Supplementary Material for

SimELIT: A Novel GUI-based Comprehensive Ion Trajectory Simulation Software for Mass Spectrometry

Cameron Giberson[1], Rajesh K Singh[3], Jaehun Chun[2], Jason Zhong[3], Yehia M. Ibrahim[1], Gregory K Schenter[2], Joon-Yong Lee[1*], Sandilya VB Garimella[1*]

[1]Earth and Biological Sciences Directorate, Pacific Northwest National Laboratory, Richland WA

[2]Physical and Computational Sciences Directorate, Pacific Northwest National Laboratory, Richland WA

[3]Energy and Environment Directorate, Pacific Northwest National Laboratory, Richland WA

*corresponding author(s): sandilya.garimella@pnnl.gov, joonyong.lee@pnnl.gov

A1. Implementation Details

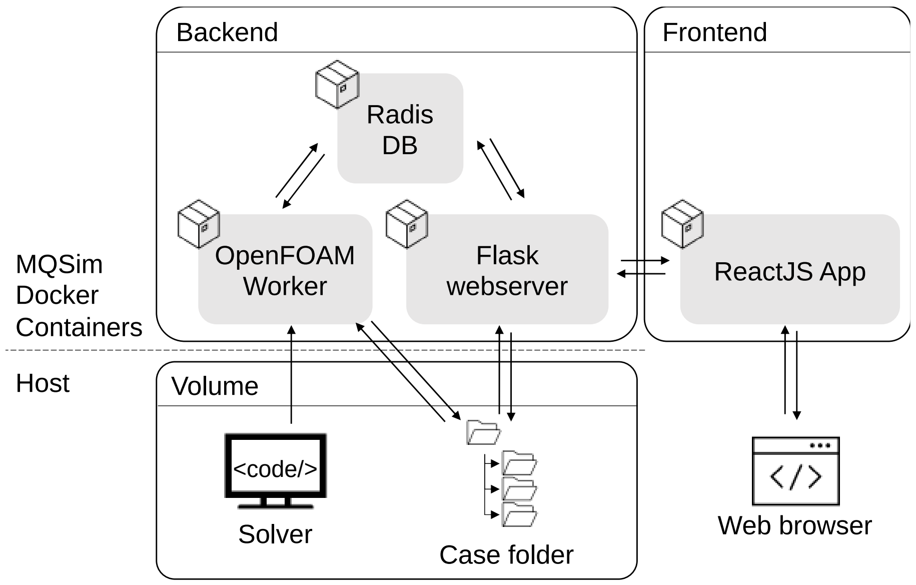


Figure S1. Overall architecture of the SimELIT GUI tool. SimELIT consists of four Docker containers: Flask webserver, Radis DB for message broker, Celery worker for OpenFOAM, and React-based frontend app. SimELIT also offers the customized solvers and case folders for users to perform ion trajectory simulation.

SimELIT is divided into frontend and backend components, with Docker providing a way to logically containerize related elements (Figure S1). The frontend web page was developed using React.js, one of the most popular Javascript libraries, due to its consistent and seamless performance with virtual DOM. The reusability and accessibility of React components will facilitate collaboration with others in the community. Material-UI, a React UI framework, was used to customize the web interface of SimELIT. Socket.IO, a JavaScript library for establishing the web socket connection, is also used for real-time, bidirectional, and event-based communication between the frontend clients and the backend containers. All control of the backend functionality is managed through HTTP requests to RESTful API calls using Python and Flask.

For an OpenFOAM simulation, two main components are required: 1) a solver and 2) a case. Functionality for control of solver execution starting and stopping, as well are directly manipulating case files is supported. The worker container is used to manage task queues so that a long-running task (e.g., an ion simulation) is asynchronously executed outside the HTTP request-response cycle from the Flask process. We employ Celery, a Python package, for task management in the worker container. Starting a run creates a new process for our custom solver to begin its execution on a worker instance independently established as a separate Docker container. Event calls in the worker instance are emitted to write the log messages to the Redis DB instance and then the Flask backend container listens to those messages and transmits back to the frontend clients.

