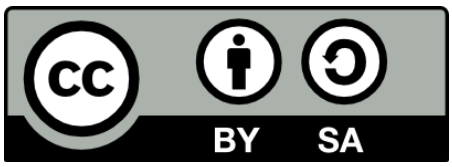
openCARP

- User's Manual -

Anton Prassl [c], *Aurel Neic* [f], *Axel Loewe* [a], *Edward Vigmond* [d,e], *Gernot Plank* [c], *Gunnar Seemann* [b], *Jorge Sánchez* [a], *Martin Bishop* [h], *Mark Nothstein* [a], *Patrick Boyle* [i], *Philipp Zschumme* [a], *Rafael Sebastian* [g], *Yung-Lin Huang* [b]

[a] *Karlsruhe Institute of Technology (KIT), Germany,* [b] *Universitäts-Herzzentrum Freiburg, Germany,* [c] *Medical University of Graz, Austria,* [d] *University of Bordeaux, France,* [e] *LIRYC Electrophysiology and Heart Modeling Institute, France,* [f] *NumeriCor GmbH, Austria,* [g] *University of Valencia, Spain,* [h] *King's College London, UK,* [i] *University of Washington,USA*

January 14, 2021



Contents

Introduction

openCARP is an open cardiac electrophysiology simulator for in-silico experiments. Its source code is public and the software is freely available for academic purposes. openCARP is easy to use and offers single cell as well as multiscale simulations from ion channel to organ level. Additionally, openCARP includes a wide variety of functions for pre- and post-processing of data as well as visualization. The Python-based carputils framework enables the user to develop and share simulation pipelines, i.e., automating in-silico experiments including all modeling, simulation, and evaluation steps.

The implementation of openCARP builds on two decades of experience gained from the proprietary predecessors, the Cardiac Arrhythmia Research Package (CARP) developed by Ed Vigmond and Gernot Plank and acCELLerate developed by Gunnar Seemann and Axel Loewe. Both simulators have been used in over 100 scientific studies. Gunnar Seemann (Freiburg, Germany) and Axel Loewe (Karlsruhe, Germany) received funding from the German Research Foundation (DFG) to develop a sustainable cardiac simulator. They asked Ed Vigmond (Bordeaux, France) and Gernot Plank (Graz, Austria) to join forces which they agreed to in March 2018. Since then, we are developing openCARP and will release the first version in March 2020. Beside the four group leaders, the core developer Aurel Neic joined the openCARP steering committee in November 2019.

1.1 Package description

The main benefit of openCARP is that it allows performing electrophysiological simulations of cardiac tissue with just a few steps. openCARP is backwards compatible with CARP/CARPentry allowing to reproduce a larger number of published studies. The program consists of three main components: a parabolic solver, an ionic current component, and an elliptic solver. Each of these components has a set of sockets in which to plug the components performing the bulk of the work. The parabolic solver is responsible for determining the propagation of the electrical activity by determining the change in transmembrane voltage from the extracellular electric field and the current state of the transmembrane voltage. The elliptic solver unit determines the extracellular potential from the transmembrane voltage at each time instant. The ionic model component describing ionic transmembrane currents is computed from a separate library linked in at compile time. openCARP has been written to run in either of two parallel computation modes, a shared memory model, or a distributed memory model. A large common code base is maintained between the two versions and only low-level wrapper functions implement the specific memory model. Parallelization is realized in the shared memory model based on OpenMP ® directives and native numerical libraries. For distributed memory parallelization, extensive use is made of the PETSc parallel library as well as MPI function calls.

1.2 Technical requirements

openCARP can be deployed on any Unix platform including all current Linux distributions as well as macOS. A Windows version is not available, although, technically, this would be achievable with some effort. Hardware platforms upon which openCARP was successfully used range from laptops to numerous large scale high performance computing (HPC) facilities around the world.

1.3 Distribution packages

The software is freely available for academic purposes under the Academic Public License and various distribution models were conceived to deal with technicalities of installation and support. The following openCARP packages are supported:

- **Docker container:** We provide an openCARP container based on Linux. This allows a fast execution of the software and all its command line tools. No support for meshalyzer visualization.
- **Source code:** The source code of openCARP is available under git.opencarp.org. We provide an easy way to compile using CMake which takes care of looking up the path to the dependencies. You can find a more detailed explanation in the Installation section.
- **Binary package:** Binaries for current Linux distributions and macOS are planned for the future. Currently we support the Ubuntu LTS versions.

Installation

In this section you will find a complete description on how to install openCARP and different tools with just a few steps.

2.1 openCARP packages

openCARP is a set of different packages that together are a powerful tool to create in-silico experiments.

- openCARP
- carputils
- meshtool
- meshalyzer

and a number of examples.

2.2 Installation of openCARP Docker containers

First, install Docker for your platform.
There are 2 options for the openCARP installation using Docker.

1. After the Docker installation use docker commands to pull and use the docker images.
2. Alternatively, install using our script which will take care of all installation steps for you.

2.2.1 Pull the docker image (Option 1)

Pull the latest openCARP docker image:

```
docker pull docker.opencarp.org/opencarp/opencarp:latest
```

Run the docker image using the following terminal command:

```
docker run -it docker.opencarp.org/opencarp/opencarp:latest
```

You can now use openCARP. Head over to the experiments section to get started.

2.2.2 Install our docker script (Option 2)

Docker provides a variety of commands to utilize docker images:

```
docker --help
```

To help you quick start, our script wraps the most commonly used docker commands. Tell the install script where to place the binary files (should be a diretory in your `$PATH`). Use `sudo` if admin permission is required to write there:

```
cd <path-to-your-codebase>/openCARP/docker
./install_script.sh <path-to-your-bin>
```

For example, to install to `/usr/local/bin`, run:

```
sudo ./install_script.sh /usr/local/bin
```

The script enables you to start with the keyword `opencarp-docker` followed by the appropriate command.

Make sure the script is installed correctly:

```
opencarp-docker help
```

Pull the docker image:

```
opencarp-docker pull
```

For usage examples and more information, please refer to our docker/README.

For visualization, please also refer to installation of meshalyzer and locally install meshalyzer because it is not included in the Docker container.

2.3 Application bundles

We are currently developing a set of bundles to simplify the installation process.

2.4 Building from source

openCARP needs a minimal set of dependencies to run. First you need to install the following prerequisites using your package manager or from their official release.

2.4.1 Prerequisites

- git
- binutils
- C and C++ compilers
- make
- CMake
- PETSc
- gengetopt
- zlib-devel

Tips for installing `PETSc`: If you are not experienced with its various configurations, please refer to the following suggestions. Set `--prefix=` to the location you would install PETSc. After installation, set the location installed PETSc for the environment variable `PETSC_DIR`.

```
./configure \
    --prefix=/opt/petsc \
    --download-mpich \
    --download-fblaslapack \
    --download-metis \
    --download-parmetis \
    --download-hypre \
    --with-debugging=0 \
    COPTFLAGS='-O2' \
    CXXOPTFLAGS='-O2' \
    FOPTFLAGS='-O2'
make all
make install
```

If you have all the listed dependencies, a quick way to get openCARP up and running is the following:

Clone the repository and enter the codebase folder `openCARP`

```
git clone git@git.opencarp.org:openCARP/openCARP.git
cd openCARP
```

We provide CMake files and Makefiles for building the codebase, choose one fits your workflow.

2.4.2 Building using CMake

Run CMake to create the `_build` build folder

```
cmake -S. -B_build
```

Compile the codebase, and all built executables are located in the `_build/bin` folder.

```
cmake --build _build
```

(Optional) You can also install the executables to the system using

```
cmake --build _build --target install
```

2.4.3 Building using the Makefile

The codebase can be compiled using

```
make setup
```

This target includes updating the codebase (`make update`) and code compilation (`make all`).
Links to all compiled executables are located in the `bin` folder.

2.5 Installing carputils from Source

2.5.1 Docker

A quick way to get started is pulling the openCARP docker image.

2.5.2 Linux/MacOS

To locally run carputils, first install openCARP, Python3, and pip.
Install Python3 packages using `pip`.

```
pip3 install --upgrade pip
pip install scipy matplotlib numpy pandas scipy tables
```

Clone the carputils repository.

```
git clone https://git.opencarp.org/openCARP/carputils.git
```

Add `carputils/bin` to your `PATH`, and `carputils` to your `PYTHONPATH` For example, if you clone carputils to `$HOME/install`, add the following lines in your .bashrc:

```
export PATH=$PATH:$HOME/install/carputils/bin
export PYTHONPATH=$PYTHONPATH:$HOME/install/carputils
```

Download a settings file for carputils. Specify the proper directories for openCARP-CPU and other openCARP tools if you use them. For example,

```
CPU: $HOME/install/openCARP/bin
```

Rename the settings file to `settings.yaml` and put it in the root directory of the carputils.

2.6 Meshalyzer

Meshalyzer is a graphical program for display time dependent data on 3D finite elment meshes. Of course, it can show static data on fewer dimensions as well. It is developed to be compatible with the cardiac simulation environment openCARP.

It uses IGB anbd its own file formats which are simple and easily converted to/from more popular formats like VTK.

2.6.1 Installation of meshalyzer

Meshalyzer is regularly used on Linux and Mac machines. It has worked under Windows as well.

2.6.1.0 Prerequisities

- libpng
- FLTK 1.3.x. Manually downloaded and installed is better since the prepackaged versions often have issues with fltk-config.
- glew.
- glut
- OSMesa (To make the "windowless" rendering version)
- VTK

Except for fltk, using your package manager is recommended for installing these packages if possible. On macOS, you could use homebrew.

2.6.1.0 Compiling

If VTK is installed, edit the **make.conf** to set the `VTK_INC` and `VTK_DIR` variables to point to the directories containing the VTK header files and libraries, respectively. Comment them out otherwise. Then type:

```
make
```

For the windowless version, type:

```
make mesalyzer
```

The compiled executables will be found in the **src/** directory. Copy the binaries to a suitable location.

2.6.1.0 Documentation

To compile the doucmentation a LaTeX package is required. Follow the steps described below to create the Meshalyzer manual.

```
cd manual
pdflatex manual
pdflatex manual
```

2.7 Meshtool - A mesh manipulation utility

Meshtool is a comand-line tool written in C++. It is designed to apply various manipulations to volumetric meshes. The mesh manipulations are referred to as "meshtool modes". This are verbs that describe what meshtool is supposed to do with a mesh. The most important modes are:

- extract : Extract submeshes, surfaces or data from a given mesh.
- insert : Insert mesh information or data from a submesh back into the mesh.
- convert : Convert between CARP, VTK, VTU and MMG binary and ascii formats.
- map : Map index sets (e.g. vertex lists, surfaces) between mesh subsets.
- query : Query various mesh informations like bounding-box, tags, resolution or quality.
- smooth : Smooth surfaces between multiple regions, while preserving mesh quality.
- resample : Increase or reduce resolution of meshes, mesh regions and Purkinje systems.

