

番号 09090H01-02

2010 年 2 月 26 日

独立行政法人理化学研究所 御中

アドバンスソフト株式会社

<表題>

【粒子反応拡散シミュレータ用蛍光顕微像模擬モジュールプロトタイプの開発】  
使用説明書

---

承認		審査	作成

## 【粒子反応拡散シミュレータ用蛍光顕微像模擬モジュールプロトタイプの開発】

## 使用説明書

## 目 次

1. はじめに .....	1
2. 可視化モジュールの開発履歴 .....	1
3. フォルダ構成 .....	2
4. モジュール構成 .....	3
5. 開発環境 .....	3
6. 画面構成 .....	4
6.1. 通常モード .....	4
6.2. 蛍光顕微像モード .....	5
7. 既定の色定義 .....	6
8. 可視化に利用される HDF5 データ .....	7
8.1. データ構造 .....	7
8.2. 可視化のタイムシーケンス .....	8
8.3. データセットのフォーマット .....	8
9. 可視化のデフォルト設定 .....	9
10. API の仕様 .....	11
10.1. VisualizerError クラス .....	12
10.2. Settings クラス .....	12
10.2.1. コンストラクタ .....	12
10.2.2. dump .....	13
10.2.3. set_camera .....	13
10.2.4. set_image .....	13
10.2.5. set_ffmpeg .....	14
10.2.6. set_light .....	15
10.2.7. set_species_legend .....	15
10.2.8. set_time_legend .....	16
10.2.9. set_wireframed_cube .....	16
10.2.10. set_axis_annotation .....	17
10.2.11. set_fluorimetry .....	17
10.2.12. set_pattrs .....	18
10.2.13. set_dattrs .....	18
10.2.14. set_pfilters .....	19
10.2.15. add_plane_surface .....	20
10.3. Visualizer クラス .....	21
10.3.1. コンストラクタ .....	21

10.3.2.	output_snapshots .....	22
10.3.3.	make_movie .....	22
10.3.4.	output_movie .....	22
11.	visualizer を利用した可視化サンプルコード .....	24
12.	付録：ソースコード .....	27
12.1.	domain_kind_constants.py .....	27
12.2.	rgb_colors.py .....	27
12.3.	default_settings.py .....	28
12.4.	visualizer.py .....	33
12.5.	logger.py .....	58

## 1. はじめに

独立行政法人理化学研究所様では、細胞シミュレーション研究コミュニティにおいて共有可能な、世界最高水準のソフトウェア（E-CELL）を開発している。

本件の作業対象コードは First-Passage Kinetic Monte-Carlo(FPKMC)法の実装しており、大きさを持つ球形の剛体粒子を用いて反応拡散問題を解く。FPKMC において個々の粒子は保護領域(Protective Domain; PD)により他の粒子から保護されている。各々の粒子は保護領域の内部を拡散する。このとき、保護領域の端となる球殻は吸収境界条件を持つと仮定して拡散方程式を解くことで、乱数を用いて粒子の拡散後の位置を決める。二つの粒子の位置が十分に近い場合には、二つの粒子を一つの保護領域で囲い、粒子間の距離は粒子が接触するまで近づく位置に放射境界条件を課して拡散と反応を同時に解く。個々の保護領域内での拡散あるいは反応拡散は、非同期で計算される。

また既に Python 言語で書かれたシミュレータコードに、粒子や境界などのシミュレーションモデルの状態をファイルに入出力し、さらに保存されたファイルを 3 次元画像として可視化、動画化する機能が開発されている。

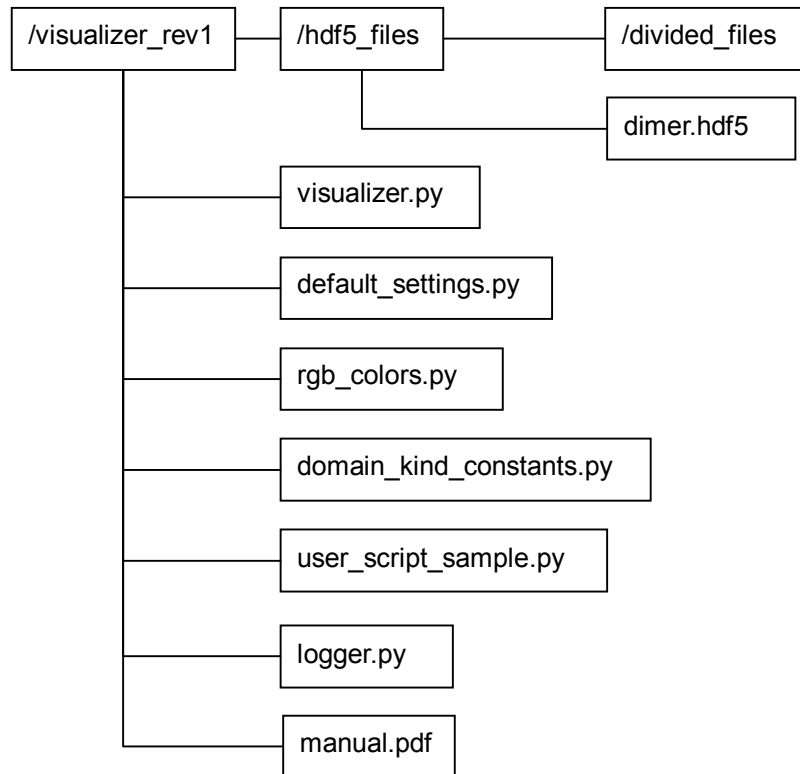
本件では、この可視化モジュールに新たに蛍光顕微像模擬モジュールプロトタイプ機能を追加した。本書は、機能追加された可視化モジュールに関する使用説明書である。

## 2. 可視化モジュールの開発履歴

リビジョン	リリース年月日	内容
0	2010/1/25	・初回リリース
1*	2010/2/26	・蛍光顕微像模擬モジュールプロトタイプ機能の追加 ・VisualizerError クラスの追加 ・バグフィックス対応:(現象)新しい Settings クラスを生成しても古いオブジェクトの設定が継承される。(原因)リストと辞書を利用したデフォルト変数の参照を Settings クラスの属性に代入していた。(対応)それらの変数に対して deepcopy を行うように変更した。

\*注:本件での開発内容。

### 3. フォルダ構成



ファイル・フォルダ名	内容
<code>/visualizer_rev1</code>	リビジョン1の visualizer パッケージ
<code>rgb_colors.py</code>	色見本の定義
<code>default_settings.py</code>	粒子の可視化特性のデフォルト値を定義
<code>user_script_sample.py</code>	ユーザスクリプトのサンプル
<code>domain_kind_constants.py</code>	domain 名などの定数を定義
<code>visualizer.py</code>	粒子・保護球可視化コード (HDF5 ファイルの読み込み、レンダラーの構成、画像出力、動画の作成)
<code>logger.py</code>	e-cell シミュレータ内部に埋め込む HDF5 ファイルの出力コード
<code>manual.pdf</code>	本使用説明書
<code>/hdf5_files</code>	<code>user_script_sample.py</code> で利用されるサンプルデータ
<code>dimer.hdf5</code>	複数時刻の粒子・シェルデータを含んだデータファイル
<code>/divided_files</code>	<code>dimer.hdf5</code> を時刻別、粒子、シェル別に保存したデータ。

4. モジュール構成

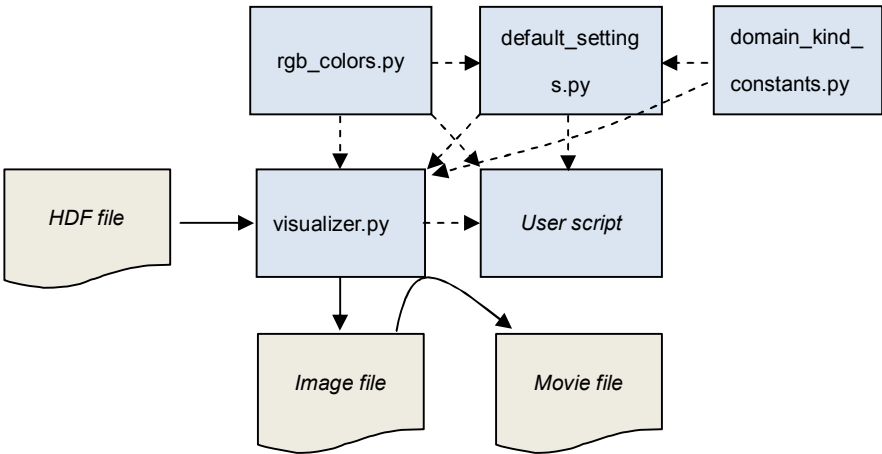


図 1:プログラムの構成  
(実線矢印:データ IO, 点線矢印:import による取り込み)

5. 開発環境

環境	項目	内容
ハードウェア	CPU	Intel Core2 6300 1.86GHz
	HDD	200GB (USB 外付け HDD)
	メモリー	2.0GB
	グラフィックカード	NVIDIA Quadro NVS 285 (256MB)
ソフトウェア	オペレーティングシステム	Ubuntu-9.0.4 Linux
	プログラム開発言語	Python-2.6.2
	外部ライブラリ	NumPy-1.2.1, VTK-5.0.4, h5py-1.2.1, HDF5-1.6.6, FFmpeg (SVN-r20796)
	IDE	Eclipse-3.5, pydev-1.5.0, pleiades-1.3.2

本可視化機能を動作させるには、標準の e-cell パッケージで要求されるライブラリの他に、VTK(python binding を含む),h5py, HDF5, FFmpeg が要求されるため、予めこれらのソフトウェアをインストールする必要がある。

## 6. 画面構成

### 6.1. 通常モード

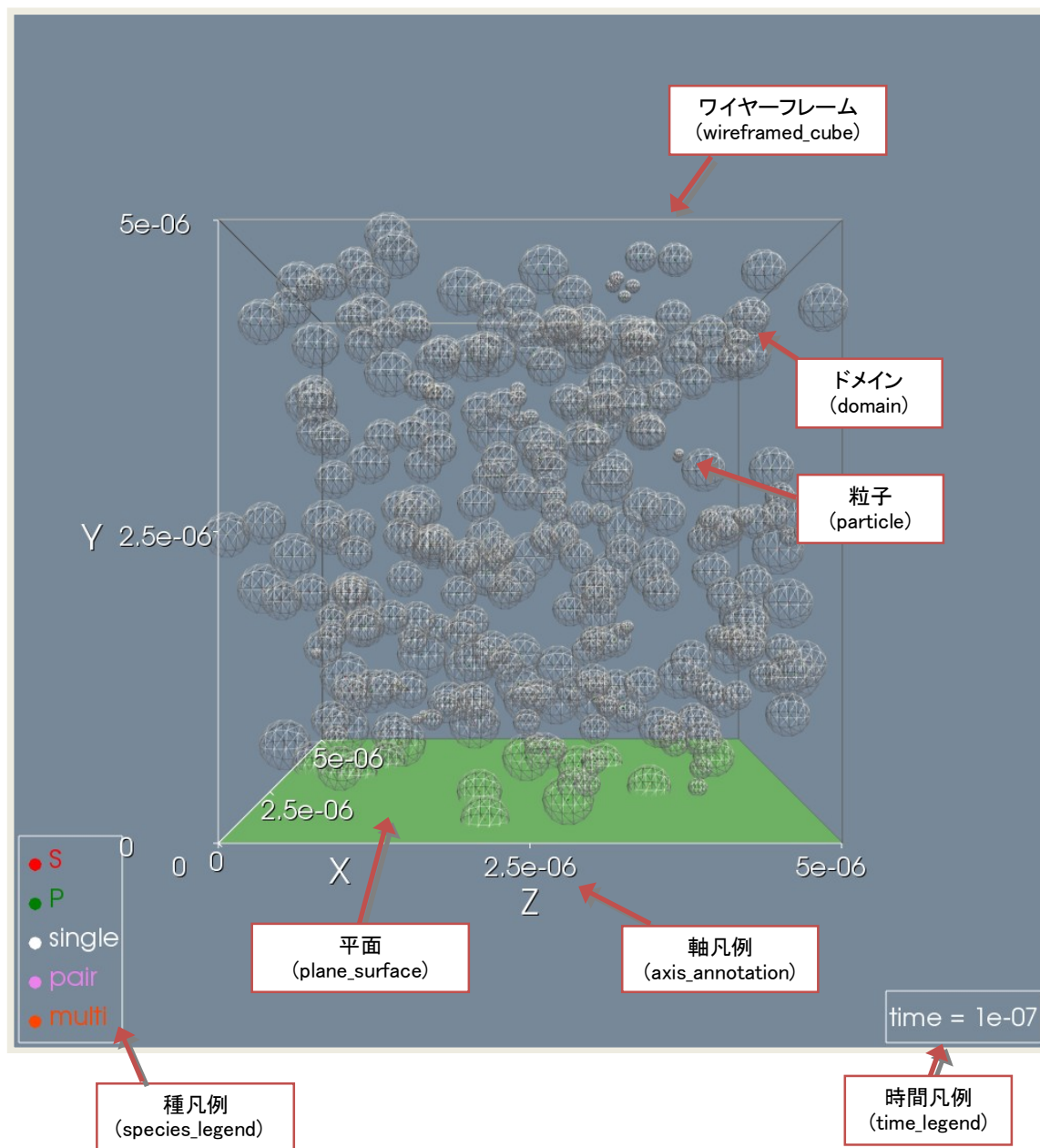


図 2:可視化された画面の構成(通常モード)

