# 2D_FacilitatedExclusionProcess Reference Manual

This reference manual details functions, modules, and objects included in 2D_FacilitatedExclusionProcess, describing what they are and what they do. Whether you are a new user looking to understand the functionality of the code or an experienced developer seeking details on specific methods, this documentation serves as your guide.

# System Configuration

## Description

SystemConfiguration is a structure that defines the configuration of a ParticleSystem.

## Properties

### bool directional_FEP

Boolean that defines whether the system is directional or not.

- 0 → Non-directional Facilitated Exclusion Process (CLG)
- 1 → Directional Facilitated Exclusion Process (FEP)

### bool is_periodic

Boolean that defines whether the system is periodic or not.

- 0 → Non-periodic system
- 1 → Periodic system

### unsigned int horizontal_boundaries

State of the left and right boundaries. Three possible values:

- OPENED (0)
- CLOSED (1)
- RESERVOIR (2)

### unsigned int vertical_boundaries

State of the left and right boundaries. Three possible values:

- OPENED (0)
- CLOSED (1)
- RESERVOIR (2)

# Site

## Description

Site is a class that defines one site of a ParticleSystem.

## Properties

`bool inside_the_system`

Boolean that defines whether the particle is inside the system or not.

`0` → Outside the system

`1` → Inside the system

`int x`

Position of the site on the x-axis.

`int y`

Position of the site on the y-axis.

`unsigned int state`

State of the site. Three possible values:

- `EMPTY` (0)
- `OCCUPIED` (1)
- `CLOSED` (2)

`std::vector<Edge*> edges`

Array containing the addresses of the site's edges.

`std::vector<Site*> neighbors`

Array containing the addresses of the site's neighboring sites.

# Edge

## Description

`Edge` is a class that defines one edge of a `ParticleSystem`.

## Properties

`bool is_vertical`

Boolean that defines whether the edge is horizontal or vertical.

`0` → Horizontal edge

`1` → Vertical edge

`vector<Site*> sites`

Array containing the addresses of the two sites connected to the edge.

`int active_index`

Index of the edge's adress in the `active_edges` array of the `ParticleSystem` class. If the edge is not active, then `active_index = -1`.

# Reservoir

## Description

`Reservoir` is a structure that defines a reservoir connected to a `ParticleSystem`.

## Properties

`unsigned int size`

Size of the reservoir. If the reservoir is connected to the left/right boundary of the system, then `size` is equal to the width of the system. If the reservoir is connected to the top/bottom boundary of the system, then `size` is equal to the length of the system.

`std::bernoulli_distribution bernoulli`

Bernoulli distribution of the reservoir.

# Particle system

## Description

`ParticleSystem` defines a 2-dimensional lattice, where active particles jump randomly at rate 1 to each empty nearest neighbor.

## Properties

`unsigned int L1`

Length of the system.

`unsigned int L2`

Width of the system.

`unsigned int system_size`

Size of the system (`= L1 * L2`).

`unsigned int dimension`

Dimension of the system (`= 1 or 2`).

`std::vector<unsigned int> boundaries`

Array containing the system boundaries states. Three possible values:

- `OPENED` (`0`)

- `CLOSED` (1)
- `RESERVOIR` (2)

`std::vector<Site> sites`

Array containing all the sites of the system.

`std::vector<Edge> edges`

Array containing all the edges of the system.

`std::vector<Edge*> active_edges`

Array containing all the active edges adresses of the system. This attribute enables to select a random active edge in the `singleStep` method.

`std::map<unsigned int, Reservoir> reservoirs`

Array containing the reservoirs connected to the system.

`unsigned int n_particles`

Number of particles in the site.

`unsigned int n_edges`

Number of edges in the site.

`unsigned int n_vertical_edges`

Number of vertical edges.

`unsigned int n_horizontal_edges`

Number of horizontal edges.

`unsigned int n_active_edges`

Number of active edges.

`unsigned int n_reservoirs`

Number of sites connected to a reservoir.

`std::vector<unsigned int> n_occupied_sites`

Array of size `L1` containing the number of occupied sites at each section `x` of the system.

`std::vector<unsigned int> n_active_sites`

Array of size `L1` containing the number of active sites at each section `x` of the system.

`std::vector<unsigned int> activity`

Array of size `L1` containing the number of active edges at each section `x` of the system. To avoid counting the same edge multiple times, only the upper edge and the right edge of each site are considered.

## Methods

`void initSystem(const std::vector<bool> particles_array)`

Initialize the particle system with a `L1*L2` boolean array.

`0` → Empty site

`1` → Occupied site

Example:

```
unsigned int L = 10;
SystemConfiguration system_config = {0, 1, OPENED, OPENED};

std::vector<bool> particles_array = {0 , 1 , 1 , 0 , 0 , 1 , 0 , 1 , 1 , 0};

ParticleSystem* system = createSystem(L, 1, system_config);
system->initSystem(particles_array);
```

`void initSystem(const double particle_density)`

Initialize the particle system with a particle density between `0` and `1`.

Example:

```
unsigned int L = 10;
SystemConfiguration system_config = {0, 1, OPENED, OPENED};

double particles_density = 0.3;

ParticleSystem* system = createSystem(L, L, system_config);
system->initSystem(particles_density);
```

`void initSystem(const double density_a, const double density_b)`

Initialize the particle system with a particle density that varies from `density_a` to `density_b` through the system. `density_a` and `density_b` are between `0` and `1`.

