller than the neighbour's id (this is to avoid writing twice the same line)

`Solidator.point_process.who_am_i_at_the_end` (*is_dead*, *neighbour_messagers*, *own_id*, *position*)
Tells all neighbours by *neighbour_messagers* the vertex *own_id* and *position* if not *is_dead*, else send them D

CHAPTER 12

REST API Usage

A running API is hosted at <https://micro-tiling.tk>

12.1

POST /

Creates a new job

- **Description:**
- **Produces:** ['application/json']

Responses

200 - Successfully created job

Name	Description	Type
id		string

405 - Invalid input

GET /{jobId}/state

Gets the state of a job

- **Description:**
- **Produces:** ['application/json']

Parameters

Name	Position	Description	Type
jobId	path	The ID of the requested job	string

Responses

200 - *State of the job*

Name	Description	Type
completion		integer
state		string

400 - *Invalid ID supplied*

404 - *Unknown Job*

405 - *Invalid Method*

GET /{jobId}/address

Gets the address needing ensicoins for a job

- **Description:**
- **Produces:** ['application/json']

Parameters

Name	Position	Description	Type
jobId	path	The ID of the requested job	string

Responses

200 - *Ensicoins address*

Name	Description	Type
address		string

400 - *Invalid ID supplied*

404 - *Unknown Job or No Address needed*

405 - *Invalid Method*

GET /{jobId}/result

Get the final mosaic in a job

- **Description:**
- **Produces:** ['application/json']

Parameters

Name	Position	Description	Type
jobId	path	The ID of the requested job	string

Responses

200 - SVG file

Name	Description	Type
result		string

400 - Invalid ID supplied

404 - Unknown Job or Job not done

405 - Invalid Method

CHAPTER 13

Indices and tables

- `genindex`
- `modindex`
- `search`

a

`a_pi.app`, [11](#)
`a_pi.segment_generator`, [11](#)

b

`bf_interpreter`, [1](#)

c

`client_library.client`, [9](#)
`Cruxingator.smtp_serv`, [19](#)
`Cruxingator.split`, [19](#)

d

`database`, [3](#)

e

`ensicoin`, [5](#)

g

`gopher.golfer`, [7](#)

m

`Millllllllll.millllllllll`, [13](#)
`Millllllllll.rotate`, [13](#)

s

`Solidator.point_process`, [22](#)
`Solidator.solidator`, [21](#)

t

`Translator.app`, [17](#)
`Translator.translator`, [17](#)

u

`Unitator.clipping`, [15](#)
`Unitator.unitator`, [15](#)

A

`a_pi.app` (module), 11
`a_pi.segment_generator` (module), 11
`action()` (in module `a_pi.app`), 11

B

`bf_interpreter` (module), 1

C

`client_library.client` (module), 9
`clip_right()` (in module `Unitator.clipping`), 15
`clip_unit_square()` (in module `Unitator.clipping`), 15
`close_db()` (in module `a_pi.app`), 11
`create_app()` (in module `a_pi.app`), 11
`create_app()` (in module `Translator.app`), 17
`create_points()` (in module `Solidator.solidator`), 21
`Cruxingator.smtp_serv` (module), 19
`Cruxingator.split` (module), 19
`cut()` (in module `Cruxingator.split`), 20

D

`database` (module), 3
`debug()` (in module `Solidator.point_process`), 22
`decode()` (in module `Milllllll.rotate`), 13
`decode()` (in module `Translator.translator`), 17
`decode_nine()` (in module `Translator.translator`), 17
`decrypt()` (in module `Cruxingator.smtp_serv`), 19

E

`encode()` (in module `Milllllll.rotate`), 13
`encode()` (in module `Translator.translator`), 17
`encrypt()` (in module `Translator.app`), 17
`ensicoin` (module), 5

