eQ .json Data Specification

james winkle

January 2020

1 Introduction

The eQ software is an agent-based model (ABM) that simulates experimental synthetic biological applications of rod-shaped bacterial cells growing in microfluidic traps. The software is written in C++ and was inspired by the ABM `gro` (Eric Klavins lab, `http://depts.washington.edu/soslab/gro/`), which used a 2D physics engine known as *Chipmunk* (`http://chipmunk-physics.net/`) to model cells growing in quasi-2D environments. eQ also uses Chipmunk 2D; however, the cell growth model was completely re-written (see *Physical Biology* **14**, 5). Additionally, eQ integrates the open-source finite-element software Fenics (`https://fenicsproject.org/`) to efficiently couple inter-cellular signaling with the ABM cellular circuits. Fenics computations are done in parallel using MPI: one processing element is assigned for the physics engine, and additional elements are used for the diffusion solver, one for each diffusible molecule in the simulation. Thus, each simulation time step uses a dedicated computing core for the rate-limiting computations in true parallel.

eQ records data in the JSON format and the format of the data is described here. The C++ JSON library provided by N.Lohmann (`https://github.com/nlohmann/json`) is used in eQ for convenient direct recording of data. Once recorded, the data (a .json text file) can easily be loaded and manipulated in matlab using built-in JSON functions provided by matlab.

JSON is a lightweight ASCII text format with key-value data structures. An `object` is an unordered list of key-value pairs: a string-based `key` and a `value`, where each `value` is a valid JSON data type: a string, a number, another layer `object`, an `array`, a boolean or a NULL object. An `array` is an ordered listing of values, all of the same type, which may itself be a structured `object`. Objects in a JSON file are not considered ordered unless they are part of an `array`. The data of an `object` is always referred to by its `key`.

2 JSON File Details

An eQ simulation has top-level data description, as in the following example:

```
jfile["jsonSimVersion"] = "v00_03";
jfile["timeSinceEpoch"] = timeStamp;
jfile["nodeID"]         = nodeID;
jfile["simNumber"]      = simNumber;
jfile["frames"] = jframes;
jfile["parameters"] = eQ::parameters;
```

The key `frames` refers to an array of frame data and is a snapshot of the simulation at a particular timestep and at a framerate set in the simulation. A frame has a record of all the cell divisions that occurred since the last frame and a list of cell data (position, angle, length, etc...) for cells present at that snapshot time only. This data is structured as follows:

```
thisFrame["simTime"] = sim->simTime;
thisFrame["divisions"] = sim->ABM->divisionList;
thisFrame["cells"] = jcells;
```

Drilling down one level deeper, the `divisions` key references an array where each entry is a division event structure which is recorded as in the following (source) C++ code:

```
void eQabm::recordDivisionEvent
    (double timeStamp,
    std::shared_ptr<eColi> cell,
    std::shared_ptr<eColi> daughter)
{
    //initial cell at t=0: parent is -ID (length 0)
    if(nullptr == cell)
    {
        divisionList.push_back(
                    std::make_tuple(
                            timeStamp,
                            daughter->parentID,
                            0.0,//parent length=0
                            daughter->getCellID(),
                            daughter->getLengthMicrons()
                    )
        );
    }
    else
    {
        divisionList.push_back(
                    std::make_tuple(
                            timeStamp,
                            cell->getCellID(),
                            cell->getLengthMicrons(),
                            daughter->getCellID(),
                            daughter->getLengthMicrons()
                    )
        );
    }
}
```

Thus, initial seed cell `ID` will have a parent identification number set to `-ID`, whereas newly divided cells will have one new daughter cell `ID`, while also recording the `ID` of the parent (which is kept for the other daughter cell). This data structure allows one to build a lineage tree and calculate statistics, for example for average birth length, time to division, etc...

Further, the `cells` key refers to the array of cell data structures, for those cells present in the simulation at the timestep recorded (note cells are removed from the simulation when they exit the trap boundary, but this event is not recorded).