In SimELIT, OpenFOAM's case folder structure for each custom solver will be provided by default. In general, all case dictionaries consist of multiple plain text files in directories named /0, /system, and /constant. Initial and boundary conditions are usually stored in directory /0 and the number in directory name represents the physical time layer of the solutions calculated during the simulation. /system contains text files to set up parameters associated with the solution procedure. For example, the controlDict file contains simulation control parameters including a target solver name (i.e., application), start/end time, time step, and parameters for data output. /constant contains the case mesh information in /constant/polyMesh/ and files for physical properties. The application will only consider these folders and report their contents. Case files are dynamically discoverable allowing for variations in solver dictionaries; one only must place various solvers and case folders in a known location for the application to fully discover and associate the two. The full contents of files are reported to the UI. The API exposes functionality for reporting on selected case folders. From a specific folder and specific file, the full contents of a file can be queried. The API can also accept the edited dictionary that is then saved to the file system. PyFoam was initially sought as a way of parsing and manipulating files, but it was determined that a full file editable by the user would present a direct and simplified approach to mutating files.

When executing, a solver will report meaningful information to the user. In SimELIT, because the solver is executing in a background process, the information typically written to the console from the running solver is captured from the output stream and then published on a WebSocket. The frontend will listen for events published on the WebSocket and display this information in a terminal-like window embedded in the webpage. Because the data is continually written irrespective of the UI being loaded, the UI can return to reporting on the output after the browser tab is closed.

A2. Step-by-Step Tutorials

Overview

OpenSIM is cross-platform React based GUI for managing OpenFOAM v7 running in Ubuntu Bionic operating system images. The intent of the application is to simplify the editing and managing of OpenFOAM cases by way of a lightweight graphical user interface for editing and simulation control. The application has been containerized and users are required to have **Docker** and **Docker-Compose** installed on their system to run the application; any system that can run Docker and a browser application can utilize SimELIT's solvers.

Installation

SimELIT has no installer. Instead, the application is distributed as a containerized collection of source code and Docker files that constitute the application. Docker and Docker-compose will need to be present on the user's systems prior to building and running the application. Docker can be downloaded at https://www.docker.com/. Docker-compose is used to build and manage running of the application. Windows and Mac desktop installations of Docker will automatically install Docker-compose on the user's system. Linux users will need to take additional steps to install

Docker-compose and detailed instructions can be found at https://docs.docker.com/compose/install/.

The application itself can be downloaded from https://github.com/PNNL-Comp-Mass-Spec/SimELIT. Once downloaded, the application should be moved to its own director if not already in one. The topmost level of the SimELIT directory contains a single docker-compose.yml (**Figure 1**) file needed for Docker-compose to create the Docker images that make up the discrete elements of the application (e.g., the web server, backend application code, etc.).

Data Management

The /data folder in the topmost directory contains three solvers and associated case folders. Solver folders contain source code that will be automatically built and installed in the image by OpenFOAMS's make utility. All solvers in the /data directory are automatically recompiled during the building process and are stored in the memory space of the Docker container upon completion. Because of this, edits to the solver's source code require a rebuilding of the application. All solvers and case folders in this directory are discovered by SimELIT and will be displayed to the user in the browser window on start-up. Any additional solvers or case folders must be placed in this directory before the application is built for them to be discovered by the application.

Building

To build the application, in a terminal, navigate to the topmost directory containing the **docker-compose.yml** file **Figure S2**. In the terminal window, enter the command *docker-compose build*. This will initiate the build process and the application will begin downloading necessary resources and generating the application images. Alternatively, a Makefile is included for systems that have Make installed; the command *make build* will execute the same docker-compose build instruction but is offered as a simplified alternative. Solvers located in the /data directory will be automatically copied into the image's memory space during this step. If ever this code needs to change, this build step will need to be repeated.