## 6.2. 蛍光顕微像モード

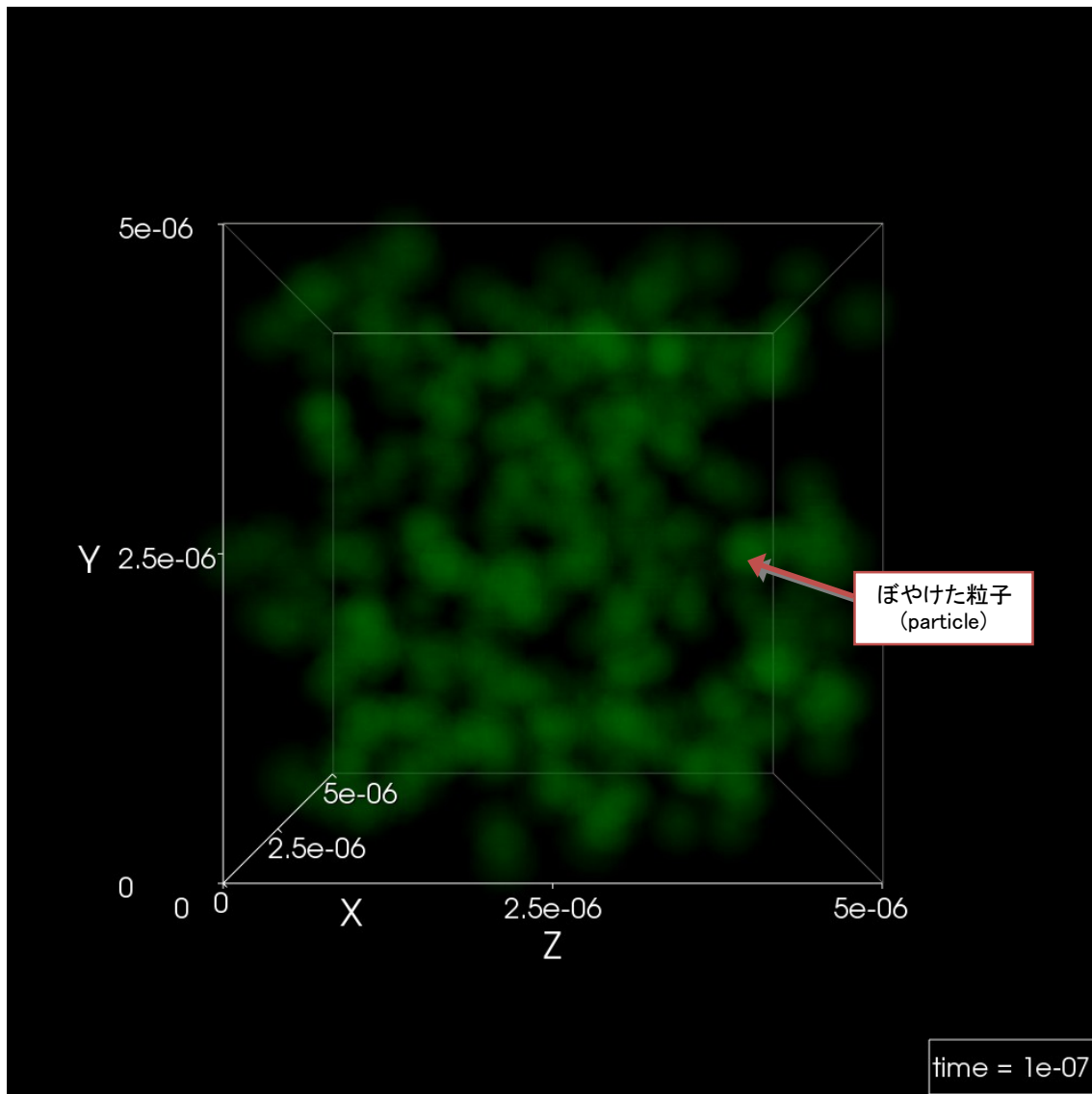


図 3:可視化された画面の構成(蛍光顕微像モード)

通常モードとの違い：


- 光の波長の広がりを持った粒子が表示される。
- ドメイン・種凡例は表示されない。
- 背景色が通常モードとは独自に設定される。
- 現時点で通常モードで利用できるフィルター関数は利用できない。

表示モードの変更の仕方については、第 10.2.11 節参照。



## 7. 既定の色定義

rgb\_colors.py で定義された色見本 (<http://www.w3.org/TR/css3-color/#svg-color> より抜粋)

用途	色	Color name
テキスト		RGB_BLACK
		RGB_WHITE
粒子		RGB_RED
		RGB_GREEN
		RGB_BLUE
		RGB_PURPLE
		RGB_ORANGE
		RGB_YELLOW
		RGB_CYAN
		RGB_GRAY
		RGB_YELLOW_GREEN
		RGB_LIGHT_GREEN
		RGB_DARK_GREEN
		RGB_LIGHT_BLUE
		RGB_DARK_BLUE
		RGB_LIGHT_CYAN
		RGB_DARK_CYAN
		RGB_LIGHT_GRAY
ドメイン		RGB_WHITE
		RGB_VIOLET
		RGB_ORANGE_RED
背景		RGB_LIGHT_SLATE_GRAY
		RGB_DARK_SLATE_BLUE
		RGB_DARK_SLATE_GRAY
		RGB_SLATE_BLUE
		RGB_SLATE_GRAY

## 8. 可視化に利用される HDF5 データ

### 8.1. データ構造

図 4 に HDF5 のデータ構造を示す。各時刻の粒子・シェルデータは `data` グループの下に時間を表すグループが置かれる。その各グループには少なくとも `particles` もしくは `shells`, `shells_particle_association`, `domain_shell_association`, `domains` のデータが含まれている必要がある。ルート直下には、必ず `species` データセットが置かれ、粒子を可視化する上で半径と名前が参照される。

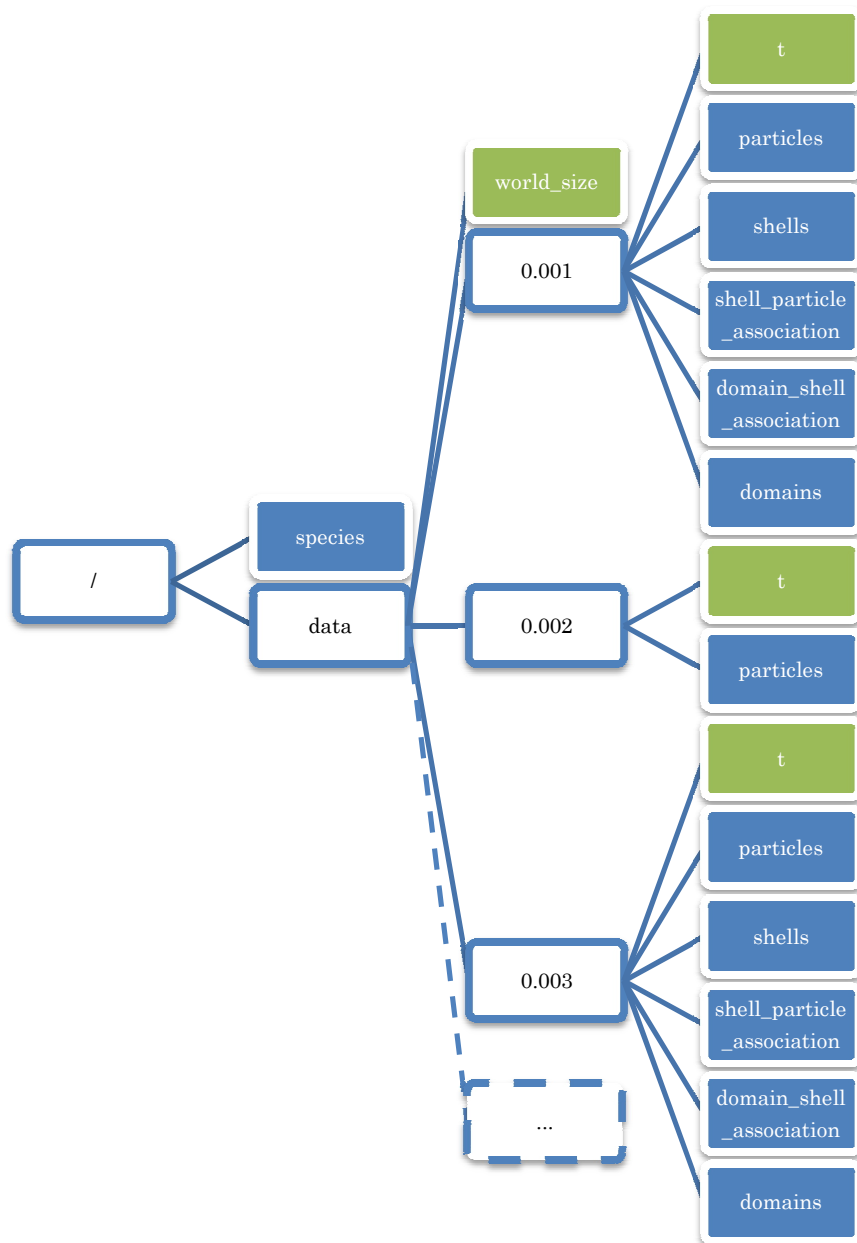


図 4: 可視化に利用される HDF5 ファイルのデータ構造

□はデータグループ、■はデータセット、■はアトリビュートを表す。

## 8.2. 可視化のタイムシーケンス

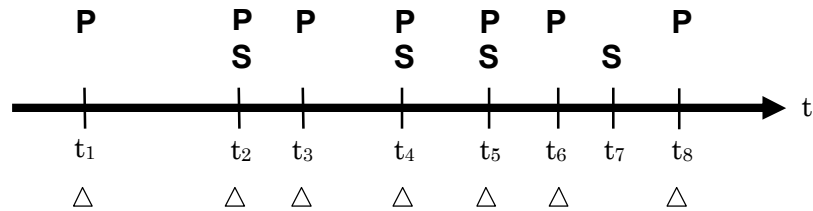


図 5: 可視化のタイムシーケンス。P,S はそれぞれ HDF5 データに含まれる particles データセットおよび shells, shell\_particle\_association, domain\_shell\_association, domains データセットの組を表す。可視化時刻は、データセット P がある時刻で決められ、それを基にした粒子の可視化と、同時刻のデータセット S もしくはそれがなければその時刻に最も近い過去のデータセット S を基にしたドメインの可視化がなされる。△記号が可視化される時刻である。

## 8.3. データセットのフォーマット

可視化に利用されるデータセットは、1 次元配列として与えられる。配列の各要素のデータフォーマットは、タグ名とビルドイン型のペアの組み合わせで指定される。u4,u8 は 4, 8 バイト符号なし整数、S32 は 32 文字の文字列、f8 は 8 バイト実数を表す。

データセット	データフォーマット
species	[ ('id', 'u8',), ('name', 'S32',), ('radius', 'f8',), ('D', 'f8',), # diffusion coefficient ]
particles	[ ('id', 'u8',), ('species_id', 'u8',), ('position', 'f8', (3,)) ]
shells	[ ('id', 'u8',), ('radius', 'f8',), ('position', 'f8', (3,)), ]

	]
domains	[ ( 'id', 'u8', ), ( 'kind', 'u4', ), ]
shell_particle_association	[ ( 'shell_id', 'u8', ), ( 'particle_id', 'u8', ), ]
domain_shell_association	[ ( 'shell_id', 'u8', ), ( 'domain_id', 'u8', ), ]