Most modes require the user to specify an object (e.g. what to extract or insert) followed by mandatory and optional mode options.

2.7.1 Supported Mesh Formats

The currently supported mesh formats are:

- openCARP/CARPentry: Ascii and binary formats used by the openCARP and CARPentry cardiac electrophysiology simulator.
- VTK: Ascii and binary legacy formats used by The Visualization Toolkit (VTK).
- VTU: Binary format used by VTK.
- MMG: Ascii format used by MMG3D.
- OBJ: Wavefront ascii format. Used only for surfaces.

2.7.2 Building

No 3rd-party libraries are required. The code is being built and used under many Linux distributions, MacOS and Windows 10 (using WSL).

Meshtool is build (in parallel, no -j option required) via the command

```
make
```

The compiled binary is then located in

```
meshtool/meshtool
```

The build process can be customized in the `my_switches.def` file. This file is auto-generated by the first build run. There, the user can configure some basic build settings like:

- Compile in debug or in optimized mode.
- Link statically, if static librarie are availiable.
- Use OpenMP.
- Use (`long`, `double`) or (`int`, `float`) for (`mt_int`, `mt_real`)
- choose between intel, gnu, clang compilers

Mathematical model formulation

openCARP is a generic solve for the cardiac bidomain equations and uses finite elements to discretize the cardiac domain. While the bidomain equations are considered to be an accurate description of cardiac bioelectric activity, for many scenarios of practical interest the computationally less expensive monodomain equation suffices.

3.1 Ionic model equations

For electrophysiological simulations, the basic unit of the model is the cellular membrane. Cellular models of cardiac electrical activity were first constructed over 65 years ago, and today, these models have been refined and are routinely put into organ scale simulations. The modelling components are described below.

Ionic models refer to the electrical representation of the cell based on the movement of ions across the cell membrane, resulting in a change in the voltage across the membrane. Schematically, a capacitor is placed in parallel with a number of ionic transport mechanisms. In general, an ionic model is of the form:

$$I_{\mathrm{m}} = C_m \frac{dV_{\mathrm{m}}}{dt} + \Sigma_\chi I_\chi \tag{1}$$

where V_{m} is the voltage across the cell membrane, I_{m} is the net current across the membrane, C_{m} is the membrane capacitance, and I_χ is a particular transport mechanism, either a channel, pump or exchanger. Additional equations may track ionic concentrations and the processes which affect them, like calcium-induced release from the sarcoplasmic reticulum. By convention, outward current, i.e., positive ions leaving the cell, is defined as being positive, while inward currents are negative.

Channels are represented as resistors in series with a battery. The battery represents the Nernst Potential resulting from the electrical field developed by the ion concentration difference across the membrane. It is given by

$$E_S = \frac{RT}{zF} \ln \frac{[S]_i}{[S]_e}$$

where R is the gas constant, T is the temperature, F is Faraday's constant and z is the valence of the ion species S. Channels are dynamical systems which open and close in response to various conditions like transmembrane voltage, stretch, ligands, and other factors. The opening and closing rates of the channels vary by several orders of magnitudes, and are generally, nonlinear in nature. Thus, the equivalent electrical resistance of a channel is a time dependent quantity.

The first representation of a cardiac cell was that produced by D. Noble of a Purkinje cell, based on modification of the Hodgkin-Huxley nerve action potential. Since then, hundreds of models have been developed for many reasons. Ionic models need to be developed to match experimental procedures if the models are to be predictive and offer insight into mechanistic workings. In mammals, action potential durations range

from tens of milliseconds for mice to several hundreds of milliseconds for large animals. Atrial myocyte protein expression is quite different from that of ventricular myocytes, resulting in different action potential durations and shapes. Even nearby cells exhibit action potential differences due to slightly different levels of channel protein expression. The complexity of ionic models has been steadily growing as more knowledge is gained through better experimental techniques, equipment and specific blockers of transport mechanisms. As more mechanisms are identified as affecting electrophysiology, either directly or indirectly, they are incorporated into ionic models. Selection of the model to use is not always obvious as different models may be available for the same species and heart location, which may yield quite different behaviour. Regardless, identifying the strengths and weaknesses of an ionic model for a particular application is important.

Ionic models may be biophysically detailed or phenomenological. Biophysically detailed models attempt to discretely depict important currents and processes within the cell. Early models had approximately ten equations depicting only sodium and potassium channels, while present models have hundreds of equations taking into account not only membrane ion transporters, but intracellular calcium handling, mitochondrial function, and biochemical signalling pathways. Conversely, phenomenological ionic models use a set of currents which faithfully reproduce whole cell behaviour in terms of action potential shape and duration, and restitution properties. The currents in the phenomenological model are not physiological but can be considered as amalgamations of known currents. While these models are computationally much simpler and easier to tune, they lose the direct correspondence with the biophysics which makes implementing drug effects or cellular pathologies challenging. Finally, cellular automata models have also been used, and can be considered as a type of phenomenological model. These models do not use differential equations but have a set of rules which dictate transitions between discrete states of the model. As such, these models are computationally light, and can be as detailed as required. However, behaviour may not be as rich as differential equations. The choice of model, biophysical or phenomenological, depends on the nature of the problem being considered, and the availability of computational resources for the problem size.

3.1.1 Hodgkin-Huxley dynamics

The Hodgkin-Huxley approach is named after the Nobel laureates who were the first to develop a mathematical model of the neural action potential. Single channel activity recordings show that channels have a small set of discrete conductance states, with the channel stochastically transitioning between closed states and open states. Short time analysis of a single channel is very difficult to interpret but ensemble averaging clearly reveals smooth kinetics in changes of channel conductance. In the Hodgkin-Huxley formulation, channel conductance is assumed to be controlled by gates which take on values between 0 and unity, representing the portion of the cells in one state. Since cells have hundreds of ion channels if not more, this approximation holds well. Current flow produced by ion species X passing through a channel is then described by

$$I_{\mathrm{S}} = \overline{g}_S \prod_n \eta_n (V_{\mathrm{m}} - E_{\mathrm{S}})$$

where $\overline{g}_{\mathrm{S}}$ is the maximum conductance of the channel and η_{n} is a gating variable. Often the gating variable is assumed to follow first order dynamics so

$$\begin{aligned} \frac{d\eta}{dt} &= \alpha(V_{\mathrm{m}})(1-\eta) - \beta(V_{\mathrm{m}})\eta \\ &= \frac{\eta_\infty(V_{\mathrm{m}}) - \eta}{\tau_\eta(V_{\mathrm{m}})} \end{aligned}$$

where α and β are rates which can be cast into an equivalent form of a steady state value (η_∞) and a rate of change (τ_η). The advantage of this latter formulation is that mathematically, the update of the gating variables can be performed by a numerical integration method, the Rush-Larsen technique, which is guaranteed to unconditionally keep the gating variable bounded within the range [0,1] while allowing a large time step.

3.1.2 Markov models

Markov models describe the channel as a set off states, such as the conductance states seen in single channel recordings, as well as the non conducting states through which a channel must pass to reach them. Transitions between states are described by rate constants which can be functions of concentrations or voltages, or fixed. These is a variables for each state which represent the portion of channels in a cell which are in that state. As such, the sum of state variables is unity. A Hodgkin-Huxley made can be converted to a Markov model by considering a gating variable as a two state model. However, the advantage of the Markov representation is its ability to model drug interaction. Essentially, drug binding doubles the number of possible states in an ionic model, augmenting the original states with drug-bound versions. One may easily limit drug binding to a particular subset of channel states, and more finely control channel kinetics.

3.2 Bidomain equations

The bidomain equations relate intracellular potential, ϕ_i, to the extracellular potential, ϕ_e, through the transmembrane current density, I_m. They are written in the standard form as

$$\nabla \cdot \sigma_i \nabla \phi_i = \beta I_m - I_i \tag{2}$$

$$\nabla \cdot \sigma_e \nabla \phi_e = -\beta I_m - I_e \tag{3}$$

where σ_i and σ_e are the intracellular and extracellular (homogenized) bidomain conductivity tensors, respectively, β is the bidomain surface-to-volume ratio, I_e and I_i are

an extracellular and intracellular current density stimuli, respectively, and I_m is the transmembrane current.

The transmembrane current is given by

$$V_m = \phi_i - \phi_e \tag{4}$$

$$I_m = C_m \frac{\partial V_m}{\partial t} + I_{ion} - I_{tr} \tag{5}$$

$$\frac{\partial \eta}{\partial t} = f(V_m, \eta, t) \tag{6}$$

$$I_{ion} = g(V_m, \eta, t) \tag{7}$$

where I_{tr} is an assumed transmembrane current density stimulus, as delivered by an intracellular electrode, C_m is the capacitance per unit area (fixed to $1\mu\text{F/cm}^2$), V_m is the transmembrane voltage which is defined as $\phi_i - \phi_e$, and I_{ion} is the current density flowing through the ionic channels. Ionic currents I_{ion} (equation 7) depend on the membrane state and various ion concentrations throughout the cell and in the interstitial domain which are given by the state equation 6. This form of the bidomain equations is referred to as the *parabolic-parabolic* form since both equations 2 and 3 are parabolic PDEs.

By adding Eq. 2 and Eq. 3 and using the definition of V_m, the equations can be cast in a slightly different form with V_m and ϕ_e as the independent variables[**?**].

$$\nabla \cdot (\sigma_i + \sigma_e)\nabla \phi_e = -\nabla \cdot \sigma_i \nabla V_m - I_e - I_i \tag{8}$$

$$\nabla \cdot \sigma_i \nabla V_m = -\nabla \cdot \sigma_i \nabla \phi_e + \beta I_m \tag{9}$$

Eq. 8 is an elliptic equation and Eq. 9 is a parabolic equation. Hence, this recast is referred to as the *elliptic-parabolic* which is the more popular cast for solving the equations since the two main variables of interest, V_m and ϕ_e, are retained. This is more convenient for experimental validation at the tissue scale.