In the first column of the system, the particle density is equal to `density_a`. In the column `i`, the particle density is equal to `density_a + (density_b - density_a) * i/(L2-1)`. In the last column of the system, the particle density is equal to `density_b`.

Example:

```
unsigned int L = 10;
SystemConfiguration system_config = {0, 1, OPENED, OPENED};

double density_a = 0.;
double density_b = 0.9;

ParticleSystem* system = createSystem(L, L, system_config);
system->initSystem(density_a, density_b);
```

### `void setReservoirDensity(unsigned int reservoir_position, double diffusion_rate)`

Set the diffusion rate of the reservoir at position `reservoir_density`. `diffusion_rate` must be between `0` and `1`.

Example:

```
unsigned int L = 10;
SystemConfiguration system_config = {0, 1, OPENED, OPENED};

double particles_density = 0.25;

double alpha_0 = 0.3;
double alpha_L = 0.7;

ParticleSystem* system = createSystem(L, L, system_config);
system->initSystem(particles_density);

system->setResevoirDensity(LEFT, alpha_0);
system->setReservoirDensity(RIGHT, alpha_L);
```

### `unsigned int getRandomInt(const unsigned int a, const unsigned int b) const`

Return a random integer from `a` to `b`.

### `Site* getSite(const unsigned int x, const unsigned int y)`

Return the adress of the system site at position `(x,y)`. `x` must be between `0` and `L1-1`, and `y` must be between `0` and `L2-1`.

Example:

```
unsigned int x = system->getRandomInt(0, L1-1);
unsigned int y = system->getRandomInt(0, L2-1);
Site* site = system->getSite(x, y);
```

### `bool isAnEmptySite(const Site* site) const`

Return `1` if the site is empty.

### `bool isAnOccupiedSite(const Site* site) const`

Return `1` if the site is occupied.

### `void addParticle(Site* site)`

Add a particle at a site of the system. The site must not be occupied.

Example:

```
Site* site;
do {
  unsigned int x = system->getRandomInt(0, L1-1);
  unsigned int y = system->getRandomInt(0, L2-1);
  site = system->getSite(x, y);
}while(system->isAnEmptySite(site));

system->addParticle(site);
```

### `void removeParticle(Site* site)`

Remove a particle at a site of the system. The site must be occupied.

Example:

```
Site* site;
do {
  unsigned int x = system->getRandomInt(0, L1-1);
  unsigned int y = system->getRandomInt(0, L2-1);
  site = system->getSite(x, y);
}while(system->isAnOccupiedSite(site));

system->removeParticle(site);
```

### `double singleStep()`

Select a random active edge and move the particle to the empty site. If the system is connected to reservoirs, then, in addition to performing a particle jump, the `singleStep` method can also activate a reservoir. This involves modifying the state of one of the sites connected to a reservoir.

Return the macroscopic time of the step `dt = 1/n_actions` (with `n_actions = n_active_edges + n_reservoirs`).

### `unsigned int getLength() const`

Return the length of the system.

### `unsigned int getWidth() const`

Return the width of the system.

### `unsigned int getSize() const`

Return the number of sites of the system.

### `unsigned int getDimension() const`

Return the dimension of the system.

### `unsigned int getTotalNumberOfOccupiedSites() const`

Return the total number of occupied sites of the system.

### `unsigned int getTotalNumberOfActiveSites() const`

Return the total number of active sites of the system.

### `unsigned int getTotalNumberOfActiveEdges() const`

Return the total number of active edges of the system.

### `Edge* getActiveEdge(const unsigned int i) const`

Return a specific active edge. `i` must be between `0` and `n_active_edges-1`.

`Edge* getLastJump() const`

Return the adress of the last active edge that has been selected.

`vector<Site*> getSites(const Edge* edge) const`

Return an array containing the adresses of the two sites at each side of the edge.

`output[0] = left_site` and `output[1] = right_site` if the edge is vertical. `output[0] = top_site` and `output[1] = bottom_site` if the edge is horizontal.

`unsigned int getNumberOfOccupiedSites(const unsigned int x) const`

Return the number of occupied sites at the section of the system `n_occupied_sites`.

`unsigned int getNumberOfActiveSites(const unsigned int x) const`

Return the number of active sites at the section `x` of the system `n_active_sites`.

`unsigned int getNumberOfActiveEdges(const unsigned int x) const`

Return the number of active edges at the section `x` of the system `activity`.

`bool isNotAClosedSite(const Site* site) const`

Return `1` if the site is not closed.

`bool isInsideTheSystem(const Site* site) const`

Return `1` if the site is inside the system.

`std::vector<bool> getParticlesArray() const`

Returns a `L1*L2` boolean containing the states of all sites of the system.

`0` → Empty site

`1` → Occupied site

`void saveParticlesArray(const string filename) const`

Write the position of all occupied sites in a .txt file.

# Useful functions

`ParticleSystem* createSystem(const unsigned int horizontal_direction, const unsigned int vertical_direction, const SystemConfiguration system_config)`

Create a subclass of `ParticleSystem`.

**Input parameters:**

- `unsigned int horizontal_direction`

  Length of the system (must be greater than `0`)

- `unsigned int vertical_direction`

  Width of the system (must be greater than `0`)

- `SystemConfiguration system_config`

  Configuration of the particle system

```
std::vector<bool> loadSystemArray(const string filename,
const unsigned int system_size, const unsigned int system_dimension)
```

Return a boolean array of size `system_size` created from a .txt file containing the position of all occupied sites (see the method `saveParticlesArray`).