## 9. 可視化のデフォルト設定

可視化設定のデフォルト値は、全て `default_settings.py` で与えられる。それらの変数と値は下記の表の通りである。これらの変数とデフォルト値は全て後述する `Settings` クラスの属性値となる。

変数名	値
<code>image_height</code>	1000
<code>image_width</code>	1000
<code>image_background_color</code>	<code>rgb_colors.RGB_LIGHT_SLATE_GRAY</code>
<code>image_file_name_format</code>	<code>'image_%04d.png'</code>
<code>ffmpeg_movie_file_name</code>	<code>'movie.mp4'</code>
<code>ffmpeg_bin_path</code>	<code>''</code>
<code>ffmpeg_additional_options</code>	<code>'-sameq -r 5'</code>
<code>camera_focal_point</code>	<code>(0.5, 0.5, 0.5)</code>
<code>camera_base_position</code>	<code>(-2.0, 0.5, 0.5)</code>
<code>camera_azimuth</code>	0
<code>camera_elevation</code>	0
<code>camera_view_angle</code>	45
<code>light_intensity</code>	1

species_legend_display	True
species_legend_border_display	True
species_legend_location	0
species_legend_height	0.2
species_legend_width	0.1
species_legend_offset	0.005
time_legend_display	True
time_legend_border_display	True
time_legend_format	'time = %g'
time_legend_location	1
time_legend_height	0.05
time_legend_width	0.15
time_legend_offset	0.005
wireframed_cube_display	True
axis_annotation_display	True
axis_annotation_color	rgb_colors.RGB_WHITE
default_pattr	{ 1: {'color':rgb_colors.RGB_RED, 'opacity':1.0}, 2: {'color':rgb_colors.RGB_GREEN, 'opacity':1.0}, 3: {'color':rgb_colors.RGB_BLUE, 'opacity':1.0}, 4: {'color':rgb_colors.RGB_PURPLE, 'opacity':1.0}, 5: {'color':rgb_colors.RGB_ORANGE, 'opacity':1.0}, 6: {'color':rgb_colors.RGB_YELLOW, 'opacity':1.0}, 7: {'color':rgb_colors.RGB_CYAN, 'opacity':1.0}, 8: {'color':rgb_colors.RGB_GRAY, 'opacity':1.0}, 9: {'color':rgb_colors.RGB_YELLOW_GREEN, 'opacity':1.0}, 10: {'color':rgb_colors.RGB_LIGHT_GREEN, 'opacity':1.0}, 11: {'color':rgb_colors.RGB_DARK_GREEN, 'opacity':1.0}, 12: {'color':rgb_colors.RGB_LIGHT_BLUE, 'opacity':1.0}, 13: {'color':rgb_colors.RGB_DARK_BLUE, 'opacity':1.0}, 14: {'color':rgb_colors.RGB_LIGHT_CYAN, 'opacity':1.0}, 15: {'color':rgb_colors.RGB_DARK_CYAN, 'opacity':1.0}, 16: {'color':rgb_colors.RGB_LIGHT_GRAY, 'opacity':1.0}, }
undefined_pattr	{'color':rgb_colors.RGB_BLACK, 'opacity':0.5}

user_pattr	{}
default_dattr	{ domain_kind_constants.SINGLE: {'color':rgb_colors.RGB_WHITE, 'opacity':0.2}, domain_kind_constants.PAIR: {'color':rgb_colors.RGB_VIOLET, 'opacity':0.2}, domain_kind_constants.MULTI: {'color':rgb_colors.RGB_ORANGE_RED, 'opacity':0.2}, }
undefined_dattr	{'color':rgb_colors.RGB_BLACK, 'opacity':0.2}
user_dattr	{}
plane_surface_color	rgb_colors.RGB_WHITE
plane_surface_opacity	1
plane_surface_origin	(0, 0, 0)
plane_surface_axis_1	(1, 0, 0)
plane_surface_axis_2	(0, 1, 0)
plane_surface_list	[]
pfilter_pid_func	None
pfilter_pos_func	None
pfilter_sid_func	None
pfilter_sid_map	None
pfilter_sid_map_func	None
fluorimetry_display*	False
fluorimetry_wave_length*	546.0e-9
fluorimetry_luminescence_color*	rgb_colors.RGB_GREEN
fluorimetry_axial_voxel_number*	100
fluorimetry_background_color*	rgb_colors.RGB_BLACK
fluorimetry_shadow_display*	False
fluorimetry_accumulation_mode*	0
fluorimetry_brightness*	1.0

\*注:リビジョン1で新たに追加された変数。

## 10. API の仕様

### 10.1. VisualizerError クラス

本クラスは、`visualizer.py` モジュール内で利用される例外クラスである。`visualizer.py` モジュール内でユーザーによる設定エラーが生じた場合、このクラスが投げられる。以下に仕様を示す。

```
class VisualizerError(Exception):
    "Exception class for visualizer"
    def __init__(self, info):
        self.__info = info
    def __repr__(self):
        return self.__info
    def __str__(self):
        return self.__info
```

### 10.2. Settings クラス

本クラスは、後述する `Visualizer` クラスのための可視化設定を行うクラスである。このクラスに含まれる属性は、`default_settings` モジュールに含まれるパブリック属性と同一である。従って、`Settings` クラスの属性として、第 9 節で示した変数に直接アクセスすることができる。

例：画像サイズの高さを 1000 に設定する。

```
import visualizer
settings = visualizer.Settings()
settings.image_height = 1000
```

その他に `Settings` クラスの属性値を与える手段として、ラッパー関数が提供されている。その関数の命名規則は、変数のカテゴリーを `XXX`、値を `YYY` とすると、基本的に「`set_XXX (YYY=value)`」という形式で与えられている。

#### 10.2.1. コンストラクタ

引数仕様		
<pre>def __init__(self,               user_settings_dict = None               ):</pre>		
変数名	型	内容
<code>user_settings_dict</code>	マップ型	<code>Settings</code> クラスの属性値をマップ型で指定。但し、指定できる属性は、 <code>default_settings</code> モジュールで定義されたパブリック属性に限る。変数が与えられた場

		合、default_settings モジュールで定義された_dict_とマージした設定が使われる。
--	--	--

## 10.2.2. dump

引数仕様
def dump( <i>self</i> ):
機能: Settings クラスに定義された設定値をソートして print する。

## 10.2.3. set\_camera

引数仕様

```
def set_camera (self,
                forcal_point = None,
                base_position = None,
                azimuth = None,
                elevation = None,
                view_angle = None
                ):
```

変数名	型	内容
forcal_point	(x,y,z) (但し x,y,z は、world_size を 1	カメラの焦点座標を指定。(例:(0.5,0.5,0.5))
base_position	単位とした実数座標値で指定。)	カメラの基本座標を指定。(例:(-2.0,0.5,0.5))
azimuth	実数	カメラの基本座標からの方位方向の移動量を角度[deg]で指定。
elevation	実数	カメラの基本座標からの高度方向の移動量を角度[deg]で指定。
view_angle	実数	カメラの視野角を角度[deg]で指定。

## 10.2.4. set\_image



## 引数仕様

```
def set_image (self,
               height = None,
               width = None,
               background_color = None,
               file_name_format = None
               ):
```

変数名	型	内容
height	整数	画像サイズの高さをピクセル単位で指定。
width	整数	画像サイズの幅をピクセル単位で指定。
background_color	(r,g,b) ( $0 \leq r,g,b \leq 1$ )	画像の背景色を r,g,b 値で指定。
file_name_format	文字列	画像ファイルのカウンタ指定子を含めた画像ファイル名のフォーマット。(例: image_%04d.png) 注: 指定された拡張子から画像ファイルの形式が決められる。利用可能な拡張子は, .bmp, .tif, .bmp, .png の4種類である。

## 10.2.5. set\_ffmpeg

## 引数仕様

```
def set_ffmpeg (self,
                movie_file_name = None,
                bin_path = None,
                additional_options = None
                ):
```

変数名	型	内容
movie_file_name	文字列	保存する動画ファイル名を指定。
bin_path	文字列	ffmpeg のバイナリパスを指定。何も指定がないと、環境変数\$PATH から ffmpeg のバイナリをサーチして実行する。
additional_options	文字列	ffmpeg のコマンドオプションを指定。 (例: -sameq -r 5) 但し、入出力ファイルの指定は、Visualizer クラス内部で行うため、このオプションで指定してはならな

		い。
--	--	----

## 10.2.6. set\_light

引数仕様		
<pre>def set_light (self,                intensity = None                ): </pre>		
変数名	型	内容
intensity	実数	光の強度を指定。(例:1.0)

## 10.2.7. set\_species\_legend

引数仕様		
<pre>def set_species_legend (self,                         display = None,                         border_display = None,                         location = None,                         height = None,                         width = None,                         offset = None                         ): </pre>		
変数名	型	内容
display	True, False	種凡例を表示するか否か。
border_display	True, False	凡例枠を表示するか否か。
location	整数	凡例の画面上の位置 (0:左下,1:右下,2:左上,3:右上)
height <sup>注</sup>	実数	凡例の高さ。但し、画像サイズの高さに対する比で指定。(例:0.2)
width <sup>注</sup>	実数	凡例の幅。但し、画像サイズの幅に対する比で指定。(例:0.1)
offset	実数	画像の隅から凡例をどれくらいずらして表示するかを指定。画像サイズの比で指定。(例:0.05)

**注:** 凡例のフォントサイズは、与えられた width,height から必要な大きさを調整して決められる。フォントが小さくなりすぎる場合には、これらを大きく取る必要がある。

## 10.2.8. set\_time\_legend

引数仕様		
<pre>def set_time_legend (self,                     display = None,                     border_display = None,                     legend_format = None,                     location = None,                     height = None,                     width = None,                     offset = None                     ):</pre>		
変数名	型	内容
display	True, False	時間凡例を表示するか否かを指定。
border_display	True, False	凡例枠を表示するか否かを指定。
legend_format	文字列	時間表示のフォーマットを指定。(例:time=%g)
location	整数	凡例の画面上の位置(0:左下,1:右下,2:左上,3:右上)
height <sup>注</sup>	実数	凡例の高さ。但し、画像サイズの高さに対する比で指定。(例:0.1)
width <sup>注</sup>	実数	凡例の幅。但し、画像サイズの幅に対する比で指定。(例:0.2)
offset	実数	画像の隅から凡例をどれくらいずらして表示するかを指定。画像サイズの比で指定。(例:0.05)

注: 凡例のフォントサイズは、width,height を固定して凡例に含まれる記号と文字列の大きさから調整される。フォントが小さくなりすぎる場合には、これらを適宜調整する必要がある。

## 10.2.9. set\_wireframed\_cube

引数仕様		
<pre>def set_wireframed_cube (self,                         display = None                         ):</pre>		
変数名	型	内容
display	True, False	ワイヤーフレームを表示するか否かを指定。

## 10.2.10. set\_axis\_annotation

引数仕様		
<pre>def set_axis_annotation (self,                         display = None,                         color = None                         ): </pre>		
変数名	型	内容
display	True, False	軸凡例を表示するか否かを指定。
color	(r,g,b) ( $0 \leq r,g,b \leq 1$ )	軸凡例の文字色を r,g,b 値で指定。

## 10.2.11. set\_fluorimetry

引数仕様		
<pre>def set_fluorimetry (self,                     display = None,                     wave_length = None,                     luminescence_color = None,                     axial_voxel_number = None,                     background_color = None,                     shadow_display = None,                     accumulation_mode = None,                     brightness = None                     ): </pre>		
変数名	型	内容
display	True, False	蛍光顕微像モードで表示するか否かを指定。
wave_length	実数(>0)	光の波長[m] (粒子をぼかす際の長さ)を指定。 (例 : 546.0e-9)
luminescence_color	(r,g,b) ( $0 \leq r,g,b \leq 1$ )	蛍光色を指定。
axial_voxel_number	整数(>0)	一軸当たりの解像度(ボクセル数)を指定。
background_color	(r,g,b) ( $0 \leq r,g,b \leq 1$ )	蛍光顕微像モードでの背景色を指定。
shadow_display	True, False	影付けの有無を指定。

<b>accumulation_mode</b>	整数	ガウス分布の重なり処理(0:max, 1:sum)を指定。
<b>brightness</b>	実数(>0)	明るさを指定。(例: 1.0)