3.3 Monodomain equation

Assuming that anisotropy ratios between intracellular and extracellular domains are equal, that is, the tensors can be related by a scalar, λ, like

$$\sigma_e = \lambda \sigma_i \tag{10}$$

we can recast the bidomain equation in a simpler form. Plugging Eq. 10 into 2 and using 4 yields

$$\nabla \cdot \sigma_i \nabla \phi_i = \beta I_m - I_i \tag{11}$$

$$\nabla \cdot \sigma_i \nabla \phi_e = \nabla \cdot \sigma_i \nabla \phi_i - \nabla \cdot \sigma_i \nabla V_m = -\frac{1}{\lambda}(\beta I_m + I_e). \tag{12}$$

Subtracting 12 from 11 and multiplying with $\lambda/(1+\lambda)$ results in

$$\frac{\lambda}{1+\lambda}\nabla \cdot \sigma_i \nabla V_m = \beta I_m + \frac{1}{1+\lambda} I_e - \frac{\lambda}{1+\lambda} I_i. \tag{13}$$

Now we subtract 11 from 13 to arrive at

$$\frac{\lambda}{1+\lambda}\nabla \cdot \sigma_i \nabla V_m = \beta I_m + \underbrace{\frac{1}{1+\lambda} I_e - \frac{\lambda}{1+\lambda} I_i}_{-\beta I_{tr}}. \tag{14}$$

$$(1+\lambda)\nabla \cdot \sigma_i \nabla \phi_e = -\nabla \cdot \sigma_i \nabla V_m - I_e - I_i \tag{15}$$

As indicated in Eq. 14, the combined effect of intracellularly and extracellularly injected stimulus currents can be interpreted as a depolarizing transmembrane stimulus if we define

$$I_{tr} = -\frac{1}{\beta}\left(\frac{1}{1+\lambda} I_e - \frac{\lambda}{1+\lambda} I_i\right). \tag{16}$$

The choice $I_e = -I_i$ at any given site is equivalent to a transmembrane current stimulus of strength I_i/β, that is,

$$\beta I_{tr} = -\left(\frac{1}{1+\lambda} I_e - \frac{\lambda}{1+\lambda} I_i\right) = -\left(-\frac{1}{1+\lambda} I_i - \frac{\lambda}{1+\lambda} I_i\right) = -\left(-\frac{1+\lambda}{1+\lambda} I_i\right) = I_i. \tag{17}$$

This is consistent with the notion that the injection of a positive current I_i into the intracellular space increases ϕ_i which exerts a depolarizing effect upon $V_m = \phi_i - \phi_e$. Further, in this case the current terms in Eq. 15 cancel out. As expected, any current injected in one domain is withdrawn at the same spot in the other domain. Therefore no current flow occurs as a consequence of I_e or I_i and no extracellular potential field is set up. All changes in ϕ_e are caused indirectly then via changes in V_m.

Inspecting Eqs. 14-15 reveals that, unlike in the full bidomain case where $\sigma_e \neq \lambda\sigma_i$ holds, the temporal evolution of the transmembrane voltage in Eq. 14 is fully independent of ϕ_e. Hence, if only the evolution of V_m is of interest, only Eq. 14 needs to be solved, but not the elliptic PDE given by 15 which is a more expensive task. For more detailed considerations we refer to the report by Nielsen et al [**?**].

It is worth noting that in 1D monodomain and bidomain equations are always equivalent when using twice the harmonic mean conductivity tensor. This holds true also for planar wave fronts in 3D propagating along any eigenaxes of the conductivity tensors. The monodomain is an approximation which can be used whenever the effect of extracellular fields upon tissue polarization can be ignored. As mentioned above, since the temporal evolution of the V_m in Eq. 14 is fully independent of ϕ_e, any changes in the extracellular potential fields cannot exert any influence upon V_m.

Therefore, under the assumption of equal anisotropy ratios one needs to solve only the parabolic PDE above with the intracellular conductivity set to twice the harmonic mean, $\sigma_i\sigma_e(\sigma_i + \sigma_e)^{-1}$. This yields

$$\nabla \cdot (\sigma_m \nabla V_m) = \beta I_m + \beta I_t r \tag{18}$$

where the bidomain equivalent monodomain conductivity σ_m is given as

$$\sigma_m = \sigma_i\sigma_e(\sigma_i + \sigma_e)^{-1}. \tag{19}$$

3.4 Units

Units are discussed for the monodomain equations for the sake of simplicity, but the same is valid for the bidomain. The monodomain equation can be written in the form

$$\nabla \cdot (\hat{\sigma}_m \nabla V_m) = \beta I_{ion} + \beta C_m \frac{\partial V_m}{dt} \tag{20}$$

The following units are used:

Symbol	Input Unit	Internal Unit	Conversion
V_m	mV	mV	1
I_{ion}	$\mu A/cm^2$	$\mu A/cm^2$	1
t	ms	ms	1
x, y, z	μm	μm	1
β	μm^{-1}	μm^{-1}	1
C_m	fixed value (1)	$\mu F/cm^2$	-
$\hat{\sigma}_m$	S/m	$mS/\mu m$	10^3

Table 1: Units used in openCARP

There are a few exceptions. Although t is input in ms in general, this is not the case for dt, where μs are used.

Rewriting the equation in a semi-discrete form reveals that the equation is not consistent in terms of units, neither with the input units nor the internal units:

$$\frac{1}{\Delta} \cdot \left(\hat{\sigma}_m \frac{1}{\Delta} V_m \right) = \beta I_{ion} + \frac{\beta C_m}{\Delta t} \Delta V_m \tag{21}$$

Inconsistencies arise due to the use of two different units for space coordinates, μm on the left hand side and for β, cm on the right hand side for C_m and I_{ion}. Nonetheless, the choice of internal units make sense. μm is an appropriate unit for resolving the tissue and cm is used in almost any model dealing with membrane kinetics. The inconsistency will be dealt with in the final form after spatial discretization using the finite element method.

3.5 The Galerkin finite element formulation

The Galerkin method is a special case of the Method of Moments which minimizes the weighted residual. The solution is multiplied with a weighting function, ω, and integrated over the entire domain, Ω:

$$\int_{\Omega} \nabla \cdot (\hat{\sigma}_m \nabla V_m)\, \omega \, d\Omega = \beta \int_{\Omega} \left(I_{ion} + C_m \frac{\partial V_m}{dt} \right) \omega \, d\Omega \tag{22}$$

We assign a set of functions which will represent our solution. For FEM, these function are local in scope and are the shape functions of the element, α_i. For the Galerkin method, we choose the weighting function to be the shape functions. In operator notation, $\mathcal{L} = \nabla \cdot \sigma \nabla$, and $b = \beta I_m$

$$\mathcal{L}\phi = b \tag{23}$$

$$< \mathcal{L}\phi, \omega > = < b, \omega > \tag{24}$$

$$\underbrace{< \mathcal{L} \sum_i \alpha_i, \alpha_j >}_{\mathbf{K}} = \underbrace{< \sum_i b_i \alpha_i, \alpha_j >}_{\mathbf{M}} \tag{25}$$

3.6 The local stiffness matrix

$$K_{ij} = < \mathcal{L}\alpha_i, \alpha_j > = \int_{\Omega_e} \nabla \alpha_i \cdot \sigma_{m_{ij}} \nabla \alpha_j d\Omega_e \tag{26}$$

where K_{ij} is a single entry of the local stiffness matrix which we obtain by integrating over the volume Ω_e of the element e. In terms of units, the stiffness matrix is $[mS]$:

$$(\nabla \alpha_i) \left[\frac{1}{\mu m} \right] (\sigma_{m_{ij}}) \left[\frac{mS}{\mu m} \right] (\nabla \alpha_j) \left[\frac{1}{\mu m} \right] (\Omega_e) \left[\mu m^3 \right] = (K_{ij}) \, [mS] \tag{27}$$

3.7 The local mass matrix

$$M_{ij} = < \alpha_i, \alpha_j > = \int_{\Omega_e} \alpha_i \alpha_j d\Omega_e \tag{28}$$

where M_{ij} is a single entry of the local stiffness matrix which we obtain by integrating over the volume Ω_e of the element e. In terms of units, the stiffness matrix is $[\mu^3]$:

$$(\alpha_i) \, [-] \, (\alpha_j) \, [-] \, (\Omega_e) \left[\mu m^3 \right] = (M_{ij}) \left[\mu^3 \right] \tag{29}$$

Rewriting 22 in a semi-discrete form yields

$$\mathbf{Kv} = \underbrace{\frac{\beta C_m}{\Delta t}}_{\kappa} \mathbf{M} \Delta \mathbf{v} + \beta \mathbf{M} \mathbf{I_{ion}} \tag{30}$$

In terms of units, κ is evidently inconsistent and needs to be converted to ensure compatibility between left-hand and right-hand side:

$$\kappa = \left[\frac{1}{\mu m}\right]_{(\beta)} \left[\frac{\mu F}{cm^2}\right]_{(C_m)} \left[\frac{1}{ms}\right]_{(\Delta t)} \tag{31}$$

Rewriting μF as

$$\mu F = \frac{\mu A s}{V} = \frac{mA \cdot ms}{V} = mS \cdot ms \tag{32}$$

and inserting into 31 yields

$$\kappa = \left[\frac{1}{\mu m}\right]_{(\beta)} \left[\frac{mS \cdot ms}{cm^2}\right]_{(C_m)} \left[\frac{1}{ms}\right]_{(\Delta t)} = \left[\frac{mS}{cm^2 \mu m}\right]_{(\kappa)} \tag{33}$$

To arrive at μA, as required to match the left-hand side, when multiplying with $\mathbf{M}$, we convert κ to a per μm base. The required conversion factor is

$$\kappa = \left[\frac{mS}{10^8 \mu m^2 \mu m}\right]_{(\kappa)} = 10^{-8} \left[\frac{mS}{\mu m^3}\right]_{(\kappa)} = 10^{-8} \tilde{\kappa} \tag{34}$$

Using $\tilde{\kappa}$ results in a consistent equation now where all terms are given in μA, except for the term linked to I_{ion}. No unit conversion is required for I_{ion} due to the operator splitting technique used within openCARP:

$$\frac{\beta C_m}{\Delta t} \mathbf{M} \Delta \mathbf{v} = \mathbf{K} \mathbf{v} \tag{35}$$

$$C_m \frac{\Delta \mathbf{v}}{\Delta t} = -\mathbf{I_{ion}} \tag{36}$$

That is, the inconsistency is resolved since the term given in μm drop out and only those terms given on a per cm base remain. Note that in a computing scheme where the ODE is not split of from the parabolic PDE, an additional unit conversion would be required to make the term $\beta \mathbf{M} I_{ion}$ consistent. For the sake of completeness, in this case I_{ion} has to be multiplied by

$$\begin{aligned} \beta \mathbf{M} I_{ion} = \left[\frac{1}{\mu m}\right]_{(\beta)} \left[\mu m^3\right]_{(\mathbf{M})} \left[\frac{\mu A}{cm^2}\right]_{(I_{ion})} &= \left[\frac{1}{\mu m}\right]_{(\beta)} \left[\mu m^3\right]_{(\mathbf{M})} \left[\frac{\mu A}{10^8 \mu m^2}\right]_{(I_{ion})} \\ &= \beta \mathbf{M} \tilde{i}_{ion} \end{aligned} \tag{37}$$

where $\frac{mS}{\mu m}$

$$\left[\frac{\mu A}{\mu m^2}\right]_{(\tilde{i}_{ion})} = 10^{-8} \left[\frac{\mu A}{cm^2}\right]_{(I_{ion})} \tag{38}$$

to arrive at μA like with all other terms of the equation.