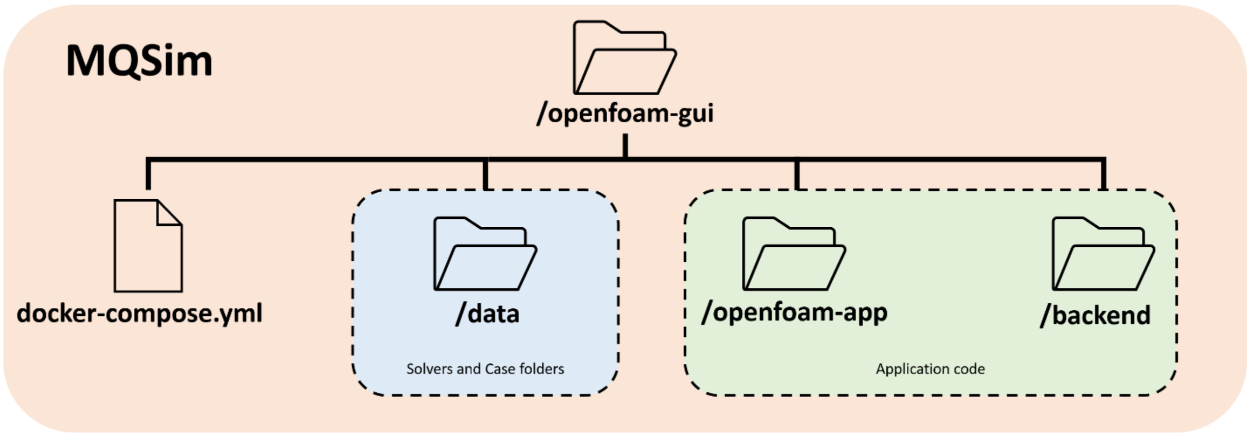


*Figure S2. The SimELIT project directory. The application when downloaded or cloned from GitHub will be under the **openfoam-gui** directoy. The docker compose 'build' and 'up' commands should be executed in this topmost directory.*

Running

Once the build process has completed, the application is ready to be run. In the same terminal window and directory location, enter the command *docker-compose up* (or *Make run*, if Make is present on the system). The issuance of this command starts the various containers that comprise the application All solvers copied into the application container will be compiled and installed automatically.

To navigate to the GUI, open a browser tab and navigate to **localhost:3000** to view the SimELIT GUI. The application will remain running if the browser tab or window is closed, so long as the Docker service and containers are running.

Editing Case Files

Figure S3 shows the case folder and solver selection page with both a solver and a case folder selected. All solvers and case folders discovered by SimELIT will be displayed in their respective drop-down menus. A console window is also displayed to the users (on all pages) that displays output from the terminal and from log messages printed during the simulation process. A simple input line allows users more familiar with Linux or OpenFOAM to issue commands directly to the running Ubuntu container or invoke OpenFOAM utilities (see **Figure S3**). Selecting a case folder allows for case files to be edited in the **EDIT CASES** page.