## 10.2.12. set\_pattr

引数仕様		
<pre>def set_pattr (self,                species_id,                color = None,                opacity = None,                name = None,                radius = None                ):</pre>		
変数名	型	内容
<b>species_id</b>	整数	属性を設定する粒子種のインデックス。入力必須。
<b>color</b>	(r,g,b) ( $0 \leq r,g,b \leq 1$ )	粒子の表示色を r,g,b 値で指定。
<b>opacity</b>	実数	粒子の不透明度を 0 以上,1 以下で指定。
<b>name</b>	文字列	種凡例に記載する粒子の名前を指定。
<b>radius</b>	実数	粒子の大きさを設定。但し、単位は e-cell の粒子半径の単位と同一。

## 10.2.13. set\_dattr

引数仕様		
<pre>def set_dattr (self,                domain_kind,                color = None,                opacity = None                ):</pre>		
変数名	型	内容
<b>domain_kind</b>	整数	属性を設定するドメイン種のインデックス。 1:single,2:pair,3:multi のいずれかを指定。入力必須。
<b>color</b>	(r,g,b) ( $0 \leq r,g,b \leq 1$ )	ドメインの表示色を r,g,b 値で指定。

opacity	実数	ドメインの不透明度を 0 以上,1 以下で指定。
---------	----	--------------------------

## 10.2.14. set\_pfilters

引数仕様		
<pre>def set_pfilters (self,                   pid_func = None,                   pos_func = None,                   sid_func = None,                   sid_map = None,                   sid_map_func = None,                   ): </pre>		
変数名	型	内容
pid_func	関数オブジェクト	粒子の表示・非表示を粒子 id で決定するフィルター関数を指定。
pos_func	関数オブジェクト	粒子の表示・非表示を粒子座標で決定するフィルター関数を指定。
sid_func	関数オブジェクト	粒子の属性を粒子種で決定するフィルター関数を指定。
sid_map	{ species_id_0 : display_species_id_0, species_id_1 : display_species_id_1, ... } (但し、各要素は整数。)	HDF5 ファイル内に定義されたオリジナルな粒子種 species_id と表示上の粒子種 display_species_id を対応付ける辞書を指定。この辞書は HDF5 ファイルにある全ての species_id に対するキーが記述されていなければならない。尚、sid_map_func でも同様な設定を行うことができ、もしそれが指定されていた場合、それを優先する。
sid_map_func	関数オブジェクト	HDF5 ファイル内に定義された粒子種 species_id を表示上の粒子種 display_species_id に対応付ける関数を指定。尚、sid_map が指定されていても、こちらの設定が優先される。

set\_pfilters で指定できるコールバック関数の引数仕様

引数仕様		
<code>def user_pfilter_pos_func ( pos ): return display</code>		
変数名	型	内容
<b>pos</b>	(x,y,z)	粒子座標 x,y,z。但し、単位は、e-cell 内部で使われているものと同一。
<b>display</b>	True or False	粒子の表示・非表示を返却。

引数仕様		
<code>def user_pfilter_pid_func ( particle_id ): return display</code>		
変数名	型	内容
<b>particle_id</b>	符号無し整数	粒子 ID。
<b>display</b>	True or False	粒子の表示・非表示を返却。

引数仕様		
<code>def user_pfilter_sid_func ( display_species_id ): return pattr</code>		
変数名	型	内容
<b>display_species_id</b>	符号無し整数	表示上の粒子種 ID。
<b>pattr</b>	{ 'color': color, 'opacity': opacity, 'name': name, 'radius': radius }	粒子の属性を辞書で返却。但し、color=(r,g,b) ( $0 \leq r,g,b \leq 1$ )、opacity は粒子の不透明度( $0 \leq \text{opacity} \leq 1$ )、name は種凡例に記載される種の名前、radius は粒子半径。但し粒子半径の単位は、e-cell 内部で使われているものと同一。

引数仕様		
<code>def user_pfilter_sid_map_func ( species_id ): return display_species_id</code>		
変数名	型	内容
<b>species_id</b>	符号無し整数	HDF5 ファイル内に定義された粒子種 ID。
<b>display_species_id</b>	符号無し整数	表示上の粒子種 ID。

#### 10.2.15. add\_plane\_surface

引数仕様	
<code>def add_plane_surface ( self,</code>	
<code>color = None,</code>	

	<code>opacity = None,</code> <code>origin = None,</code> <code>axis1 = None,</code> <code>axis2 = None</code> <code>):</code>	
変数名	型	内容
<b>color</b>	(r,g,b) ( $0 \leq r,g,b \leq 1$ )	平面オブジェクトの色を定義。
<b>opacity</b>	実数	平面オブジェクトの不透明度を0以上,1以下で設定。
<b>origin</b>	(x,y,z)	平面オブジェクトの原点を定義。
<b>axis1</b>	(但し x,y,z は、 world_size を 1	平面オブジェクトの軸を定義。axis1-origin が軸ベクトルとなる。
<b>axis2</b>	単位とした実数 座 標 値 で 指 定。)	平面オブジェクトの軸を定義。axis2-origin が軸ベクトルとなる。

### 10.3. Visualizer クラス

本クラスは、指定された HDF5 ファイルから 3 次元画像・動画ファイルを出力する。

#### 10.3.1. コンストラクタ

引数仕様		
<pre>def __init__(self,               HDF5_file_path_list,               user_settings =Settings(),               ): </pre>		
変数名	型	内容
<b>HDF5_file_path_list</b>	文字列もしくは 文字列リスト	可視化したい HDF5 ファイルのパスを指定。複数のパスを指定したい場合、パスのリストとして与える。この場合、複数のファイルに含まれるデータを時間系列で連結し可視化を行う。
<b>user_settings</b>	Settings クラス	ユーザー設定された Settings クラスを指定。何も指定がなければ、Settings クラスのコンストラクタが呼び出される。



## 10.3.2. output\_snapshots

引数仕様		
<pre>def output_snapshots (self,                       image_file_dir                       ):     return snapshot_list</pre>		
変数名	型	内容
image_file_dir	文字列	画像ファイルの保存先を指定。
snapshot_list	文字列リスト	保存された画像ファイル名のリストを返却。

## 10.3.3. make\_movie

引数仕様		
<pre>def make_movie (self,                 image_file_dir,                 movie_file_dir                 ):     </pre>		
変数名	型	内容
image_file_dir	文字列	動画ファイルの元となる画像ファイルの保存先を指定。
movie_file_dir	文字列	動画ファイルの保存先を指定。

## 10.3.4. output\_movie

引数仕様		
<pre>def output_movie (self,                   movie_file_dir,                   image_tmp_root = None                   ):     </pre>		
変数名	型	内容
movie_file_dir	文字列	動画ファイルの保存先を指定。
image_tmp_dir	文字列	動画ファイル出力に必要な画像ファイルの一時保存先を指定。何も指定が無ければ、作業フォルダ上の一時フォルダに保存。利用された一時フォルダは、一

		時的に出力された画像ファイルの消去後、フォルダが空であればそれを消去する。
--	--	---------------------------------------

## 11. visualizer を利用した可視化サンプルコード

サンプルを起動させるには、`visualizer_rev1` フォルダにて

```
%python user_script_sample.py
```

を実行するだけでよい。これで、サンプル内に記述された 7 つの可視化ケース(内：通常モード：4 ケース、蛍光顕微像モード：3 ケース)が実行され、同フォルダに動画ファイルおよび画像ファイルを納めたフォルダが生成される。

### user\_script\_sample.py

```
"""
user_script_sample.py:

User script sample for visualizer
"""

from rgb_colors import *
import visualizer
import glob

# Case 1: Most simple script

vs = visualizer.Visualizer('./hdf5_data/dimer.hdf5')
vs.output_movie('./')

# Case 2: Custom Particle Snapshot

settings = visualizer.Settings()
settings.set_image(file_name_format = 'case2_%04d.png')
settings.ffmpeg_movie_file_name = 'case2.mp4'
settings.add_plane_surface (origin = (0.0, 0.0, 0.0),
                             axis1 = (1.0, 0.0, 0.0),
                             axis2 = (0.0, 0.0, 1.0),
                             color = RGB_LIGHT_GREEN)

vs = visualizer.Visualizer('./hdf5_data/dimer.hdf5', settings)
vs.output_snapshots(image_file_dir = './images')
vs.make_movie(image_file_dir = './images', movie_file_dir = './')

# Case 3: Focused Snapshot

settings = visualizer.Settings({'camera_view_angle':5})
```

```

settings.set_image(file_name_format = 'case3_%04d.png')
settings.set_ffmpeg(movie_file_name = 'case3.mp4')

vs = visualizer.Visualizer('./hdf5_data/dimer.hdf5', settings)
vs.output_movie(movie_file_dir = './')

# Case 4: Filtered Particle Snapshot

settings.set_image(file_name_format = 'case4_%04d.png')
settings.set_ffmpeg(movie_file_name = 'case4.mp4')

def user_pfilter_sid_func(display_species_id):
    if(display_species_id == 1):
        return {'color':RGB_YELLOW, 'name':'aaa'}
    else:
        return None

def user_pfilter_sid_map_func(species_id): # return display_species_id
    if(species_id == 1):
        return 2
    else:
        return 1

settings.pfilter_sid_func = user_pfilter_sid_func
settings.pfilter_sid_map_func = user_pfilter_sid_map_func
settings.set_dattrs(1, color = RGB_BLUE)

vs = visualizer.Visualizer(glob.glob('./hdf5_data/*.hdf5'), settings)
vs.output_movie(movie_file_dir = './')

#### Blurry effect cases (Case 5-7)
#### (New feature of revision 1)

settings = visualizer.Settings()

# Case 5: Accumulation mode to max (This is default mode)
settings.set_fluorimetry(display = True)
settings.set_ffmpeg(movie_file_name = 'case5.mp4')
settings.set_image(file_name_format = 'case5_%04d.png')
vs = visualizer.Visualizer('./hdf5_data/dimer.hdf5', settings)
vs.output_snapshots(image_file_dir = './images')
vs.make_movie(image_file_dir = './images', movie_file_dir = './')

```

```
# Case 6: Accumulation mode to sum (opacity_scaling = 1.0)
settings.set_fluorimetry(accumulation_mode = 1)
settings.set_ffmpeg(movie_file_name = 'case6.mp4')
vs.output_movie(movie_file_dir = './')

# Case 7: Accumulation mode to sum (opacity_scaling = 0.5)
settings.set_fluorimetry(brightness = 0.5)
settings.set_ffmpeg(movie_file_name = 'case7.mp4')
vs.output_movie(movie_file_dir = './')

print 'finished'
```

## 12. 付録：ソースコード

### 12.1. domain\_kind\_constants.py

#### domain\_kind\_constants.py

```
"""
domain_kind_constants.py:

Domain kind indices and names are defined for visualizer

"""

SINGLE = 1
PAIR = 2
MULTI = 3

DOMAIN_KIND_NAME = {
    SINGLE: 'single',
    PAIR: 'pair',
    MULTI: 'multi',
}
```

### 12.2. rgb\_colors.py

#### rgb\_colors.py

```
"""
rgb_colors.py:

RGB color list for visualizer
Following names refer 'X11 color names'.
You can see the color at: http://www.w3.org/TR/css3-color/#svg-color, or related pages.

Note:Range of the RGB values is not [0,255], but [0,1].

"""

# Selected colors for text
RGB_WHITE = (1.0, 1.0, 1.0)
RGB_BLACK = (0.0, 0.0, 0.0)

# Selected colors for particles
RGB_RED = (1.0, 0.0, 0.0)
```

```
RGB_GREEN = (0.0, 0.502, 0.0)
RGB_BLUE = (0.0, 0.0, 1.0)
RGB_PURPLE = (0.502, 0.0, 0.502)
RGB_ORANGE = (1.0, 0.647, 0.0)
RGB_YELLOW = (1.0, 1.0, 0.0)
RGB_CYAN = (0.0, 1.0, 1.0)
RGB_GRAY = (0.502, 0.502, 0.502)
RGB_YELLOW_GREEN = (0.604, 0.804, 0.196)
RGB_LIGHT_GREEN = (0.565, 0.933, 0.565)
RGB_DARK_GREEN = (0.0, 0.392, 0.0)
RGB_LIGHT_BLUE = (0.678, 0.847, 0.902)
RGB_DARK_BLUE = (0.0, 0.0, 0.545)
RGB_LIGHT_CYAN = (0.878, 1.0, 1.0)
RGB_DARK_CYAN = (0.0, 0.545, 0.545)
RGB_LIGHT_GRAY = (0.827, 0.827, 0.827)

# Selected colors for shells
RGB_VIOLET = (0.933, 0.510, 0.933) # for pair
RGB_ORANGE_RED = (1.0, 0.271, 0) # for multi

# Selected colors for background
RGB_SLATE_BLUE = (0.416, 0.353, 0.804)
RGB_SLATE_GRAY = (0.439, 0.502, 0.565)
RGB_LIGHT_SLATE_GRAY = (0.467, 0.533, 0.60)
RGB_DARK_SLATE_GRAY = (0.184, 0.310, 0.310)
RGB_DARK_SLATE_BLUE = (0.282, 0.239, 0.545)
```