The parabolic PDE is now consistent as can be seen here:

$$[mS]_{(\mathbf{K})} \cdot [mV]_{(\mathbf{v})} = \left[\frac{mS}{\mu m^3}\right]_{(\tilde{\kappa})} \cdot \left[\mu m^3\right]_{(\mathbf{M})} \cdot [mV]_{(\Delta \mathbf{v})} = [\mu A] \tag{39}$$

3.8 Conductivities

β is the ratio of the surface area of all cells within a volume to the volume. It is based on the cellular surface to volume ratio and the number of cells per unit volume. Conductivities are not material properties alone but also take into account the geometry. Let σ_ξ be the true isotropic conductivity of domain ξ. In any one direction, we homogenize over 1 domain to arrive at an equivalent conductivity in principal direction eta, $\sigma^h_{\xi,\eta}$:

$$\frac{L}{\sigma^h_{\xi,\eta}\overline{\Gamma}} = \frac{1}{N}\sum_{k=1}^{N}\left(\int_0^{L_\eta}\frac{d\eta}{\sigma_\xi \mathcal{A}(\eta)} + \sum_j R_{k,j}\right) \tag{40}$$

where N is the number of parallel cells, $\overline{\Gamma}$ is the average domain dimension in direction η, L_η is the dimension of the cell in η, $\mathcal{A}_\eta$ is the cross-sectional area along η, and $R_{k,j}$ is the resistance of any gap junctions crossed. The principle directions are along the fibre (f), in the plane of the sheet (s), and the sheet normal (n).

To homogenize over both domains, the following relationship must be obeyed

$$\int_\Omega \nabla\cdot\overline{\sigma}_\xi\nabla\phi_\xi\, d\Omega = \pm\beta I_{ion} \tag{41}$$

which implies that

$$\overline{\sigma}_\xi = \begin{bmatrix} A_{\xi,f} & 0 & 0 \\ 0 & A_{\xi,s} & 0 \\ 0 & 0 & A_{\xi,n} \end{bmatrix}\overline{\sigma}^h_\xi \tag{42}$$

where $A_{\xi,\eta}$ is the portion of the volume cross-section in direction η occupied by domain ξ. If we assume the following mapping which scales the domains in the unit volume to occupy the entire volume

$$(f,s,n) \xrightarrow{T} (f',s',n') \tag{43}$$

where

$$T(f,s,n) = \left(f, \frac{s}{L_{\xi,s}}, \frac{n}{L_{\xi,n}}\right) \tag{44}$$

This leads to the relationship that the fraction of the volume occupied by domain ξ is the Jacobian of this scaling.

File Formats

The input and output files used in openCARP adhere to their own specific structure. This structure brings flexibility making different datasets easier to export or import. These file formats have been optimized to handle large amounts of data for fast I/O for pre- and post-processing. Using meshtool you can also import/export to different standard file formats. (e.g. VTK, obj, ensight).

4.1 Input Files

4.1.1 Parameter file

Extension: .par

The parameter file contains all the input options to define a simulation. These parameters can be specified either by using this file or by giving them as command line arguments. The last option definition, whether in a file or on the command line, overrides any previous definition. The command line is parsed from left to right. Command line options may be placed in the parameter file by removing the preceding `-` and using `=`. For example, the time step is specified on the command line by `-dt f` where `f` is a floating point number. This can be placed in a parameter file on a line by itself as `dt = f`.

openCARP uses the parameter file as an input file to read all information regarding input files, output files and parameters for the simulation as follows:

> `openCARP +F parameter.par`

4.1.2 Element file

Extension: .elem

In general, openCARP supports various types of elements which can be mixed in a single mesh. The file begins with a single header line containing the number of elements, followed by one element definition per line. The element definitions are composed of an element type specifier string, the nodes for that element, then optionally an integer specifying the region to which the element belongs.

In the table below is an example of the file format:

Format	Example
n	2000
T_0 $N_{0,0}$ $N_{0,1}$ $N_{0,mi-1}$ $[\text{elemTag}]_0$	Tt 0 1 2 150 1
...	...
T_{n-1} $N_{n-1,0}$ $N_{n-1,1}$ $N_{n-1,mi-1}$ $[\text{elemTag}]_{n-1}$	Tt 123 124 125 210 10

The node indicies are determined by their order in the points file. Note that the nodes

are 0-indexed, as indicated in Sec. Node file above. The element formats supported in openCARP, including their element type specifier strings are given in table below. Note that cH element type is for internal use only, not supported in mesh files. The ordering of nodes in each of the 3D element types is shown in Nodal ordering of 3D element types.

Type Specifier	Description	Interpolation	#Nodes
Ln	Line	linear	2
cH *	Line	cubic	2
Tr	Triangle	linear	3
Qd	Quadrilateral	Ansatz	4
Tt	Tetrahedron	linear	4
Py	Pyramid	Ansatz	5
Pr	Prism	Ansatzv	6
Hx	Hexahedron	Ansatz	8

4.1.3 Node file

Extension: .pts

The node (or points) file starts with a single header line with the number of nodes, followed by the coordinates (x,y,z) of the nodes, one per line, in μm.

Format	Example
n	2000
$x_0\ y_0\ z_0$	1000.00 1000.00 300.00
...	...
$x_{n-1}\ y_{n-1}\ z_{n-1}$	10000.00 10000.00 3000.00

4.1.4 Fiber orientation file

Extension: .lon

Fibres are defined on a per-element basis in openCARP. They may be defined by just the main fibre direction, in which case a transversely isotropic conductivity must be used, or additionally specifying the sheet (or transverse) direction to allow full orthotropy in the model. The file format starts with a single header line with the number of fibre vectors defined in the file (1 for fibre direction only, 2 for fibre and sheet directions), and then one line per element with the values of the fibre vector(s). Note that the number of fibres must be equal to the number of elements read from the element file.

Format	Example
n_f	2
$f_{0,x}\ f_{0,y}\ f_{0,z}\ s_{0,x}\ s_{0,y}\ s_{0,z}$	0.831 0.549 0.077 0.473 -0.775 0.417
...	...

Format	Example
$f_{n-1,x}$ $f_{n-1,y}$ $f_{n-1,z}$ $s_{n-1,x}$ $s_{n-1,y}$ $s_{n-1,z}$	0.0 0.0 0.0 0.0 0.0 0.0

4.1.5 Pulse definition file

Extension: .trc

The pulse definition file allows to introduce a signal with a predefined shape by the user. It starts with a single header line that is the total number of samples of the signal. Then, the signal is defined with one line per sample (time followed by the signal value to be applied between this time and the following time step). Note that the time must always increase from line to line.

Format	Example
n	1000
t_0 val_0	0 60
...	...
t_{n-1} val_{n-1}	1000 0

4.1.6 Vertex specification file

Extension: .vtx

Format	Example
n	1000
intra | extra	intra
node_0	0
...	...
node_{n-1}	123

4.1.7 Vertex adjustment file

Extension: .adj

Format	Example
n	1000
intra | extra	intra
node_0 val_0	0 1000
...	...
node_{n-1} val_{n-1}	123 65

4.2 Output Files

4.2.1 IGB file

Extension: .igb

openCARP simulations usually export data to a binary format called IGB. IGB is a format developed at the University of Montreal and originally used for visualizing regularly spaced data. An IGB file is composed of a 1024-byte long header followed by binary data which is terminated by the ASCII form feed character ($\hat{}L/character12$). For unstructured grids, the individual dimensions are meaningless but x * y * z should be equal to the number of vertices. The header is composed of strings of the following format, separated by white space: Keyword: value. Note that the header must be padded to 1024 bytes.

Keywords	Type	Keywords	Type
x	int	inc_x	float
y	int	inc_y	float
z	int	inc_z	float
t	int	dim_x	float
systeme	string	dim_y	float
type	string	dim_z	float
unites	string	unites_x	string
facteur	float	unites_y	string
zero	float	unites_z	string
org_x	float	comment	string
org_y	float	aut_name	string
org_z	float	transparent	hex
org_t	float		

Format	Example
x:int y:int z:int t:int type:string systeme:string org_t:float dim_t:float unites_x:string unites_y:string unites_z:string unites_t:string unites:string	x:333136 y:1 z:1 t:1452 type:float systeme:little_endian org_t:0 dim_t:1451 unites_x:um unites_y:um unites_z:um unites_t:ms unites:mV facteur:1 zero:0
float_{n0} float_{n1} ... float_{n-1} ... float_{n0} float_{n1} ... float_{n-1}	-80 -80 ... -80 ... -75 -75 ... -75

4.2.2 Dynamic points file

Extension: .dynpts

Points may also move in space as functions of time. This can be used to represent displacement during contraction, for example. The number of points must remain constant for all time instants, as well as the elements defined by them. Dynamic point files use the IGB format (see IGB file) with data type vec3f.

4.2.3 Vector data file

Extension: .vec and .vpts To display vector data, an auxiliary set of points must be defined by a file with the .vpts suffix. It follows the same format as the Node file. A file with the same base name but having the extension .vec defines the vector and scalar data. The scalar datum, as indicated, is optional. The .vec format can also be used for visualization of fiber orientations stored in .lon files. For this sake, a .vpts file must be generated holding the coordinates of the centers of each element in the mesh. This is conveniently achieved with GlElemCenters and changing the fiber file extension to .vec. For example, for a mesh with elements and where corresponds to the components of the fiber vector :

Format	Example
data.x data.y data.z [scalar_datum]	8.12453e-01 -2.22623e-01 1.25001e00
data.x data.y data.z [scalar_datum]	5.68533e-01 -1.81807e-01 1.04956e00
data.x data.y data.z [scalar_datum]	5.70671e-01 -1.67572e-01 1.06615e00

Vector data can also be input as an IGB file using the types vec3f, vec4f, vec3d, vec4d where 3 or 4 refers to the number of elements in each datum, and d and f refer to float or double. The first 3 elements define the value of the vector field, and the optional 4-th element is the scalar component as above. This file has the suffix .vec.igb.