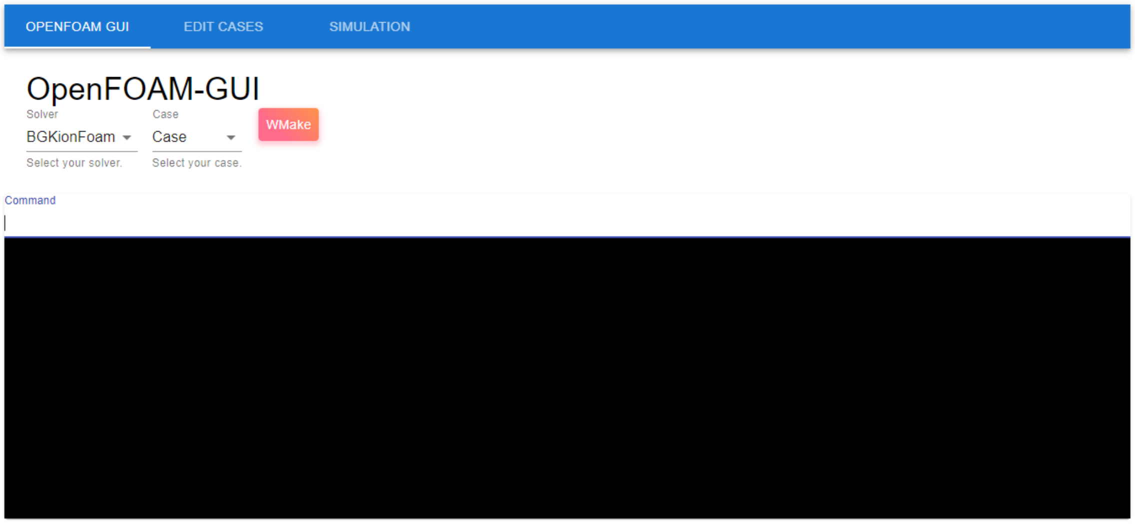
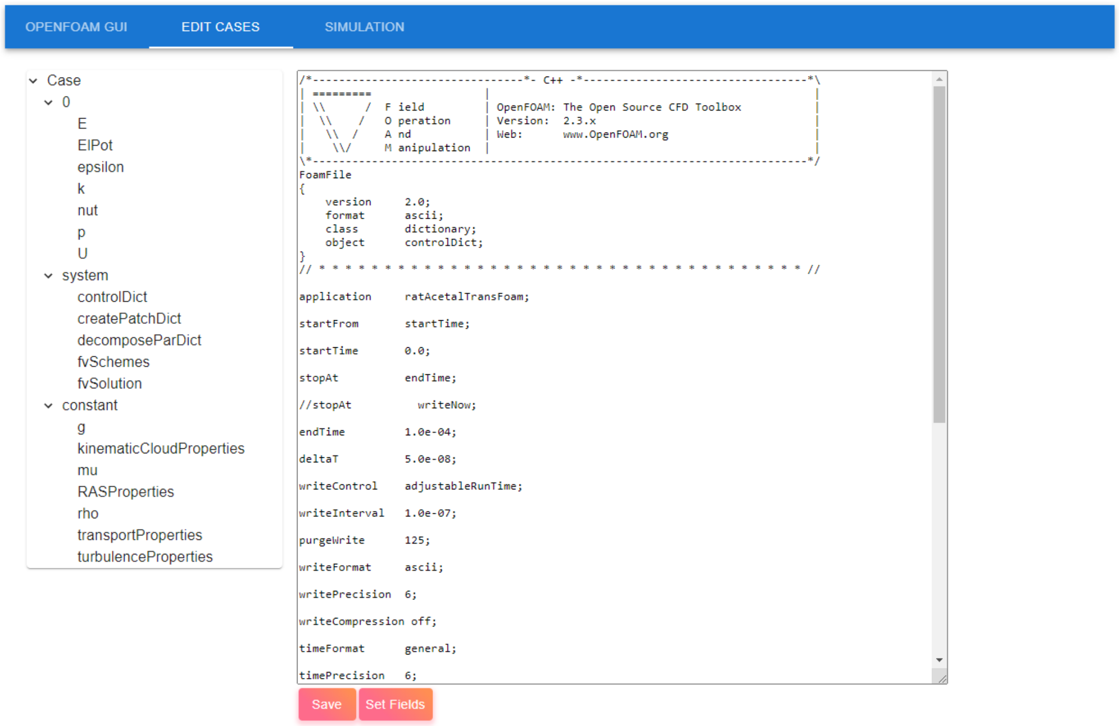


Figure S3. The ***OPENFOAM-GUI*** *page. Case folders and Solvers are selected here. A command line input and terminal window allow the user to enter shell commands and inspect application log messages.*

In the **EDIT CASES** page (**Figure S4**), the user is presented with a file tree displaying the files in the previously selected case folder. SimELIT will only display the contents of the /0, /constant, and /system folders. Selecting any file will display it in the panel to the right of the file tree. The file in its entirety is displayed in an editable text window. Changes to files are not written back to file until the **Save** button is selected. Additionally, the Save button will only save the currently selected file. Optionally, the **Set Fields** button can be used to invoke the OpenFOAM setFields utility.