F

`find_intersections()` (in module `Cruxingator.split`), 20

G

`generate_id_tuple()` (in module `Cruxingator.split`), 20
`generate_keys()` (in module `ensicoin`), 5
`generate_mosaic()` (in module `client_library.client`), 9
`get_address()` (in module `client_library.client`), 9
`get_db()` (in module `a_pi.app`), 11
`get_entries()` (in module `gopher.golfer`), 7
`get_output()` (`bf_interpreter.Interpret` method), 1
`get_pos()` (`Cruxingator.split.Point` method), 19
`get_pos()` (`Solidator.solidator.Point` method), 21
`get_selectors()` (in module `gopher.golfer`), 7
`go_through_brainfuck()` (in module `Milllll-lll.rotate`), 13
`go_through_brainfuck()` (in module `Translator.translator`), 17
`gopher.golfer` (module), 7

H

`handle_DATA()` (`Cruxingator.smtp_serv.Handler` method), 19
`handle_RCPT()` (`Cruxingator.smtp_serv.Handler` method), 19
`Handler` (class in `Cruxingator.smtp_serv`), 19

I

`Interpret` (class in `bf_interpreter`), 1
`intersect()` (`Cruxingator.split.Segment` method), 19

L

`launch_job()` (in module `client_library.client`), 9
`listen()` (in module `Unitator.unitator`), 15

M

`main()` (in module `gopher.golfer`), 7
`main()` (in module `Solidator.point_process`), 22
`manage_state()` (in module `client_library.client`), 9
`merge()` (`Cruxingator.split.Point` method), 19

merge() (*Solidator.solidator.Point method*), 21
 Milllllllll.milllllllll (*module*), 13
 Milllllllll.rotate (*module*), 13
 MillServicer (*class in Milllllll.milllllll*), 13
 mirror_and_turn() (*in module Milllllll.rotate*), 13
 mirror_and_turn_segments() (*in module Milllllll.rotate*), 13

N

norm2() (*Cruxingator.split.Vect method*), 20

O

open_db() (*in module database*), 3
 open_process() (*in module Solidator.solidator*), 21

P

p (*class in Unitator.clipping*), 15
 pairs() (*in module Cruxingator.split*), 20
 Point (*class in Cruxingator.split*), 19
 Point (*class in Solidator.solidator*), 21

R

random_segment() (*in module a_pi.segment_generator*), 11
 read_neighbours() (*in module Solidator.point_process*), 22
 read_position() (*in module Solidator.point_process*), 22
 recv_data() (*in module Unitator.unitator*), 15
 remove_deg_1() (*in module Solidator.solidator*), 21

S

Segment (*class in Cruxingator.split*), 19
 Segment (*class in Solidator.solidator*), 21
 segment (*class in Unitator.clipping*), 15
 segment_to_tuple() (*in module Milllllll.milllllll*), 13
 send() (*in module Translator.app*), 17
 send() (*in module Unitator.unitator*), 15
 send_to() (*in module ensicoin*), 5
 serve() (*in module Milllllll.milllllll*), 13
 signal_death() (*in module Solidator.point_process*), 22
 Solidator.point_process (*module*), 22
 Solidator.solidator (*module*), 21
 step() (*bf_interpreter.Interpret method*), 1
 svg_coord() (*in module Solidator.point_process*), 22
 svg_output() (*in module Solidator.point_process*), 22

T

terminate() (*in module a_pi.app*), 11
 translate_segments() (*in module Translator.translator*), 17

translation() (*in module Translator.app*), 17
 Translator.app (*module*), 17
 Translator.translator (*module*), 17
 Turn() (*Milllllll.milllllll.MillServicer method*), 13
 turn_right() (*in module Unitator.clipping*), 16

U

unit() (*in module Unitator.unitator*), 15
 Unitator.clipping (*module*), 15
 Unitator.unitator (*module*), 15
 update_state() (*in module database*), 3

V

Vect (*class in Cruxingator.split*), 19
 Vect (*class in Solidator.solidator*), 21

W

wait_for_flag() (*in module ensicoin*), 5
 wait_for_pubkey() (*in module ensicoin*), 5
 who_am_i_at_the_end() (*in module Solidator.point_process*), 22
 write() (*in module Milllllll.milllllll*), 14
 write() (*in module Unitator.unitator*), 15