4.2.4 Auxiliary grid file

Extension: .pts_t, .elem_t, .dat_t

This format is used by Meshalyzer to display additional data on an auxiliary grid. The grid may contain any of the elements forming the main grid (points, lines, surface elements, volume elements), and the elements may change as a function of time. If scalar data were already read, the number of time instants in the auxiliary grid must either be one, or the same as the scalar data. The points file may have only one time instant, which is then assumed to be constant and applied for all time steps.

A vertex file is mandatory and has the extension .pts_t. An element file is optional. If present, it has the same base name as the vertex file but with the extension .elem_t. Scalar data may also be optionally defined on the auxiliary grid. The file has the same basename as the vertex file but with the extension .dat_t.

Single cell simulations

The single cell experiment tool bench is quite versatile and can be used to replicate a wider range of single cell experimental protocols. Bench handles all time units in ms; all voltages in mV and currents in pA/pF (or μA/cm^2 considering the fixed C_m of 1pF/cm^2). The following sections demonstrate how one explores the available bench functions and explains the command line arguments. To list all available bench options, use:

```
> bench --help
```

You can find a detailed explanation of the parameters available in bench in the Bench commands section.

5.1 Running a simulation

5.1.1 Running in plain mode

Running bench in plain mode is simple and fast. You can call bench in the terminal window by command line as shown in this example:

```
> bench --imp COURTEMANCHE --imp-par "g_Kr*1.6" --stim-curr 20 --
    numstim 4 --bcl 1000
```

5.1.2 Running with carputils

Setting up an in-silico experiment with carputils and bench allows to combine several steps to create more complex experiments than in the plain mode. carputils also provides interfaces for visualization and can be used to expose only certain bench parameters to the user.

This is an example of an experiment defined using carputils:

```
import os
from datetime import date

from carputils import settings
from carputils import tools
from carputils import mesh
from carputils import testing

def parser():
    parser = tools.standard_parser()
    group  = parser.add_argument_group('experiment specific options')
    group.add_argument('--imp',  default='converted_COURTEMANCHE',
                       choices=['TT2','converted_COURTEMANCHE','HH'],
                       help='pick ionic model (default is TT2). For a
   full list look inside opencarp installation folder /physics/limpet/
   models')
```

```
    group.add_argument('--duration',
                       type=float, default=1000.,
                       help='Duration of simulation [ms] (default is
    1000.)')
    group.add_argument('--stim_strength',
                       type=float, default=60.,
                       help='pick transmembrane current stimulus
    strength in [uA/cm^2] (default is 60.)')
    group.add_argument('--stim_dur',
                       type=float, default=2.,
                       help='pick transmembrane current stimulus
    duration in [ms] (default is 2.)')
    #-----------------------------------------------------------------
    group.add_argument('--imp_par', default='',
                       help='parameter to modified in ionic model.')
    group.add_argument('--plug_in', default='',
                       help='plug_in to be added to base ionic model.')
    group.add_argument('--plug_par', default='',
                       help='parameter to modified in plug-in.')
    group.add_argument('--bcl',
                       type=float, default=1000.,
                       help='Basic Cycle Length for stimulation in [ms].
    ')
    group.add_argument('--overlay', default='',
                       choices=['True','False'],
                       help='Overlays all existing experiments if True')
    #-----------------------------------------------------------------
    return parser

def jobID(args):
    today = date.today()
    return '{}_basic_{}{}{}{}'.format(today.isoformat(),args.imp,args.
    imp_par,args.plug_in,args.plug_par)

@tools.carpexample(parser, jobID)
def run(args, job):
    # define & configure EP model
    cmd = [ settings.execs.BENCH,
            '--imp={}'.format(args.imp) ]

    if args.imp_par is not '':
        cmd += [ '--imp-par', args.imp_par ]

    # define & configure Ionic plug-in
    if args.plug_in is not '':
        cmd += [ '--plug-in', args.plug_in ]

    if args.plug_par is not '':
        cmd += [ '--plug-par', args.plug_par ]

    # setup stimulus
    cmd += [ '--stim-curr', args.stim_strength,
             '--numstim',  int(float(args.duration)/float(args.bcl)+1), #
     Number of stimulus based on the simulation duration and BCL
```

```
                '--bcl', args.bcl ]

    # run bench with available ionic models
    cmd  += ['--duration', args.duration ]

    # executing bench
    job.mpi(cmd,'Running {}'.format(args.imp))

if __name__ == '__main__':
    run()
```

Tissue simulations

6.1 Running a simulation

openCARP (similar to bench) offers two ways for defining in-silico experiments: plain mode and carputils. We recommend using carputils which gives the ability to create multi-step experiments and easily share them.

6.1.1 Running in plain mode

Running a simulation in plain mode needs a parameter file.

All available option can be displayed in the terminal using:

```
> openCARP +Help
```

More verbose explanation of command line parameters is obtained by using the *+Doc* parameter which prints the same output as *+Help*, but provides more details on each individual parameter:

```
> openCARP +Doc
```

and for specific help on one parameter for example type:

```
> openCARP +Help phys_type[0].ptype
```

For a complete detailed explanation of the parameters available in openCARP please see the openCARP commands section. Down below, we describe in detail a parameter file used to simulate a bidomain problem.

```
#------------------
#Output folder name
#------------------
simID = TESTOUTPUT #Name of the output folder created when a simulation
    starts

#--------------
#Mesh basename
#--------------
meshname = testmesh #Basename of the mesh in openCARP format

#-----------------------------------------------------------------
#Definition of physics regions for each unique tagged region in the mesh
#-----------------------------------------------------------------
num_phys_regions = 2 #Total number of physical regions. In a bidomain
    simulation we have intracelluar and extracellular domain.
phys_region[0].name = "Extracellular domain"
phys_region[0].ptype = 1 #Extracellular Electrics
phys_region[0].num_IDs = 1
phys_region[0].ID[0] = 100
phys_region[1].name = "Intracellular domain"
phys_region[1].ptype = 0 #Intracellular Electrics
```

```
phys_region[1].num_IDs = 1
phys_region[1].ID[0] = 1

#-------------------------------------------------------------------
#Definition of ionic model and plugins (IMP) for each unique tagged
    region in the mesh
#-------------------------------------------------------------------
num_imp_regions = 1
imp_region[0].im = COURTEMANCHE # Ionic model proposed by Courtemanche et
     al.
imp_region[0].im_param = "g_CaL-55%,g_K1+100%,blf_i_Kur-50%,g_to-65%,g_Ks
    +100%,maxI_pCa+50%,maxI_NaCa+60%,g_Kr*1.6" #Parameters modification as
     suggested by Loewe et al. to simulate the effect of persistent atrial
     fibrillation remodel
imp_region[0].name = "AFRemodeledTissue"
imp_region[0].num_IDs = 1
imp_region[0].ID = 1

#-------------------------------------------------------------------
#Definition of gregion for each unique tagged region in the mesh
#-------------------------------------------------------------------
num_gregions = 2
gregion[0].g_il = 0.118
gregion[0].g_it = 0.118
gregion[0].g_in = 0.118
gregion[0].g_el = 0.4262
gregion[0].g_et = 0.4262
gregion[0].g_en = 0.4262
gregion[0].num_IDs = 1
gregion[0].ID = 1
gregion[1].g_bath = 0.7 #Conductivity value for an isotropic bath
gregion[1].num_IDs = 1
gregion[1].ID  = 100

#--------------------
#Definition of stimuli
#--------------------
num_stim = 1
stimulus[0].stimtype = 0
stimulus[0].strength = 120.0
stimulus[0].duration = 2.0
stimulus[0].start = 0
stimulus[0].x0 = 2100
stimulus[0].xd  = 32000
stimulus[0].y0 = 2100
stimulus[0].yd = 310
stimulus[0].z0 = 2100
stimulus[0].zd = 4200
floating_ground = 1

#-------------------------------
#Definition of general parameters
#-------------------------------
tend = 5.0 #Total time for simulation
```

```
spacedt = 1.0 #Frequency of
timedt = 1.0 #Frequency of
gridout_i = 2 #Output intracellular grid for visualizing the results of
    the intracellular domain

#-----------------------------------------------
#Definition of solving problem and numerical methods
#-----------------------------------------------
bidomain = 1
dt = 10 #Time step in microseconds
parab_options_file = parab_solve_opts #File describing the solver and its
     options
ellip_options_file = ellip_solv_opts #File describing the solver and its
    options
```

6.1.2 Running with carputils

Setting up an in-silico experiment with carputils simplifies the process and allows to create more complex experiments than in the plain mode.

This is an example of a basic structure for an experiment defined using carputils:

```
#-----------------------------------------------
#Import basic Python modules
#-----------------------------------------------
import os
from datetime import date

#-----------------------------------------------
#Import carputils Python modules
#-----------------------------------------------
from carputils import settings
from carputils import tools
from carputils import mesh
from carputils import testing

#-----------------------------------------------
#carputils parser function
#-----------------------------------------------
def parser():
    parser = tools.standard_parser()
    group = parser.add_argument_group('experiment specific options')
    group.add_argument('--ionic_model',  default='TT2',
                       choices=['TT2','COUTEMANCHE','HH'],
                       help='pick ionic model (default is TT2). For a
    full list look inside opencarp installation folder /physics/limpet/
    models')
    group.add_argument('--duration',
                       type=float, default=1000.,
                       help='Duration of simulation (ms) (default is
    1000.)')
    group.add_argument('--stim-strength',
                       type=float, default=60.,
```

```
                        help='pick transmembrane current stimulus
    strength in [uA/cm^2] (default is 60.)')
    group.add_argument('--stim-dur',
                        type=float, default=2.,
                        help='pick transmembrane current stimulus
    duration in [ms] (default is 2.)')
    return parser

#--------------------------------------------------
#carputils jobID name for output folder
#--------------------------------------------------
def jobID(args):
    today = date.today()
    return '{}_basic_{}'.format(today.isoformat(), args.duration)

#--------------------------------------------------
#carputils main function
#--------------------------------------------------
@tools.carpexample(parser, jobID)
def run(args, job):
#--------------------------------------------------
#Here goes your experiment
#--------------------------------------------------
if __name__ == '__main__':
    run()
```

Pre and post processing

openCARP provides several standalone tools for pre- and post-processing of simulations.

7.1 mesher

mesher is a program for generating simple regular FE meshes. It can also produce element tag regions and fiber definitions during element generation.

- **size:** Set the size of the mesh in each axis direction.
- **resolution:** Set the resolution of the mesh in each axis direction.
- **bath:** Set the bath size in each axis direction. Negative sizes denote a bath on both sides.
- **fibers:** Set fiber rotation.
- **mesh:** Set output mesh name.
- **regions:** Defines tags for regions in the mesh.