### 12.3. default\_settings.py

#### **default\_settings.py**

```
"""
default_settigs.py:

    Default settings for visualizer

"""

import rgb_colors
import domain_kind_constants

#-----
# Output image settings
```

```
#-----
image_height = 1000
image_width = 1000
image_background_color = rgb_colors.RGB_LIGHT_SLATE_GRAY
image_file_name_format = 'image_%04d.png' # Must be compatible with FFmpeg's input-file notation

#-----
# FFmpeg command settings
#-----
# Output movie filename
ffmpeg_movie_file_name = 'movie.mp4'

# FFmpeg binary path (ex. '/usr/local/bin/ffmpeg')
# For empty string, trace back to $PATH.
ffmpeg_bin_path = ""

# FFmpeg option (Please specify FFmpeg's options except IO-filename option.)
ffmpeg_additional_options = '-sameq -r 5'

#-----
# Camera settings
#-----
# Focal point of x,y,z (This unit is world_size)
camera_focal_point = (0.5, 0.5, 0.5)

# Base position of x,y,z (This unit is world_size)
camera_base_position = (-2.0, 0.5, 0.5)

# Movement along azimuth direction from base position [degree]
camera_azimuth = 0.0

# Elevation from base position [degree]
camera_elevation = 0.0

# View angle [degree]
camera_view_angle = 45.0

#-----
# Light settings
#-----
light_intensity = 1.0

#-----
```



```
# Species legend settings
#-----
species_legend_display = True
species_legend_border_display = True
species_legend_location = 0 # 0:left bottom, 1:right bottom, 2:left top, 3:right top
species_legend_height = 0.2 # This is normalized to image height.
species_legend_width = 0.1 # This is normalized to image width.
species_legend_offset = 0.005

#-----
# Time legend settings
#-----
time_legend_display = True
time_legend_border_display = True
time_legend_format = 'time = %g'
time_legend_location = 1 # 0:left bottom, 1:right bottom, 2:left top, 3:right top
time_legend_height = 0.05 # This is normalized to image height.
time_legend_width = 0.15 # This is normalized to image width.
time_legend_offset = 0.005

#-----
# Wireframed cube
#-----
wireframed_cube_display = True

#-----
# Axis annotation
#-----
axis_annotation_display = True
axis_annotation_color = rgb_colors.RGB_WHITE

#-----
# Particle attributes
#-----
default_pattns = \
{
    # species_id: {'color':RGB_color, 'opacity':opacity_value}
    1: {'color':rgb_colors.RGB_RED, 'opacity':1.0},
    2: {'color':rgb_colors.RGB_GREEN, 'opacity':1.0},
    3: {'color':rgb_colors.RGB_BLUE, 'opacity':1.0},
    4: {'color':rgb_colors.RGB_PURPLE, 'opacity':1.0},
    5: {'color':rgb_colors.RGB_ORANGE, 'opacity':1.0},
    6: {'color':rgb_colors.RGB_YELLOW, 'opacity':1.0},
```

```

7: {'color':rgb_colors.RGB_CYAN, 'opacity':1.0},
8: {'color':rgb_colors.RGB_GRAY, 'opacity':1.0},
9: {'color':rgb_colors.RGB_YELLOW_GREEN, 'opacity':1.0},
10: {'color':rgb_colors.RGB_LIGHT_GREEN, 'opacity':1.0},
11: {'color':rgb_colors.RGB_DARK_GREEN, 'opacity':1.0},
12: {'color':rgb_colors.RGB_LIGHT_BLUE, 'opacity':1.0},
13: {'color':rgb_colors.RGB_DARK_BLUE, 'opacity':1.0},
14: {'color':rgb_colors.RGB_LIGHT_CYAN, 'opacity':1.0},
15: {'color':rgb_colors.RGB_DARK_CYAN, 'opacity':1.0},
16: {'color':rgb_colors.RGB_LIGHT_GRAY, 'opacity':1.0},
}

undefined_pattr = {'color':rgb_colors.RGB_BLACK, 'opacity':0.5}

user_pattr = \
{
    # This format must be follows.
    # species_id: {'color':color, 'opacity':opacity, 'name':name, 'radius':radius}
}

#-----
# Domain attributes
#-----

default_dattr = \
{
    # domain_kind: {'color':color, 'opacity':opacity}
    domain_kind_constants.SINGLE: {'color':rgb_colors.RGB_WHITE, 'opacity':0.2},
    domain_kind_constants.PAIR: {'color':rgb_colors.RGB_VIOLET, 'opacity':0.2},
    domain_kind_constants.MULTI: {'color':rgb_colors.RGB_ORANGE_RED, 'opacity':0.2},
}

undefined_dattr = {'color':rgb_colors.RGB_BLACK, 'opacity':0.2}

user_dattr = \
{
    # This format must be follows.
    # domain_kind: {'color':RGB_color, 'opacity':opacity_value}
}

#-----
# Surface object: plane
#-----
plane_surface_color = rgb_colors.RGB_WHITE

```

```
plane_surface_opacity = 1.0
# Original point of plane (This unit is world_size)
plane_surface_origin = (0, 0, 0)
# Axis 1 of plane (This unit is world_size)
plane_surface_axis_1 = (1, 0, 0)
# Axis 2 of plane (This unit is world_size)
plane_surface_axis_2 = (0, 1, 0)

plane_surface_list = []

#-----
# Filters for particles
#-----
# function of display filter by particle_id:
#   bool pfilter_pid_func( unsigned int*8 particle_id )
pfilter_pid_func = None

# function of display filter by position:
#   bool pfilter_pos_func( double pos[3] )
pfilter_pos_func = None

# function of display filter by species_id:
#   patrr pfilter_sid_func( unsigned int*8 species_id )
pfilter_sid_func = None

# particle species_id map
# This format must be follows.
#   pfilter_sid_map = { \
#       species_id_0 : display_species_id_0,
#       species_id_1 : display_species_id_1,
#       ...
#       species_id_N : display_species_id_N
#   }
#
pfilter_sid_map = None

# particle species_id mapping function:
#   unsigned int*8 display_species_id pfilter_sid_map_func( unsigned int*8 species_id )
pfilter_sid_map_func = None

#-----
# Micro fluorimetry settings
#-----
```

```
fluorimetry_display = False
fluorimetry_wave_length = 546.0e-9
fluorimetry_luminescence_color = rgb_colors.RGB_GREEN
fluorimetry_axial_voxel_number = 100
fluorimetry_background_color = rgb_colors.RGB_BLACK
fluorimetry_shadow_display = False
fluorimetry_accumulation_mode = 0 # 0:to max, 1:to sum
fluorimetry_brightness = 1.0
```

#### 12.4. visualizer.py

##### **visualizer.py**

```
"""
```

visualizer.py:

Visialization module of particles and shells  
in HDF5 file outputed from E-Cell simulator

Revision 0 (2010/1/25 released)  
First release of this module.

Revision 1 (2010/2/26 released)  
New features:  
- Blurry effect of particles is available.  
- Exception class is added for visualizer.  
Bug fixes:  
- Fixed a bug caused that newly created Settings object has history  
of the old objects.

This module uses following third-party libraries:  
- VTK (Visualization Tool Kit)  
- h5py (Python biding to HDF5 library)  
- numpy (Numerical Python)  
- FFmpeg (To make a movie from outputed snapshots)

Please install above libraries before use this module.

```
"""
```

```
import os
import sys
```

```
import tempfile
import math
import time

import h5py
import vtk
import numpy

import domain_kind_constants
import rgb_colors
import default_settings
import copy

class VisualizerError(Exception):

    "Exception class for visualizer"

    def __init__(self, info):
        self.__info = info

    def __repr__(self):
        return self.__info

    def __str__(self):
        return self.__info

class Settings(object):

    "Visualization setting class for Visualizer"

    __slots__ = []
    for x in dir(default_settings):
        # Skip private variables in default_settings.py
        if x[0] != '_': __slots__.append(x)

    def __init__(self, user_settings_dict = None):

        settings_dict = default_settings.__dict__.copy()

        if user_settings_dict != None:
```

```
        if type(user_settings_dict) != type({}):
            print 'Illegal argument type for constructor of Settings class'
            sys.exit()
        settings_dict.update(user_settings_dict)

    for key, val in settings_dict.items():
        if key[0] != '_': # Data skip for private variables in setting_dict.
            if type(val) == type({}) or type(val) == type([]):
                copy_val = copy.deepcopy(val)
            else:
                copy_val = val
            setattr(self, key, copy_val)

    def __set_data(self, key, val):
        if val != None:
            setattr(self, key, val)

    def set_image(self,
                  height = None,
                  width = None,
                  background_color = None,
                  file_name_format = None
                  ):

        self.__set_data('image_height', height)
        self.__set_data('image_width', width)
        self.__set_data('image_background_color', background_color)
        self.__set_data('image_file_name_format', file_name_format)

    def set_ffmpeg(self,
                  movie_file_name = None,
                  bin_path = None,
                  additional_options = None
                  ):

        self.__set_data('ffmpeg_movie_file_name', movie_file_name)
        self.__set_data('ffmpeg_bin_path', bin_path)
        self.__set_data('ffmpeg_additional_options', additional_options)

    def set_camera(self,
```

```
        forcal_point = None,
        base_position = None,
        azimuth = None,
        elevation = None,
        view_angle = None
    ):
        self.__set_data('camera_forcal_point', forcal_point)
        self.__set_data('camera_base_position', base_position)
        self.__set_data('camera_azimuth', azimuth)
        self.__set_data('camera_elevation', elevation)
        self.__set_data('camera_view_angle', view_angle)

def set_light(self,
               intensity = None
               ):
    self.__set_data('light_intensity', intensity)

def set_species_legend(self,
                       display = None,
                       border_display = None,
                       location = None,
                       height = None,
                       width = None,
                       offset = None
                       ):
    self.__set_data('species_legend_display', display)
    self.__set_data('species_legend_border_display', border_display)
    self.__set_data('species_legend_location', location)
    self.__set_data('species_legend_height', height)
    self.__set_data('species_legend_width', width)
    self.__set_data('species_legend_offset', offset)

def set_time_legend(self,
                    display = None,
                    border_display = None,
                    format = None,
                    location = None,
                    height = None,
                    width = None,
                    offset = None
                    ):
```

```

        ):
        self.__set_data('time_legend_display', display)
        self.__set_data('time_legend_border_display', border_display)
        self.__set_data('time_legend_format', format)
        self.__set_data('time_legend_location', location)
        self.__set_data('time_legend_height', height)
        self.__set_data('time_legend_width', width)
        self.__set_data('time_legend_offset', offset)

def set_wireframed_cube(self,
                        display = None
                        ):
    self.__set_data('wireframed_cube_display', display)

def set_axis_annotation(self,
                        display = None,
                        color = None
                        ):
    self.__set_data('axis_annotation_display', display)
    self.__set_data('axis_annotation_color', color)

def set_fluorimetry(self,
                    display = None,
                    wave_length = None,
                    luminescence_color = None,
                    axial_voxel_number = None,
                    background_color = None,
                    shadow_display = None,
                    accumulation_mode = None,
                    brightness = None
                    ):
    self.__set_data('fluorimetry_display', display)
    self.__set_data('fluorimetry_wave_length', wave_length)
    self.__set_data('fluorimetry_luminescence_color', luminescence_color)
    self.__set_data('fluorimetry_axial_voxel_number', axial_voxel_number)
    self.__set_data('fluorimetry_background_color', background_color)
    self.__set_data('fluorimetry_shadow_display', shadow_display)
    self.__set_data('fluorimetry_accumulation_mode', accumulation_mode)
    self.__set_data('fluorimetry_brightness', brightness)