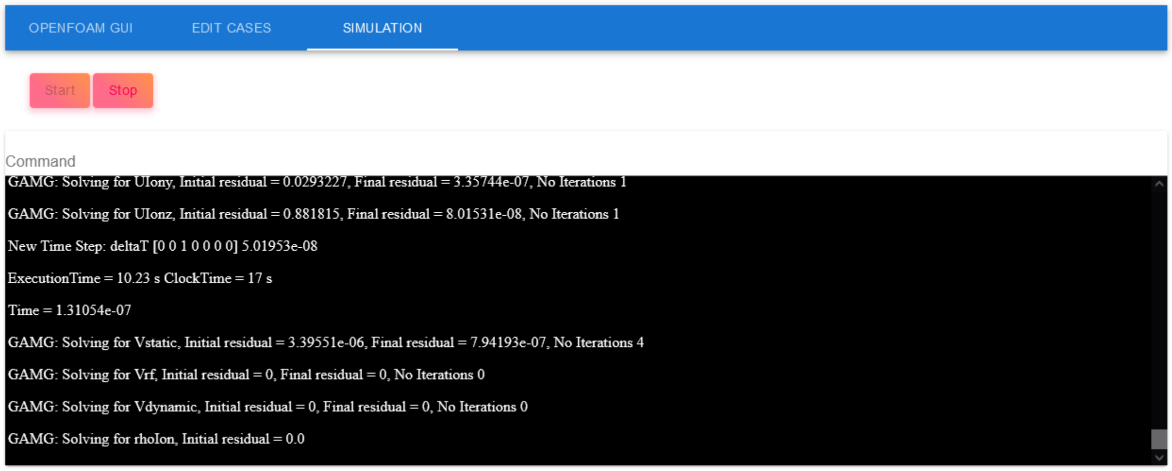


```
/*---------------------------------*- C++ -*-----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.3.x                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     ratAcetalTransFoam;

startFrom       startTime;

startTime       0.0;

stopAt          endTime;

//stopAt          writeNow;

endTime         1.0e-04;

deltaT          5.0e-08;

writeControl    adjustableRunTime;

writeInterval   1.0e-07;

purgeWrite      125;

writeFormat     ascii;

writePrecision  6;

writeCompression off;

timeFormat      general;

timePrecision   6;
```

*Figure S4. The **EDIT CASES** page displaying the selected controlDict file. Edits here can be saved back to file.*

Simulation

Once case editing has been completed, the final step is navigating to the **SIMULATION** tab. See **Figure S5** for its layout. Users are presented with the option to begin a simulation by selecting the



```
GAMG: Solving for UIony, Initial residual = 0.0293227, Final residual = 3.35744e-07, No Iterations 1

GAMG: Solving for UIonz, Initial residual = 0.881815, Final residual = 8.01531e-08, No Iterations 1

New Time Step: deltaT [0 0 1 0 0 0 0] 5.01953e-08

ExecutionTime = 10.23 s ClockTime = 17 s

Time = 1.31054e-07

GAMG: Solving for Vstatic, Initial residual = 3.39551e-06, Final residual = 7.94193e-07, No Iterations 4

GAMG: Solving for Vrf, Initial residual = 0, Final residual = 0, No Iterations 0

GAMG: Solving for Vdynamic, Initial residual = 0, Final residual = 0, No Iterations 0

GAMG: Solving for rhoIon, Initial residual = 0.0
```

Start button and end a running simulation with the **Stop** button. Once started, the output from a simulation is captured and displayed in the UI's terminal window. The results of the simulation are output to the case folder under consideration.

Figure S5. The ***SIMULATION*** *page. Simulation run control and captured log information are shown.*