NOTE: The unit of the size and resolution command line parameters are in centimeters, while the unit of the mesh resolution is in micrometers.

You can query mesher by using:

```
> mesher +Help
```

7.2 igbutils

You can find igbutils inside the tools folder in the openCARP root folder. For all tools, help is output with the -h option.

7.2.1 igbhead

Perform operations on the headers of IGB files. Most commonly, it it used to print the header information but can also be usd to modify it. This utility can also be used to change the storage type of the file, or put an IGB header on to ASCII and binary data file.

7.2.2 igbapd

A simple utility to compute action potential durations.

7.2.3 igbops

This utility can perform basic math operations on one or two IGB files. There are preset functions as well as a parser for arbitrary functions. For example, one can compute the difference between two IGB files, find maxima over time, or apply a box filter.

7.2.4 igbextract

This is a tool to extract a hyperslab of data from an IGB file. The format of the output can be chosen between ASCII, binary or IGB.

Bench commands

In the section below you will find a detailed description of the commands available in bench.

8.1 Help and information

Parameters

--help	-h	Print help and exit
--detailed-help		Print help, including all details and hidden options, and exit
--full-help		Print help, including hidden options, and exit
--version	-V	Print version and exit
--list-imps		list all available IMPs (default=off)
--plugin-outputs		list the outputs of available plugins (default=off)
--imp-info		print tunable parameters and state variables for particular IMPs (default=off)

8.2 Simulation control

Parameters

--duration=DOUBLE	-a	duration of simulation
--past-stim=DOUBLE		duration after last stim (default='1000.')
--dt=DOUBLE	-D	time step (default='.01')
--num=INT	-n	number of cells (default='1')
--ext-vm-update		update Vm externally (default=off)
--rseed=INT		random number seed (default='1')

8.3 Stimulation

Parameters

--stim-curr=FLOAT	-n	stimulation current (default=‘60.0’)
--stim-volt=FLOAT		stimulation voltage
--stim-file=STRING		use signal from a file for current stim
--stim-dur=DOUBLE	-T	duration of stimulus pulse (default=‘1.’)
--stim-assign	-A	stimulus current assignment (default=off)
--stim-species=STRING		concentrations that should be affected by stimuli,&e.g. ’K$_i$: $Cl'_i(default = `K'_i)$
--stim-ratios=STRING		proportions of stimlus current carried by each species,&e.g. ’0.7:0.3’ (default=‘1.0’)
--resistance=DOUBLE		coupling resistance [kOhm] (default=‘100.0’)
--numstim=INT		number of stimuli (default=‘1’)
--stim-start=DOUBLE	-i	start stimulation (default=‘1.’)
--bcl=DOUBLE	-b	basic cycle length (default=‘1000.0’)
--stim-times=STRING		comma separated list of stim times (default=‘’)
--DIA		interpret stim times as distolic intervals (default=off)

8.4 Restitution

--restitute=STRING	restitution experiment (possible values="S1S2", "dyn", "S1S2f")
--res-file=STRING	definition file for restitution parameters (default='')
--res-trace	output ionic model trace (default=off)
--res-state-vector	save state vector (default=off)

8.5 Ionic model

Parameters

--imp=STRING	-I	IMP to use (default='MBRDR')
--imp-par=STRING	-p	params to modify IMP (default='') [+| − |/| ∗ | =][+|−][.[][e|E[−|+]][%]
--plug-in=STRING	-P	plugins to use, separate with ':' (default='')
--plug-par=STRING	-m	params to modify plug-ins, separate params with ','and plugin params with ':' (default='')
--load-module=STRING		load a module for use with bench (implies –imp) (default='')

8.6 Clamping

Parameters

--clamp=DOUBLE	-l	clamp Vm to this value (default='0.0')
--clamp-dur=DOUBLE	-L	duration of Vm clamp pulse (default='0.0')
--clamp-start=DOUBLE		start time of Vm clamp pulse (default='10.0')
--clamp-SVs=STRING		colon separated list of state variable to clamp(default='')
--SV-clamp-files=STRING		: separated list of files from which to read state variables (default='')
--SV-I-trigger		apply SV clamps at each current stim (default=on)
--AP-clamp-file=STRING		action potential trace applied at each stimulus (default='')
--clamp-ini=DOUBLE		outside of clamp pulse, clamp Vm to this value (default='-80.0')
--clamp-file=STRING		clamp voltage to a signal read from a file (default='')

8.7 Strain

Parameters

--strain=DOUBLE	-N	amount of strain to apply (default=‘0’)
--strain-time=DOUBLE	-t	time to strain (default=‘200’)
--strain-dur=FLOAT	-y	duration of strain (default=tonic) [ms]
--strain-rate=DOUBLE		time to apply/remove strain (default=‘2’)

8.8 Output

Parameters

--dt-out=DOUBLE	-o	temporal output granularity (default='1.0')
--fout[=STRING]	-O	output to files (default='BENCH_REG')
--no-trace		do not output trace (default=off)
--trace-no=INT		number for trace file name (default='0')
--bin	-B	write binary files (default=off)
--imp-sv-dump=STRING	-u	sv dump list (default='')
--plug-sv-dump=STRING	-g	: separated sv dump list for plug-ins, lists separated by semicolon (default='')
--dump-lut	-d	dump lookup tables (default=off)
--APstatistics		compute AP statistics (default=off)
--validate	-v	output all SVs (default=off)

8.9 Saving state

Parameters

--save-time=DOUBLE	-s	time at which to save binary state (default='0')
--save-file=STRING	-f	file in which to save binary state (default='a.sv')
--restore=STRING	-r	restore saved binary state file (default='a.sv')
--save-ini-file=STRING	-F	text file in which to save state of a single cell (default='singlecell.sv')
--save-ini-time=DOUBLE	-S	time at which to save single cell state (default='0.0')
--read-ini-file=STRING	-R	text file from which to read state of a single cell (default='singlecell.sv')
--SV-init=STRING		colon separated list of comma separated SV=initial_values

openCARP commands

openCARP provides a set of commands that allows you to configure your in-silico experiment. Here you can find a detailed explanation of all available parameters.

runtime_models

"openCARP offers the possibility to load custom ionic models during run-time without recompiling the simulator."

10.1 num_external_imp

Parameters

num_external_imp <Int>	(default=(Int)(0)) "Defines the number of models to read in from external libraries"

Required parameter:

external_imp

runtime_fcn

"In openCARP, you can create and use custom functions that can be loaded during run-time without recompiling the simulator."

11.1 rt_lib

Parameters

rt_lib <String>	(default=(String)(““)) “Gives a colon separated list of runtime shared object libraries“

11.2 rt_lib_args

Parameters

rt_lib_args <String>	(default=(String)(““)) “Gives colon separated list of parameters for objects“

phie_recovery

"In openCARP, you can quickly recover the extracellular potential from monodomain simulations using the parameters explained below."

12.1 phie_rec_ptf

Parameters

phie_rec_ptf <String>	(default=(String)(““)) “Defines the basename for the phie recovery file. This file specifies the phie recovery points. Use the same convention and format as for the mesh points file (see the openCARP manual file format section). Provide the filename without extension.“

12.2 phie_recovery_file

Parameters

phie_recovery_file <WFile>	(default=(WFile)(“phie_recovery“)) “Defines the file name used to output recovered extracellular potentials. Output granularity is defined by ’spacedt’.“

12.3 phie_rec_meth

Parameters

phie_rec_meth <Int>	(default=(Int)(1)) “Defines the method for recovering phie with monodomain runs.“

Parameters (continued)

Possible values are:

(Int)(3): Infrequent bidomain solve (recover phie only on phie grid, not on 'phie_rec_ptf') NOT IMPLEMENTED YET!
(Int)(2): Integrate over element-centered Im using integral solution of Poisson PDE
(Int)(1): Use FE mass and stiffness matrices
(Int)(0): Don't recover phie

12.4 dump_ecg_leads

Parameters

dump_ecg_leads <Int>

(default=(Int)(strlen(phie_rec_ptf)>0?0:1))
"Dump recovered phie data from nodes stored in 'phie_rec_ptf'.
The stated node file may contain arbitrary nodes of interest.
In the case of an ecg recording, the spatial coordinates of LA, RA, LF, V1-V6 need to be provided."

Value must be between "(Int)(0)" and "(Int)(1)"

gregions

"In this section, you can find all the information to define the conductivity of your model."

13.1 num_gregions

Parameters

num_gregions <Int>

(default=(Int)(1))
"Defines the total number of regions with respectively different conductivity settings."

Value must be greater than "(Int)(1)"

Required parameters:

gregion
gregion[PrMelem1].num_IDs
gregion[PrMelem1].name
gregion[PrMelem1].g_bath
gregion[PrMelem1].g_en
gregion[PrMelem1].g_in
gregion[PrMelem1].g_et
gregion[PrMelem1].g_it
gregion[PrMelem1].g_el
gregion[PrMelem1].g_il
gregion[PrMelem1].g_mult
gregion[PrMelem1].ID

13.2 gRegion

Description "Sets the conductivity for different regions. The array index allows enumeration of all gregions.
"

Parameters

ID <Int>

(default=(Int)(0))
"Specify a list of model regions (equivalent to mesh element tags) using this conductivity setting."

Required parameters:

gregion.num_IDs
gregion

num_IDs <Int>

(default=(Int)(0))
"Shows how many model regions use this conductivity setting"

Required parameter:

gregion

name <String>

(default=(String)(""))
"Symbolic description of this conductive region (e.g. bath, cavity etc.)"

Required parameter:

gregion

g_bath <Double>

(default=(Double)(1.))
"Defines the isotropic conductivity for non-myocardium domain, e.g blood.
The absence of anisotropy is assumed for this region. If anisotropy is required use a negative tag for the mesh elements in the .elem file and set fibre direction in the .lon file."

Value must be greater than "(Double)(0.)"

Parameters (continued)

Required parameter:

gregion

g_en <Double>

(default=(Double)(0.236))
"Defines the extracellular conductivity in the local sheet direction, i.e. perpendicular to the longitudinal fiber 'g_el' and transversal fiber directions 'g_et'."

Value must be greater than "(Double)(0.)"

Required parameter:

gregion

g_in <Double>

(default=(Double)(0.019))
"Defines the intracellular conductivity in the local sheet direction, i.e. perpendicular to the longitudinal fiber 'g_il' and transversal fiber 'g_it' directions."

Value must be greater than "(Double)(0.)"