```



```

def set_pattrs(self,
                species_id,
                color = None,
                opacity = None,
                name = None,
                radius = None
                ):
    if not self.user_pattrs.has_key(species_id):
        self.user_pattrs[species_id] = {}

    if color != None: self.user_pattrs[species_id]['color'] = color
    if opacity != None: self.user_pattrs[species_id]['opacity'] = opacity
    if name != None: self.user_pattrs[species_id]['name'] = name
    if radius != None: self.user_pattrs[species_id]['radius'] = radius

def set_dattrs(self,
                domain_kind,
                color = None,
                opacity = None
                ):
    if not domain_kind_constants.DOMAIN_KIND_NAME.has_key(domain_kind):
        error = 'Illegal domain_kind is set on set_dattrs function:'
        error += ' %d≠n' % domain_kind
        error += 'Please choose from 1:Single 2:Pair 3:Multi.'
        raise VisualizerError(error)
    elif not self.user_dattrs.has_key(domain_kind):
        self.user_dattrs[domain_kind] = {}

    if color != None: self.user_dattrs[domain_kind]['color'] = color
    if opacity != None: self.user_dattrs[domain_kind]['opacity'] = opacity

def add_plane_surface(self,
                      color = None,
                      opacity = None,
                      origin = None,
                      axis1 = None,
                      axis2 = None
                      ):

    color_ = self.plane_surface_color

```



```
class Visualizer(object):

    "Visualization class of e-cell simulator"

    def __init__(self,
                  HDF5_file_path_list,
                  user_settings = Settings()
                  ):

        if isinstance(user_settings, Settings):
            self.__settings = user_settings
        else:
            raise VisualizerError ¥
                ('Illegal argument type for user_settings in constructor of Visualizer')

        if type(HDF5_file_path_list) == type(""):
            HDF5_file_path_list = [HDF5_file_path_list]
        elif type(HDF5_file_path_list) != type([]):
            raise VisualizerError ¥
                ('Illegal argument type for HDF5_file_path_list in constructor of Visualizer')

        self.__HDF5_file_path_list = HDF5_file_path_list
        self.__renderer = vtk.vtkRenderer()
        self.__window = vtk.vtkRenderWindow()
        self.__axes = vtk.vtkCubeAxesActor2D()
        self.__cube = vtk.vtkActor()
        self.__species_legend = vtk.vtkLegendBoxActor()
        self.__time_legend = vtk.vtkLegendBoxActor()

        self.__plane_list = []
        for dummy in self.__settings.plane_surface_list:
            self.__plane_list.append(vtk.vtkActor())

    def __get_domain_color(self, domain_kind):
        return self.__dattrs.get ¥
            (domain_kind, self.__settings.undefined_dattrs)['color']

    def __get_domain_opacity(self, domain_kind):
        return self.__dattrs.get ¥
```

```
(domain_kind, self.__settings.undefined_dattrs)['opacity']

def __get_legend_position(self,
                            location,
                            height,
                            width,
                            offset):
    if location == 0:
        return (offset, offset)
    elif location == 1:
        return (1.0 - width - offset, offset)
    elif location == 2:
        return (offset, 1.0 - height - offset)
    elif location == 3:
        return (1.0 - width - offset, 1.0 - height - offset)
    else:
        raise VisualizerError("Illegal legend position: %d" % location)

def __create_particle_attrs(self, species_dataset):

    # Data transfer of species dataset to the dictionary

    species_array = numpy.zeros(shape = species_dataset.shape,
                                dtype = species_dataset.dtype)

    species_dataset.read_direct(species_array)
    species_dict = {}
    for x in species_array:
        species_id = x['id']
        species_dict[species_id] = {
            'name':x['name'],
            'radius':x['radius'],
            'D':x['D']
        }

    # Set species_id map

    if self.__settings.pfilter_sid_map_func:
        # Construct user map from function
        self.__species_idmap = {}
        for species_id in species_dict.keys():
```

```

        display_species_id = self.__settings.pfilter_sid_map_func(species_id)
        if (type(display_species_id) != type(0) and
            type(display_species_id) != type(0L)):
            error_info = 'Imperfect pfilter_sid_map_func¥n'
            error_info += 'Cannot find key of species_id in the map:'
            error_info += ' species_id = %d' % species_id
            raise VisualizerError(error_info)
        else:
            self.__species_idmap[species_id] = display_species_id

    elif self.__settings.pfilter_sid_map:
        # Set user map
        self.__species_idmap = self.__settings.pfilter_sid_map
        # Check the user map
        for species_id in species_dict.keys():
            if not self.__species_idmap.has_key(species_id):
                error_info = 'Imperfect pfilter_sid_map¥n'
                error_info += 'Cannot find key of species_id in the map:'
                error_info += ' species_id = %d' % species_id
                raise VisualizerError(error_info)

    else:
        # Set default map
        self.__species_idmap = {}
        for species_id in species_dict.keys():
            self.__species_idmap[species_id] = species_id

    # Delete duplicated numbers by set constructor

    tmplist = self.__species_idmap.values()
    tmplist.sort()
    self.__mapped_species_idset = set(tmplist)

    # Set particle attributes

    self.__pattrs = {}
    nondisplay_species_idset = set([])

    for species_id in self.__mapped_species_idset:

        # Get default name and radius from HDF5 data
        name = species_dict[species_id]['name']
        radius = species_dict[species_id]['radius']

```

```
D = species_dict[species_id]['D']

# Get default color and opacity from default_settings
if self.__settings.default_pattrs.has_key(species_id):
    def_pattr = self.__settings.default_pattrs[species_id]
else:
    def_pattr = self.__settings.undefined_pattrs

color = def_pattr['color']
opacity = def_pattr['opacity']

# Replace attributes by user attributes
if self.__settings.user_pattrs.has_key(species_id):
    user_pattr = self.__settings.user_pattrs[species_id]
    name = user_pattr.get('name', name)
    color = user_pattr.get('color', color)
    opacity = user_pattr.get('opacity', opacity)
    radius = user_pattr.get('radius', radius)

# Replace attributes by filter function
if self.__settings.pfilter_sid_func:
    pattr = self.__settings.pfilter_sid_func(species_id)
    if pattr == None:
        opacity = 0.0
        nondisplay_species_idset.add(species_id)
    else:
        name = pattr.get('name', name)
        color = pattr.get('color', color)
        opacity = pattr.get('opacity', opacity)
        radius = pattr.get('radius', radius)

self.__pattrs[species_id] = {
    'color':color,
    'opacity':opacity,
    'radius':radius,
    'name':name,
    'D':D
}

# Redefine for legend of particle species
self.__mapped_species_idset = ¥
self.__mapped_species_idset.difference(nondisplay_species_idset)
self.__num_particle_legend = len(self.__mapped_species_idset)
```

```
def __create_domain_attrs(self):

    self.__dattrs = self.__settings.default_dattrs
    user_dattrs = self.__settings.user_dattrs

    for domain_kind in self.__dattrs.iterkeys():
        if user_dattrs.has_key(domain_kind):
            self.__dattrs[domain_kind].update(user_dattrs[domain_kind])

def __create_environment(self, species_dataset, world_size):

    self.__world_size = world_size
    self.__create_particle_attrs(species_dataset)
    self.__create_domain_attrs()

    # Create vtk renderer

    self.__renderer.SetBackground(self.__settings.image_background_color)
    self.__renderer.SetViewport(0.0, 0.0, 1.0, 1.0)

    # Create a render window

    self.__window = vtk.vtkRenderWindow()
    self.__window.AddRenderer(self.__renderer)
    self.__window.SetSize(int(self.__settings.image_width),
                           int(self.__settings.image_height))
    self.__window.OffScreenRenderingOn() # This function cannot operate.

    # Create a camera

    camera = vtk.vtkCamera()

    camera.SetFocalPoint(self.__settings.camera_focal_point)
    camera.SetPosition(self.__settings.camera_base_position)

    camera.Azimuth(self.__settings.camera_azimuth)
    camera.Elevation(self.__settings.camera_elevation)
    camera.SetViewAngle(self.__settings.camera_view_angle)
    self.__renderer.SetActiveCamera(camera)
```

```

# Create a automatic light kit

light_kit = vtk.vtkLightKit()
light_kit.SetKeyLightIntensity(self.__settings.light_intensity)
light_kit.AddLightsToRenderer(self.__renderer)

# Create axis annotation

if self.__settings.axis_annotation_display:
    tprop = vtk.vtkTextProperty()
    tprop.SetColor(self.__settings.axis_annotation_color)
    tprop.ShadowOn()

    self.__axes.SetBounds(0.0, 1.0, 0.0, 1.0, 0.0, 1.0)
    self.__axes.SetRanges(0.0, self.__world_size,
                           0.0, self.__world_size,
                           0.0, self.__world_size)
    self.__axes.SetCamera(self.__renderer.GetActiveCamera())
    self.__axes.SetLabelFormat('%g')
    self.__axes.SetFontFactor(1.5)
    self.__axes.SetAxisTitleTextProperty(tprop)
    self.__axes.SetAxisLabelTextProperty(tprop)
    self.__axes.UseRangesOn()
    self.__axes.SetCornerOffset(0.0)

# Create a wireframed cube

if self.__settings.wireframed_cube_display:
    cube = vtk.vtkCubeSource()
    cube.SetBounds(0.0, 1.0, 0.0, 1.0, 0.0, 1.0)
    cube.SetCenter(0.5, 0.5, 0.5)

    mapper = vtk.vtkPolyDataMapper()
    mapper.SetInputConnection(cube.GetOutputPort())

    self.__cube.SetMapper(mapper)
    self.__cube.GetProperty().SetRepresentationToWireframe()

# Create species legend box

if self.__settings.species_legend_display:

    # Get number of lines

```



```
legend_line_numbers = self.__num_particle_legend ¥
                        + len(domain_kind_constants.DOMAIN_KIND_NAME)

# Create legend actor
self.__species_legend.SetNumberOfEntries(legend_line_numbers)
self.__species_legend.SetPosition ¥
    (self.__get_legend_position(self.__settings.species_legend_location,
                                self.__settings.species_legend_height,
                                self.__settings.species_legend_width,
                                self.__settings.species_legend_offset))
self.__species_legend.SetWidth(self.__settings.species_legend_width)
self.__species_legend.SetHeight(self.__settings.species_legend_height)

tprop = vtk.vtkTextProperty()
tprop.SetColor(rgb_colors.RGB_WHITE)
tprop.SetVerticalJustificationToCentered()

self.__species_legend.SetEntryTextProperty(tprop)

if self.__settings.species_legend_border_display:
    self.__species_legend.BorderOn()
else:
    self.__species_legend.BorderOff()

# Entry legend string to the actor
sphere = vtk.vtkSphereSource()

# Create legends of particle speices
count = 0
for species_id in self.__mapped_species_idset:
    self.__species_legend.SetEntryColor ¥
        (count, self.__pattrs[species_id]["color"])
    self.__species_legend.SetEntryString ¥
        (count, self.__pattrs[species_id]["name"])
    self.__species_legend.SetEntrySymbol(count, sphere.GetOutput())
    count += 1

# Create legends of shell spesies
offset = count
count = 0
for kind, name in domain_kind_constants.DOMAIN_KIND_NAME.items():
    self.__species_legend.SetEntryColor ¥
        (offset + count, self.__get_domain_color(kind))
```

```

        self.__species_legend.SetEntrySymbol ¥
            (offset + count, sphere.GetOutput())
        self.__species_legend.SetEntryString(offset + count, name)
        count += 1

# Create time legend box

if self.__settings.time_legend_display:

    # Create legend actor
    self.__time_legend.SetNumberOfEntries(1)
    self.__time_legend.SetPosition ¥
        (self.__get_legend_position(self.__settings.time_legend_location,
                                     self.__settings.time_legend_height,
                                     self.__settings.time_legend_width,
                                     self.__settings.time_legend_offset))

    self.__time_legend.SetWidth(self.__settings.time_legend_width)
    self.__time_legend.SetHeight(self.__settings.time_legend_height)

    tprop = vtk.vtkTextProperty()
    tprop.SetColor(rgb_colors.RGB_WHITE)
    tprop.SetVerticalJustificationToCentered()
    self.__time_legend.SetEntryTextProperty(tprop)

    if self.__settings.time_legend_border_display:
        self.__time_legend.BorderOn()
    else:
        self.__time_legend.BorderOff()

# Create planes

count = 0
for x in self.__settings.plane_surface_list:

    plane = vtk.vtkPlaneSource()
    plane.SetOrigin(x['origin'])
    plane.SetPoint1(x['axis1'])
    plane.SetPoint2(x['axis2'])

    mapper = vtk.vtkPolyDataMapper()
    mapper.SetInput(plane.GetOutput())