A cell structure definitely contains:

```
std::vector<double> cellData ={
    cell->getCenter_x()
    , cell->getCenter_y()
    , cell->getAngle()
    , cell->getLengthMicrons()
    , cell->getSpringCompression()
};
thisCell["i"] =  cell->getCellID();
thisCell["d"] =  cellData;
thisCell["p"] =  (eQ::strainType::ACTIVATOR == cell->Params.strainType) ? 0 : 1;
```

3 Matlab import and reading

The data can be read into matlab, for example using the follwing code:

```
basePath = "~/Dropbox/xps/eQ/build/";
% LOAD FILE, READ DIRECTLY JSON FILE VIA MATLAB INTERFACE
[file path] = uigetfile(sprintf("%s*.json", basePath), 'Open File');
dataIn = fileread(fullfile(path,file));
fprintf('decoding JSON file...%s\n', file);
jsonData =  jsondecode(dataIn);
```

Now, the variable `jsonData` has the entire .json file that is accessible by key-value. An example is the following, which relates to the C++ code that writes this data:

```
% EXTRACT META DATA, FRAMED DATA, AND CELL DIVISION DATA STRUCTURES:
lengthCellDataField = length(jsonData.frames(1).cells(1).d);
maxValues = zeros(1, lengthCellDataField);
minValues = zeros(1, lengthCellDataField);

% CELL DATA ARRAY KEY:
CENTER_X = 1;
CENTER_Y = 2;
ANGLE = 3;
LENGTH = 4;
COMP = 5;

simNumber    = jsonData.simNumber
simdt        = jsonData.parameters.dt;
timeStamp = num2str(jsonData.timeSinceEpoch);
trapHeight   = jsonData.parameters.simulationTrapHeightMicrons;
trapWidth    = jsonData.parameters.simulationTrapWidthMicrons;

% DETERMINE NUMBER OF CELLS TO TRACK:
numFrames = length(jsonData.frames);
numCells = zeros(numFrames,1);
maxCellNumber = 0;
for i = 1:numFrames
        numCells(i) = length(jsonData.frames(i).cells);
    for j=1:numCells(i)
        thisCellNumber = jsonData.frames(i).cells(j).i;
        if(thisCellNumber > maxCellNumber)
            maxCellNumber = thisCellNumber;
        end
    end
end
fprintf("Frames scanned; maximum cell ID=%d\n", maxCellNumber);
```

The `parameters` object is an array of key-value pairs of the parameters of the simulation. An example print of this data from matlab is as follows:

```
>> jsonData.parameters

ans =

  struct with fields:

        AnisotropicDiffusion_Axial: 1
    AnisotropicDiffusion_Transverse: 1
                            D_HSL: [2 1 double]
            K50_correlationScale: 0
       MODULUS_TIME_AVERAGE_MINS: 1
          aspectRatioThresholdHSL: 220
                       boundaries: [1 1 struct]
                     boundaryType: 'DIRICHLET_UPDATE'
                     cellInitType: 'AB_HALF'
        channelLengthMicronsLeft: 200
       channelLengthMicronsRight: 200
    channelSolverNumberIterations: 1
         defaultAspectRatioFactor: 1
                 diffusionScaling: 0.0400
              divisionNoiseScale: 0.0500
                              dt: 0.0500
                      gammaT_C14: 1.3121
                       gammaT_C4: 2.0584
           hslProductionRate_C14: 1.3121e+03
            hslProductionRate_C4: 2.0584e+03
                    hslSignaling: 1
                   lengthScaling: 5
          membraneDiffusionRates: [2 1 double]
                     modelType: 'OFF_LATTICE_ABM'
          mutantAspectRatioScale: 0.8000
              nodesPerMicronData: 1
         nodesPerMicronSignaling: 2
                 numberSeedCells: 1000
           openWalledDirichlet0: 1
          physicalDiffusionRates: [1 1 struct]
    physicalTrapHeight_Y_Microns: 100
     physicalTrapWidth_X_Microns: 500
        promoterDelayTimeMinutes: 8
                recordingInterval: 10
                     rhoe_by_rhoi: 0.2732
                          simType: 'ASPECTRATIO_INVASION'
    simulationTrapHeightMicrons: 20
     simulationTrapWidthMicrons: 100
      trapChannelLinearFlowRate: 300
                        trapType: 'NOWALLED'
```

4 Summary of the .json file object structure

```
{
    "jsonSimVersion"     : "",
    "timeSinceEpoch"     : #,
    "nodeID"             : #,
    "simNumber"          : #,
    "frames"             :
        [
            {
                "simTime"    : #,
                "divisions" :
                    [
                        [timeStamp, parentID, parentLength, daughterID, daughterLength],
                        ...
                    ],
                "cells"      :
                    [
                        {
                            "i"      : <cell id>,
                            "d"      : [x, y, angle, length, compression],
                            "p"      : <cell type 0 or 1>
                        },
                        ...
                    ]
            },
            ...
        ],
    "parameters"         :
        {
            "key"     : <value>,
            ...
        }
}
```