Required parameter:

gregion

g_et <Double>

(default=(Double)(0.236))
"Defines the extracellular conductivity transverse to the fiber direction i.e. perpendicular to the longitudinal fiber 'g_il' and local sheet 'g_it' direction."

Value must be greater than "(Double)(0.)"

Required parameter:

gregion

Parameters (continued)

g_it <Double>

(default=(Double)(0.019))
"Defines the intracellular conductivity transverse to the fiber direction i.e. perpendicular to the longitudinal fiber 'g_il' and local sheet 'g_it' direction."

Value must be greater than "(Double)(0.)"

Required parameter:

gregion

g_el <Double>

(default=(Double)(0.625))
"Defines the extracellular conductivity along the fiber direction (longitudinal)."

Value must be greater than "(Double)(0.)"

Required parameter:

gregion

g_il <Double>

(default=(Double)(0.174))
"Defines the intracellular conductivity along the fiber direction (longitudinal)."

Value must be greater than "(Double)(0.)"

Required parameter:

gregion

g_mult <Float>

(default=(Float)(1.0))
"Defines the factor by which all conductivities in the region should be multiplied (after all other modifications are performed)."

Value must be greater than "(Float)(0.)"

Required parameter:

gregion

Parameters (continued)

13.3 gi_scale_vec

Parameters

gi_scale_vec <String>	(default=(String)(““)) “Path to element-wise vector with intra-cellular conductivity scaling“

13.4 ge_scale_vec

Parameters

ge_scale_vec <String>	(default=(String)(““)) “Path to element-wise vector with extra-cellular conductivity scaling“

gvecs

"In this section, you can find all the information to define global variables in your simulation."

14.1 num_gvecs

Parameters

num_gvecs <Int>

(default=(Int)(0))
"Specify the number of global state variable vector definitions."

Required parameters:

gvec
gvec[PrMelem1].bogus
gvec[PrMelem1].imp
gvec[PrMelem1].units
gvec[PrMelem1].name
gvec[PrMelem1].ID

14.2 GVecs

Description "Gives the global state variable vectors of each model region"

Parameters

bogus <Float>

(default=(Float)(0.))
"This value indicates if a state variable is not in a region"

Required parameter:

gvec

Parameters (continued)

imp <String>

(default=(String)(“”))
“Specifies a plug-in to search for state variables“

Required parameter:

gvec

units <String>

(default=(String)(“”))
“Sets the definition for the units of the state variables.“

Required parameter:

gvec

ID <String>

(default=(String)(gvec.name))
“Defines the list of state variable names in different regions, which are combined into a global vector.“

Required parameters:

gvec.name
num_imp_regions
gvec

name <WFile>

(default=(WFile)(“sv“))
“Defines the name of the global state variable vector.
This will set the name of the output file.“

Required parameter:

gvec

Parameters (continued)

conductivity_regions

"In this section, you can find all the information to define more or more ionic models for your simulation."

15.1 num_imp_regions

Parameters

num_imp_regions <Int>

(default=(Int)(1))
"Number of different region definitions."

Value must be greater than "(Int)(1)"

Required parameters:

imp_region
imp_region[PrMelem1].cellSurfVolRatio
imp_region[PrMelem1].volFrac
imp_region[PrMelem1].plug_param
imp_region[PrMelem1].plug_sv_dumps
imp_region[PrMelem1].im_sv_dumps
imp_region[PrMelem1].num_IDs
imp_region[PrMelem1].name
imp_region[PrMelem1].plugins
imp_region[PrMelem1].im_sv_init
imp_region[PrMelem1].im
imp_region[PrMelem1].im_param
imp_region[PrMelem1].ID
gvec[PrMelem1].ID

15.2 IMPregion

Description "Sets the ionic model for different regions. The array index allows enumeration of all imp_regions."

Parameters

cellSurfVolRatio <Float>

(default=(Float)(0.14))
"Defines the default single cell surface-to-volume-ratio used if it is not specified by the ionic model plugin (IMP)."

Value must be greater than "(Float)(0.)"

Required parameter:

imp_region

volFrac <Float>

(default=(Float)(1.))
"Defines the portion of the volume occupied by cells."

Value must be between "(Float)(0.)" and "(Float)(1.)"

Required parameter:

imp_region

plug_param <String>

(default=(String)(""))
"User can modify the values for various ionic model parameters.
Every PARAMETER=VALUE modification needs to be separated with a comma."

Required parameter:

imp_region

plug_sv_dumps <String>

(default=(String)(""))
"This produces a colon separated lists of state variables of a plugin to dump (use bench –imp=XXX –plug-in=YYY –imp-info to get a list of SVs for imp XXX and plug-in YYY)"

Parameters (continued)

Required parameter:

imp_region

im_sv_dumps <String>

(default=(String)(““))
“Specify a comma separated list of state variables of an ionic model to be dumped into the simulation folder.
Use bench –imp=XXX –imp-info to get a list of SVs for imp XXX.“

Required parameter:

imp_region

im_param <String>

(default=(String)(strcmp(imp_region.im,TO_STRING(DiFran
TO_STRING():TO_STRING(G_Na·3)))
“Specify a comma separated list of changes from default values, e.g. ’APDshorten·4,Gks·1.29,...’.“

Required parameters:

imp_region.im
imp_region

ID <Int>

(default=(Int)(0))
“Array of element tags associated with this model region.“

Required parameters:

imp_region.num_IDs
imp_region

Parameters (continued)

num_IDs <Int>

(default=(Int)(0))
"Number of elements tags listed in 'imp_region[].ID[]'."

Required parameter:

imp_region

name <String>

(default=(String)(strncmp(TO_STRING(imp_region),TO_STI
TO_STRING(Myocardium):TO_STRING(Purkinje)))
"Defines the symbolic name of region. This helps with structuring extensive parameter lists and makes them more human readable."

Required parameter:

imp_region

plugins <String>

(default=(String)(""))
"Specify a colon separated list of plug-ins to use with the ionic model.
Query for available plug-ins by calling 'bench –plugin-outputs'."

Required parameter:

imp_region

im_sv_init <String>

(default=(String)(""))
"Name of the file containing the initial of state variable values."

Required parameter:

Parameters (continued)

imp_region

im <String>

(default=(String)(TO_STRING(LR1)))
"Defines the ionic model to be used in the simulation.
Query for available ionic models by calling 'bench' with the command list-imps."

Required parameter:

imp_region

random

"openCARP allows modifying of the conductivity based on the element index."

16.1 rseed

Parameters

rseed <Int>	(default=(Int)(1)) "Defines the seed for the random generator"

16.2 fluct

Parameters

fluct <Float>	(default=(Float)(0.)) "Defines the conductivity fluctuation in percent." Value must be between "(Float)(0.0)" and "(Float)(100.0)"

adjust_state_variables

"openCARP offers the possibility to modify the ionic models on a nodal basis."

17.1 num_adjustments

Parameters

num_adjustments <Int>

(default=(Int)(0))
"Size of the adjustment array."

Value must be greater than "(Int)(0)"

Required parameters:

adjustment
adjustment[PrMelem1].dump
adjustment[PrMelem1].file
adjustment[PrMelem1].variable

17.2 IMPVariableAdjustment

Description "file of adjustments"

Parameters

dump <Short>

(default=(Short)(0))
"Dump nodal adjustments for state variables for display on intra grid"

Value must be between "(Short)(0)" and "(Short)(1)"

Required parameter:

adjustment

file <String>

(default=(String)(""))
"Defines the filename filled with nodal adjustments for the state variable"

Parameters (continued)

Required parameter:

adjustment

variable <String>

(default=(String)(““))
“Defines the name of the variable that should be adjusted on initialization.
The variable name should be defined as an external variable like ’Lambda’, or an ionic model variable like ’LR1.tau_f_factor’ in the cellmodel file.“

Required parameter:

adjustment

meshing

"openCARP offers the possibility to pre-process your mesh. In this section, you can find the different parameters that can help you to, for example, retag your mesh on-the-fly."

18.1 mesh_statistics

Parameters

mesh_statistics <Flag>	(default=(Flag)(PrMFALSE)) "Compute mesh statistic parameters (only edge lengths at this point)."

18.2 meshname

Parameters

meshname <String>	(default=(String)("project")) "Defines the basename for mesh files"

18.3 orthoname

Parameters

orthoname <String>	(default=(String)(meshname)) "Basename for lon file holding orthotropy data"

18.4 orthogonalize

Parameters

orthogonalize <Short>	(default=(Short)(0)) “Toggle automatic fiber orthogonalization“ Value must be between “(Short)(0)“ and “(Short)(1)“

18.5 orthoname_output

Parameters

orthoname_output <String>	(default=(String)(““)) “Basename for .lon file holding orthotropy data that is used for output purposed, e.g. strain in this fiber direction.“

18.6 meshformat

Parameters

meshformat <Short>	(default=(Short)(2)) “Mesh format ID“ **Possible values are:** (Short)(2): Auto: Use binary if possible, else use text. (Short)(1): openCARP binary format (Short)(0): openCARP text format

18.7 numtagreg

Parameters

numtagreg <Int>

(default=(Int)(0))
"Defines the number of regions to retag"

Value must be greater than "(Int)(0)"

Required parameters:

tagreg
tagreg[PrMelem1].radius
tagreg[PrMelem1].p1
tagreg[PrMelem1].p0
tagreg[PrMelem1].elemfile
tagreg[PrMelem1].no_elem_split
tagreg[PrMelem1].type
tagreg[PrMelem1].name
tagreg[PrMelem1].tag

18.8 TagRegion

Description "Defines the regions to which to assign new tags"

Parameters

radius <Float>

(default=(Float)(100.))
"Used to describe the profile chosen in tagregion.type in more detail.
- If sphere was chosen, this value is the radius of the sphere.
- If box was chosen, this value is not necessary/ignored. Use tagregion.p0 & tagregion.p1 instead.
- If cylinder was chosen, this value is the radius of the cylinder."

Required parameter:

tagreg

Parameters (continued)

p1 <Float>

(default=(Float)(0.))
"Used to describe the profile chosen in tagregion.type in more detail.
- If sphere was chosen, this value is not necessary/ignored. Use tagregion.radius instead.
- If box was chosen, this value describes the upper right corner of the box.
- If cylinder was chosen, this value describes the center of the cylinders' top.
- For sphere & cylinder the additional parameter tagregion.radius is needed.
- For box & cylinder the additional parameter tagregion.p0 is needed."

Required parameter:

tagreg

p0 <Float>

(default=(Float)(0.) (Float)(0.) (Float)(0.))
"Used to describe the profile chosen in tagregion.type in more detail.
- If sphere was chosen, this value describes the center of the sphere.
- If box was chosen, this value describes the lower left corner of the box.
- If cylinder was chosen, this value describes the center of the cylinders' base.
- For sphere & cylinder the additional parameter tagregion.radius is needed.
- For box & cylinder the additional parameter tagregion.p1 is needed."