```

```
self.__plane_list[count].SetMapper(mapper)
self.__plane_list[count].GetProperty().SetColor(x['color'])
self.__plane_list[count].GetProperty().SetOpacity(x['opacity'])
count += 1

def __reset_actors(self):
    self.__renderer.RemoveAllViewProps()

def __create_particles(self, particles_dataset):

    # Data transfer from HDF5 dataset to numpy array for fast access
    particles_array = numpy.zeros(shape = particles_dataset.shape,
                                  dtype = particles_dataset.dtype)

    particles_dataset.read_direct(particles_array)

    for x in particles_array:

        particle_id = x['id']
        position = x['position']
        species_id = x['species_id']
        species_id = self.__species_idmap[species_id]

        if self.__settings.pfilter_pos_func:
            pos_filter_flag = self.__settings.pfilter_pos_func(position)
        else:
            pos_filter_flag = True

        if self.__settings.pfilter_pid_func:
            pid_filter_flag = self.__settings.pfilter_pid_func(particle_id)
        else:
            pid_filter_flag = True

        if(pos_filter_flag and
           pid_filter_flag and
           self.__pattrs[species_id]['opacity'] > 0.0):

            sphere = vtk.vtkSphereSource()
            sphere.SetRadius(self.__pattrs[species_id]['radius'] / self.__world_size)
            sphere.SetCenter(position[0] / self.__world_size,
                             position[1] / self.__world_size,
```

```
        position[2] / self.__world_size)

    mapper = vtk.vtkPolyDataMapper()
    mapper.SetInput(sphere.GetOutput())

    sphere_actor = vtk.vtkActor()
    sphere_actor.SetMapper(mapper)
    sphere_actor.GetProperty().SetColor ¥
        (self.__pattrs[species_id]['color'])
    sphere_actor.GetProperty().SetOpacity ¥
        (self.__pattrs[species_id]['opacity'])

    self.__renderer.AddActor(sphere_actor)

def __create_blurry_particles(self, particles_dataset):

    # Data transfer from HDF5 dataset to numpy array for fast access
    particles_array = numpy.zeros(shape = particles_dataset.shape,
                                  dtype = particles_dataset.dtype)

    particles_dataset.read_direct(particles_array)

    self.__renderer.SetBackground(self.__settings.fluorimetry_background_color)

    nx = ny = nz = self.__settings.fluorimetry_axial_voxel_number

    # Add points of particle
    points = vtk.vtkPoints()
    for x in particles_array:
        pos = x['position']
        points.InsertNextPoint(pos[0] / self.__world_size,
                               pos[1] / self.__world_size,
                               pos[2] / self.__world_size)

    poly_data = vtk.vtkPolyData()
    poly_data.SetPoints(points)
    poly_data.ComputeBounds()

    # Calc standard deviation of gauss distribution function
    wave_length = self.__settings.fluorimetry_wave_length
    sigma = 0.5 * wave_length / self.__world_size
```

```

# Create guassian splatter
gs = vtk.vtkGaussianSplatter()
gs.SetInput(poly_data)
gs.SetSampleDimensions(nx, ny, nz)
gs.SetRadius(sigma)
gs.SetExponentFactor(-0.5)
gs.ScalarWarpingOff()
gs.SetModelBounds(-sigma, 1.0 + sigma,
                  -sigma, 1.0 + sigma,
                  -sigma, 1.0 + sigma)

if self.__settings.fluorimetry_accumulation_mode == 0:
    gs.SetAccumulationModeToMax()
elif self.__settings.fluorimetry_accumulation_mode == 1:
    gs.SetAccumulationModeToSum()
else:
    raise VisualizerError('Illegal fluorimetry_accumulation_mode')

# Create filter for volume rendering
filter = vtk.vtkImageShiftScale()
# Scales to unsigned char
filter.SetScale(255.0 * self.__settings.fluorimetry_brightness)
filter.ClampOverflowOn()
filter.SetOutputScalarTypeToUnsignedChar()
filter.SetInputConnection(gs.GetOutputPort())

# Create volume property
opacity_tfunc = vtk.vtkPiecewiseFunction()
opacity_tfunc.AddPoint(0, 0.0)
opacity_tfunc.AddPoint(255, 1.0)

color = self.__settings.fluorimetry_luminescence_color
color_tfunc = vtk.vtkColorTransferFunction()
color_tfunc.AddRGBPoint(0, color[0], color[1], color[2])

property = vtk.vtkVolumeProperty()
property.SetColor(color_tfunc)
property.SetScalarOpacity(opacity_tfunc)
property.SetInterpolationTypeToLinear()

if self.__settings.fluorimetry_shadow_display:
    property.ShadeOn()
else:

```

```
        property.ShadeOff()

    mapper = vtk.vtkVolumeTextureMapper2D()
    mapper.SetInputConnection(filter.GetOutputPort())

    volume = vtk.vtkVolume()
    volume.SetMapper(mapper)
    volume.SetProperty(property)

    self.__renderer.AddVolume(volume)

def __create_shells(self,
                    shells_dataset,
                    domain_shell_assoc,
                    domains_dataset):

    # Data transfer from HDF5 dataset to numpy array for fast access
    shells_array = numpy.zeros(shape = shells_dataset.shape,
                              dtype = shells_dataset.dtype)

    shells_dataset.read_direct(shells_array)

    # Construct assosiaction dictionary
    domain_shell_assoc_array = numpy.zeros(shape = domain_shell_assoc.shape,
                                           dtype = domain_shell_assoc.dtype)

    domain_shell_assoc.read_direct(domain_shell_assoc_array)
    domain_shell_assoc_dict = dict(domain_shell_assoc_array)

    # Construct domains dictionary
    domains_array = numpy.zeros(shape = domains_dataset.shape,
                                dtype = domains_dataset.dtype)

    domains_dataset.read_direct(domains_array)
    domains_dict = dict(domains_array)

    # Add shell actors
    for x in shells_array:

        shell_id = x['id']

        try:
```

```
        domain_id = domain_shell_assoc_dict[shell_id]
    except KeyError:
        raise VisualizerError ¥
            ('Illegal shell_id is found in dataset of domain_shell_association!')

    try:
        domain_kind = domains_dict[domain_id]
    except KeyError:
        raise VisualizerError ¥
            ('Illegal domain_id is found in domains dataset!')

    if self.__get_domain_opacity(domain_kind) > 0.0:

        sphere = vtk.vtkSphereSource()
        sphere.SetRadius(x['radius'] / self.__world_size)
        sphere.SetCenter(x['position'][0] / self.__world_size,
                        x['position'][1] / self.__world_size,
                        x['position'][2] / self.__world_size)

        mapper = vtk.vtkPolyDataMapper()
        mapper.SetInput(sphere.GetOutput())

        sphere_actor = vtk.vtkActor()
        sphere_actor.SetMapper(mapper)
        sphere_actor.GetProperty().SetColor ¥
            (self.__get_domain_color(domain_kind))
        sphere_actor.GetProperty().SetRepresentationToWireframe()
        sphere_actor.GetProperty().SetOpacity ¥
            (self.__get_domain_opacity(domain_kind))

        self.__renderer.AddActor(sphere_actor)

    def __activate_environment(self, time):

        if self.__settings.axis_annotation_display:
            self.__renderer.AddViewProp(self.__axes)

        if self.__settings.wireframed_cube_display:
            self.__renderer.AddActor(self.__cube)

        if self.__settings.time_legend_display:
            self.__time_legend.SetEntryString ¥
```

```
        (0, self.__settings.time_legend_format % time)
        self.__renderer.AddActor(self.__time_legend)

    if(self.__settings.species_legend_display and ¥
        not self.__settings.fluorimetry_display) :
        self.__renderer.AddActor(self.__species_legend)

    for x in self.__plane_list:
        self.__renderer.AddActor(x)

def __output_snapshot(self, image_file_name):

    "Output snapshot to image file"

    image_file_type = os.path.splitext(image_file_name)[1]

    # Remove existing image file
    if os.path.exists(image_file_name):
        if os.path.isfile(image_file_name):
            os.remove(image_file_name)
        else:
            raise VisualizerError ¥
                ('Cannot overwrite image file: ' + image_file_name)

    w2i = vtk.vtkWindowToImageFilter()
    w2i.SetInput(self.__window)

    if image_file_type == '.bmp':
        writer = vtk.vtkBMPWriter()
    elif image_file_type == '.jpg':
        writer = vtk.vtkJPEGWriter()
    elif image_file_type == '.png':
        writer = vtk.vtkPNGWriter()
    elif image_file_type == '.tif':
        writer = vtk.vtkTIFFWriter()
    else:
        error_info = 'Illegal image-file type: ' + image_file_type + '¥n'
        error_info += 'Please choose from "bmp","jpg","png","tif".'
        raise VisualizerError(error_info)

    writer.SetInput(w2i.GetOutput())
    writer.SetFileName(image_file_name)
```



```

writer.Write()

def output_snapshots(self, image_file_dir):

    "Output snapshots from HDF5 dataset"

    # Create image file folder
    if not os.path.exists(image_file_dir):
        os.makedirs(image_file_dir)

    # Check empty path list
    if len(self.__HDF5_file_path_list) == 0:
        raise VisualizerError("Empty HDF5_file_path_list.¥n Please set it.")

    # Check accessable to the path list
    for path in self.__HDF5_file_path_list:
        if(not os.path.exists(path) or
           not os.path.isfile(path) or
           not os.access(path, os.R_OK)):
            raise VisualizerError('Cannot accsess HDF5 file: ' + path)

    # Get time seugence in data group of HDF5 files
    #   and sort the loading order

    particles_time_sequence = []
    shells_time_sequence = []

    for HDF5_file_path in self.__HDF5_file_path_list:

        HDF5_file = h5py.File(HDF5_file_path, 'r')
        data_group = HDF5_file['data']

        for time_group_name in data_group:
            time_group = data_group[time_group_name]
            time = time_group.attrs['t']
            elem = (time, HDF5_file_path, time_group_name)
            if 'particles' in time_group.keys():
                particles_time_sequence.append(elem)
            if 'shells' in time_group.keys():
                shells_time_sequence.append(elem)

        HDF5_file.close()

```

```
if len(particles_time_sequence) == 0:
    raise VisualizerError ¥
    ('Cannot find particles dataset in HDF5_file_path_list: ' ¥
     + self.__HDF5_file_path_list)

# Sort ascending time order
particles_time_sequence.sort(lambda a, b:cmp(a[0], b[0]))
# Sort descending time order
shells_time_sequence.sort(lambda a, b:-cmp(a[0], b[0]))

# Visualize by the obtained time sequence

time_count = 0
snapshot_file_list = []

for (time, HDF5_file_name, time_group_name) in particles_time_sequence:

    HDF5_file = h5py.File(HDF5_file_name, 'r')

    data_group = HDF5_file['data']
    species_dataset = HDF5_file['species']

    world_size = data_group.attrs['world_size']
    time_group = data_group[time_group_name]

    # Create environment at first time
    if time_count == 0:
        self.__create_environment(species_dataset, world_size)

    self.__reset_actors()
    self.__activate_environment(time)

    if self.__settings.fluorimetry_display:
        self.__create_blurry_particles(time_group['particles'])
    else:
        self.__create_particles(time_group['particles'])

    for (shells_time,
         shells_HDF5_file_name,
         shells_time_group_name) in shells_time_sequence:

        if time >= shells_time: # Backward time search
```

```
        open_flag = False
        if os.path.samefile(shells_HDF5_file_name, HDF5_file_name):
            shells_HDF5_file = HDF5_file
        else:
            shells_HDF5_file = h5py.File(shells_HDF5_file_name, 'r')
            open_flag = True

        shells_data_group = shells_HDF5_file['data']
        shells_time_group = shells_data_group[shells_time_group_name]
        shells_dataset = shells_time_group['shells']

        domain_shell_assoc = shells_time_group['domain_shell_association']
        domain_dataset = shells_time_group['domains']

        self.__create_shells(shells_dataset,
                             domain_shell_assoc,
                             domain_dataset)

        if open_flag:
            shells_HDF5_file.close()
        break

    image_file_name = ¥
        os.path.join(image_file_dir,
                     self.__settings.image_file_name_format % time_count)

    self.__output_snapshot(image_file_name)
    snapshot_file_list.append(image_file_name)

    HDF5_file.close()

    time_count += 1

    return snapshot_file_list

def make_movie(self,
               image_file_dir,
               movie_file_dir):

    """
    Make a movie by FFmpeg
    Please install FFmpeg (http://ffmpeg.org/) from the download site
```

```

        before use this function.
        """

        input_image_filename = ¥
            os.path.join(image_file_dir,
                          self.__settings.image_file_name_format)

        output_movie_filename = ¥
            os.path.join(movie_file_dir,
                          self.__settings.ffmpeg_movie_file_name)

        # Create movie file folder
        if not os.path.exists(movie_file_dir):
            os.makedirs(movie_file_dir)

        # Remove existing movie file
        if os.path.exists(output_movie_filename):
            if os.path.isfile(output_movie_filename):
                os.remove(output_movie_filename)
            else:
                raise VisualizerError ¥
                    ('Cannot overwrite movie file: ' + output_movie_filename)

        # Set FFMPEG options
        options = self.__settings.ffmpeg_additional_options ¥
            + '-i "' + input_image_filename + '" ' ¥
            + output_movie_filename

        if self.__settings.ffmpeg_bin_path:
            if (os.path.isfile(self.__settings.ffmpeg_bin_path) and
                os.access(self.__settings.ffmpeg_bin_path, os.X_OK)):
                os.system(self.__settings.ffmpeg_bin_path + ' ' + options)
                return
        else:
            for dir in os.environ['PATH'].split(os.pathsep):
                search_path = os.path.join(dir, 'ffmpeg')
                if os.access(search_path, os.X_OK):
                    os.system(search_path + ' ' + options)
                    return

        raise VisualizerError ¥
            ('Cannot access ffmpeg. Please set ffmpeg_bin_path correctly.')

```

```
def output_movie(self, movie_file_dir, image_tmp_root = None):

    """
    Output movie to movie_file_dir
    This function creates temporal image files to output the movie.
    These temporal files and directory are removed after the output.
    """

    if image_tmp_root == None:
        image_tmp_dir = tempfile.mkdtemp(dir = os.getcwd())
    else:
        image_tmp_dir = tempfile.mkdtemp(dir = image_tmp_root)

    snapshot_file_list = self.output_snapshots(image_tmp_dir)
    self.make_movie(image_tmp_dir, movie_file_dir)

    # Remove snapshots on temporary directory
    for snapshot_file in snapshot_file_list:
        if(os.path.exists(snapshot_file) and
           os.path.isfile(snapshot_file)):
            os.remove(snapshot_file)

    # Remove temporary directory if it is empty.
    if len(os.listdir(image_tmp_dir)) == 0:
        os.rmdir(image_tmp_dir)
```

## 12.5. logger.py

### logger.py

```
import os
import re
#import logging
import numpy

import logging
import h5py # added by sakurai@advancesoft.jp
from egfrd import Single, Pair, Multi # added by sakurai@advancesoft.jp

INF = numpy.inf
```

```
log = logging.getLogger('ecell')

class Logger:

    def __init__(self, sim, logname = 'log', directory = 'data',
                 comment = ""):

        self.sim = sim

        self.logname = logname

        self.fileCounter = 0

        self.directory = directory
        try:
            os.mkdir(directory)
        except:
            pass

        self.particleOutInterval = INF

        self.lastTime = 0.0
        self.nextTime = INF

        self.particleOutPattern = re.compile("")
        self.prepareTimecourseFile(comment)
        self.writeTimecourse()

        # Added by sakurai@advancesoft.jp
        self.HDF5_filename = self.logname + '.hdf5'
        self.HDF5_path = self.directory + os.sep + self.HDF5_filename
        # HDF5 file must be removed before logParticles
        if os.path.exists(self.HDF5_path) and os.path.isfile(self.HDF5_path):
            os.remove(self.HDF5_path)

    def setInterval(self, interval):
        self.interval = interval

    def setParticleOutPattern(self, pattern):
        self.particleOutPattern = re.compile(pattern)
```

```

def getParticleOutPattern(self):
    return self.particleOutPattern.pattern

def setParticleOutInterval(self, interval):
    self.particleOutInterval = interval
    self.lastTime = self.sim.t
    self.nextTime = self.lastTime + self.particleOutInterval

def prepareTimecourseFile(self, comment):

    self.timecourseFilename = self.logname + '_tc' + '.dat'
    self.timecourseFile = open(self.directory + os.sep + \
                               self.timecourseFilename, 'w')
    self.writeTimecourseComment(comment)

    speciesNameList = "\"" + \
        "\", \"".join(str(i) for i in self.sim.speciesList.keys()) + "\""
    columns = "[" + speciesNameList + "]"
    self.writeTimecourseComment('@ columns= ' + columns)

def writeTimecourseComment(self, s):
    self.timecourseFile.write('#' + s + '\n')

def writeTimecourse(self):
    data = [
        ]

    self.timecourseFile.write('%g' % self.sim.t + '\t')
    self.timecourseFile.write('\t'.join(
        str(len(self.sim.getParticlePool(i.type.id))) \
        for i in self.sim.getSpecies()) + '\n')
    self.timecourseFile.flush()

def writeParticles(self):
    filename = self.logname + '_' + \
        str(self.fileCounter).zfill(4) + '.dat'

    file = open(self.directory + os.sep + filename, 'w')

    file.write('#@ name = \"%s\" % str(self.logname))
    file.write('#@ count = %d % int(self.fileCounter))
    file.write('#@ t = %s % \"%g\" % self.sim.t)

```

```
file.write('#@ worldSize = %f\n' % float(self.sim.getWorldSize()))
file.write('#-----\n')

for sid in self.sim.speciesList.keys():
    pid_list = self.sim.particlePool[ sid ]
    for i in pid_list:
        particle = self.sim.particleMatrix[i]
        species = self.sim.speciesList[ sid ]
        file.write('%s\t%20.14g %20.14g %20.14g %.15g\n' %
                    (species.id, particle.position[0], particle.position[1], particle.position[2],
species.radius))

    file.write('\n')

file.close()

self.fileCounter += 1

def writeParticlesByHDF5(self):

    "This function was created by sakurai@advancesoft.jp"

    HDF5_file = h5py.File(self.HDF5_path)

    if HDF5_file.get('data', None) == None:
        data_group = HDF5_file.create_group('data')
        data_group.attrs['world_size'] = self.sim.getWorldSize()

    # Create species dataset on top level of HDF5 hierarchy

    num_species = len(self.sim.speciesList)
    species_schema = \
        [
            ('id', 'u8'),
            ('name', 'S32'),
            ('radius', 'f8'),
            ('D', 'f8'), # diffusion coefficient
        ]

    species_dset = HDF5_file.create_dataset('species', (num_species,), species_schema)

    count = 0
    for species in self.sim.speciesList.itervalues():
```



```
        species_dset[count] = (species.type.id.serial,
                                species.type['id'],
                                species.radius,
                                species.D)

        count += 1

    else:
        data_group = HDF5_file['data']

    time_group = data_group.require_group(unicode(self.nextTime))
    time_group.attrs['t'] = self.nextTime

    # Create particles dataset on the time group

    num_particles = 0
    for sid in self.sim.speciesList.keys():
        pid_list = self.sim.particlePool[ sid ]
        num_particles += len(pid_list)

    particles_schema = \
        [
            ('id', 'u8',),
            ('species_id', 'u8',),
            ('position', 'f8', (3,))
        ]

    x = numpy.zeros((num_particles,),
                    dtype = numpy.dtype(particles_schema))

    count = 0
    for sid, pid_set in self.sim.particlePool.iteritems():
        for pid in pid_set:
            particle = self.sim.particleMatrix[pid]
            species = self.sim.speciesList[sid]
            x['id'][count] = pid.serial
            x['species_id'][count] = sid.serial
            x['position'][count] = particle.position
            count += 1

    dummy = time_group.create_dataset('particles', data = x)

    HDF5_file.close()
```

```
def writeDomainsByHDF5(self):

    "This function was created by sakurai@advancesoft.jp"

    HDF5_file = h5py.File(self.HDF5_path, 'a')

    # Require data group
    data_group = HDF5_file.require_group('data')
    data_group.attrs['world_size'] = self.sim.getWorldSize()

    # Require time group
    time_group = data_group.require_group(unicode(self.nextTime))
    time_group.attrs['t'] = self.nextTime

    # Create shell dataset on the time group

    shells_schema = \
        [
            ('id', 'u8'),
            ('radius', 'f8'),
            ('position', 'f8', (3,)),
        ]

    num_shells = 0
    for domain in self.sim.domains.itervalues():
        num_shells += len(domain.shell_list)

    x = numpy.zeros((num_shells,), dtype = numpy.dtype(shells_schema))

    count = 0
    for did, domain in self.sim.domains.iteritems():
        shell_list = domain.shell_list
        for shell_id, shell in shell_list:
            x['id'][count] = shell_id.serial
            x['radius'][count] = shell.radius
            x['position'][count] = shell.position
            count += 1

    dummy = time_group.create_dataset('shells', data = x)

    # Create shell particle association dataset on the time group

    num_assocs = 0
```

```

for domain in self.sim.domains.itervalues():
    if isinstance(domain, Single):
        num_assocs += len(domain.shell_list)
    elif isinstance(domain, Pair):
        num_assocs += 2 * len(domain.shell_list)
    elif isinstance(domain, Multi):
        assert getattr(domain, 'pid_shell_id_map', None), 'Cannot access pid_shell_id_map'
        num_assocs += len(domain.pid_shell_id_map)

shell_particle_association_schema = \
    [
        ('shell_id', 'u8'),
        ('particle_id', 'u8'),
    ]

dtype_obj = numpy.dtype(shell_particle_association_schema)
x = numpy.zeros((num_assocs,), dtype = dtype_obj)

count = 0
for did, domain in self.sim.domains.iteritems():

    if(isinstance(domain, Single) or
       isinstance(domain, Pair)):

        pid_particle_pair_list = []
        if isinstance(domain, Single):
            pid_particle_pair_list = [domain.pid_particle_pair]
        elif isinstance(domain, Pair):
            pid_particle_pair_list = [domain.single1.pid_particle_pair,
                                     domain.single2.pid_particle_pair]

        for pid, particle in pid_particle_pair_list:
            for shell_id, shell in domain.shell_list:
                x['shell_id'][count] = shell_id.serial
                x['particle_id'][count] = pid.serial
                count += 1

    else: # for Multi
        assert getattr(domain, 'pid_shell_id_map', None), 'Cannot access pid_shell_id_map'
        for pid, shell_id in domain.pid_shell_id_map.iteritems():
            x['shell_id'][count] = shell_id.serial
            x['particle_id'][count] = pid.serial
            count += 1

```

```

dummy = time_group.create_dataset('shell_particle_association', data = x)

# Create domain_shell_association dataset on the time group

domain_shell_association_schema = \
    [
        ('shell_id', 'u8'),
        ('domain_id', 'u8'),
    ]

dtype_obj = numpy.dtype(domain_shell_association_schema)
x = numpy.zeros((num_shells,), dtype = dtype_obj)

count = 0
for did, domain in self.sim.domains.iteritems():
    shell_list = domain.shell_list
    for shell_id, shell in shell_list:
        x['shell_id'][count] = shell_id.serial
        x['domain_id'][count] = did.serial
        count += 1

dummy = time_group.create_dataset('domain_shell_association', data = x)

# Create domain dataset on the time group

domains_schema = \
    [
        ('id', 'u8'),
        ('kind', 'u4'),
    ]

num_domains = len(self.sim.domains.keys())

dtype_obj = numpy.dtype(domains_schema)
x = numpy.zeros((num_domains,), dtype = dtype_obj)

count = 0
for did, domain in self.sim.domains.iteritems():
    x['id'][count] = did.serial
    if isinstance(domain, Single):
        x['kind'][count] = 1
    elif isinstance(domain, Pair):

```

```
        x['kind'][count] = 2
    else: # must be Multi
        x['kind'][count] = 3
    count += 1

    dummy = time_group.create_dataset('domains', data = x)

    HDF5_file.close()

def log(self):

    self.logTimeCourse()
    self.logParticles()

def logTimeCourse(self):

    if self.sim.lastReaction:
        self.writeTimecourse()

def logParticles(self):
    sim = self.sim
    if self.nextTime <= sim.t + sim.dt:
        #if __debug__: log.info( 'log %g' % self.nextTime )

        sim.stop(self.nextTime)
#        self.writeParticles()
        self.writeParticlesByHDF5() # added by sakurai@advancesoft.jp
        self.writeDomainsByHDF5()  # added by sakurai@advancesoft.jp

        self.nextTime += self.particleOutInterval
```