Required parameter:

tagreg

Parameters (continued)

elemfile <RFile>

(default=(RFile)(““))
“Name of file with list of elements to be assigned to this region. The file needs the extension .regele with the format being the number of elements followed by one element number per line“

Required parameter:

tagreg

no_elem_split <Short>

(default=(Short)(0))
“Defines whether a mesh element needs to be fully enclosed by the tagreg.type definition to become part of this region.
1 = all nodes of each element need to be fully contained inside tagreg.type
0 = a single node of an element within tagreg.type is sufficient to become a member of this region“

Value must be between “(Short)(0)“ and “(Short)(1)“

Required parameter:

tagreg

type <Int>

(default=(Int)(1))
“Defines profile of region. 1-3 are predefined shapes while 4 can be used to create own region profile.“

Possible values are:

(Int)(4): element list
(Int)(3): cylinder
(Int)(2): block
(Int)(1): sphere

Required parameter:

Parameters (continued)

tagreg

name <String>

(default=(String)(““))
“Label for a region in the mesh. This helps with structuring extensive parameter lists and makes them more human readable.“

Required parameter:

tagreg

tag <Int>

(default=(Int)(321))
“New tag for a region in the mesh. Each element in this region is re-assigned this tag. In general, each element in a mesh is designated as part of a region. Electrical or mechanical properties are assigned based on region definitions.“

Required parameter:

tagreg

18.9 retagfile

Parameters

retagfile <String>

(default=(String)(““))
“Defines an output file storing the element labels after applying all ’dynamic’ tagreg choices to the input mesh.“

postprocessing

"In this section, you can find the parameters that define the different available experiments in openCARP."

19.1 ppID

Parameters

ppID <String>

(default=(String)("POSTPROC_DIR"))
"Defines the name of the output directory in post processing mode.
- It comes into play when choosing 'experiment 4'.
- If it is specified, it must not already exist.
- A folder with the default name will be created, if it is not specified."

19.2 experiment

Parameters

experiment <Short>

(default=(Short)(0))
"Defines how the simulation will be solved and what will be outputted"

Possible values are:

(Short)(4): Post process only
(Short)(3): Build model only
(Short)(2): Laplace solve
(Short)(1): Output FEM matrices
(Short)(0): NORMAL RUN

19.3 post_processing_opts

Parameters

post_processing_opts <Short>

(default=(Short)(0))
"Post-processing Options, add up option numbers to use multiple options"

Possible values are:

(Short)(1): Recover phie
(Short)(0): No post-processing done.

solution_methods

"openCARP offers the possibility to solve monodomain, pseudo-bidomain, and bidomain. In this section, you can find all the parameters related to the solving method and the solver parameters."

20.1 bidomain

Parameters

bidomain <Short>

(default=(Short)(0))
"Defines if the simulation is solved using the monodomain, bidomain or pseudo-bidomain approach.
- Monodomain model (less costly). Dervied from bidomain under the assumption that intracellular and extracellular tensors are related.
- Bidomain model solves for both the intra- & extracellular space.
- Pseudo-bidomain model (monodomain model with adjustments to account for bath loading effects)."

Possible values are:

(Short)(2): Pseudo-bidomain
(Short)(1): Bidomain
(Short)(0): Monodomain

20.2 pstrat

Parameters

pstrat <Short>

(default=(Short)(2))
"Defines the partitioning strategy."

Possible values are:

(Short)(2): KDtree partitioning
(Short)(1): Parmetis partitioning

Parameters (continued)

(Short)(0): Linear partitioning

20.3 pstrat_i

Parameters

pstrat_i <Short>

(default=(Short)(1))
"Defines the partitioning strategy for the intracellular grid

!!!! This parameter is unsupported in openCARP, but kept for compatibility with CARPentry !!!¡'

Possible values are:

(Short)(2): KDtree partitioning
(Short)(1): Parmetis partitioning
(Short)(0): Linear partitioning

20.4 pstrat_imbalance

Parameters

pstrat_imbalance <Float>

(default=(Float)(1.0001))
"Amount of imbalance to tolerate in parmetis partitioning"

20.5 ellip_solve

Parameters

ellip_solve <Short>

(default=(Short)(0))
“Defines the solving method for elliptic problems.
- Direct is typically more accurate but memory consuming.
- Iterative methods are more suitable for large problems.“

Possible values are:

(Short)(1): Iterative
(Short)(0): Direct if available

20.6 ellip_options_file

Parameters

ellip_options_file <String>

(default=(String)(““))
“File containing PETSc options for elliptic solver. For all available PETSc options refer to the PETSc documentation.“

20.7 floating_ground

Parameters

floating_ground <Short>

(default=(Short)(0))
“Enforces average extracellular potential to be zero.

!!!! This parameter is unsupported in openCARP, but kept for compatibility with CARPentry !!!¡‘

Value must be between “(Short)(0)“ and “(Short)(1)“

Parameters (continued)

20.8 floating_ground_refnode

Parameters

floating_ground_refnode <Int>

(default=(Int)(0))
"Reference node which is clamped to zero for elliptic solve prior to shifting phie average to zero.

!!!! This parameter is unsupported in openCARP, but kept for compatibility with CARPentry !!!¡'

Value must be greater than "(Int)(0)"

20.9 parab_solve

Parameters

parab_solve <Short>

(default=(Short)(1))
"Defines the solution method for the parabolic problem"

Possible values are:

(Short)(2): 2nd order dt
(Short)(1): Crank-Nicolson
(Short)(0): Explicit

20.10 theta

Parameters

theta <Float>

(default=(Float)(0.5))
"Defines the weight given to solution at t(n+1) when using Crank-Nicolson (theta) method"

Parameters (continued)

Value must be between "(Float)(0.1)" and "(Float)(0.99)"

20.11 parab_options_file

Parameters

parab_options_file <String> (default=(String)(""))
"File containing PETSc options for parabolic solver"

20.12 bidm_eqv_mono

Parameters

bidm_eqv_mono <Short> (default=(Short)(1))
"Use monodomain conductivities that are equivalent to the bidomain."

Possible values are:

(Short)(1): Use harmonic mean tensor
(Short)(0): use intracellular tensor

20.13 stimactivedelay

Parameters

stimactivedelay <Float> (default=(Float)(0.))
"Time after stimulus ends to continue FEM solve."

Parameters (continued)

20.14 vm_per_phie

Parameters

vm_per_phie <Short>	(default=(Short)(1)) "Defines the number of Vm solves per phi_e solve" Value must be between "(Short)(1)" and "(Short)(1000)"

20.15 par_fac

Parameters

par_fac <Short>	(default=(Short)(1)) "Defines the number of parabolic solves per global dt" Value must be between "(Short)(1)" and "(Short)(100)"

20.16 ode_fac

Parameters

ode_fac <Short>	(default=(Short)(1)) "Defines the number of ode solves per global dt" Value must be between "(Short)(1)" and "(Short)(10)"

20.17 extracell_monodomain_stim

Parameters

extracell_monodomain_stim <Flag>

(default=(Flag)(0))
"When bidomain mode is turned off, i.e. '-bidomain=0', set phi_e to be utterly determined by the extracellular stimuli."

20.18 cg_tol_ellip

Parameters

cg_tol_ellip <Double>

(default=(Double)(1.0e-8))
"conjugate gradient solver tolerance for elliptic problem"

20.19 cg_norm_ellip

Parameters

cg_norm_ellip <Short>

(default=(Short)(0))
"Pick a norm for checking convergence of elliptic solve for PETSc solvers"

Possible values are:

(Short)(3): combined tolerance, using both 0 and 2, iteration stops if either 0 or 2 are met
(Short)(2): relative tolerance
(Short)(1): absolute tolerance using L2 of unpreconditioned residual (not always possible, defaults to 0 then)
(Short)(0): absolute tolerance using L2 of preconditioned residual (energy norm)

Parameters (continued)

20.20 cg_maxit_ellip

Parameters

cg_maxit_ellip <Int>

(default=(Int)(500))
"Defines the maximum number of iterations for iterative solver of elliptic PDE."

Value must be between "(Int)(0)" and "(Int)(10000)"

20.21 ellip_use_pt

Parameters

ellip_use_pt <Short>

(default=(Short)(0))
"Choose solvers for the elliptic PDE from:
- PETSc, then set to 0, or
- PT, then set to 1."

Value must be between "(Short)(0)" and "(Short)(1)"

20.22 cg_tol_parab

Parameters

cg_tol_parab <Double>

(default=(Double)(1.0e-8))
"conjugate gradient solver tolerance for parabolic problem"

Parameters (continued)

20.23 cg_norm_parab

Parameters

cg_norm_parab <Short>

(default=(Short)(0))
"Pick a norm for checking convergence of parabolic solve (PETSc solvers only, ignored with PT)"

Possible values are:

(Short)(3): combined tolerance, using both 0 and 2, iteration stops if either 0 or 2 are met
(Short)(2): relative tolerance
(Short)(1): absolute tolerance using L2 of unpreconditioned residual (not always possible, defaults to 0 then)
(Short)(0): absolute tolerance using L2 of preconditioned residual (energy norm)

20.24 cg_maxit_parab

Parameters

cg_maxit_parab <Int>

(default=(Int)(100))
"maximum number of iterations for iterative solver of parabolic PDE.
"

Value must be between "(Int)(0)" and "(Int)(1000)"

20.25 parab_use_pt

Parameters

parab_use_pt <Short>

(default=(Short)(0))
"Choose solvers for the parabolic PDE from:
- PETSc, then set to 0, or
- PT, then set to 1."

Value must be between "(Short)(0)" and "(Short)(1)"

20.26 cg_precond

Parameters

cg_precond <Int>

(default=(Int)(2))
"Defines the preconditioning method to ensure fast convergence of the conjugate gradient method."

Possible values are:

(Int)(3): System reduction
(Int)(2): Incomplete Cholesky
(Int)(1): diagonal
(Int)(0): none

fe_methods

"You can select different methods to solve the finite element method."

21.1 mat_entries_per_row

Parameters

mat_entries_per_row <Short>

(default=(Short)(27))
"Assumed average number of entries per PETSc matrix row timportant for memory preallocation.

!!!! This parameter is unsupported in openCARP, but kept for compatibility with CARPentry !!!¡'

21.2 mass_lumping

Parameters

mass_lumping <Short>

(default=(Short)(1))
"toggles mass matrix lumping"

Possible values are:

(Short)(1): Lump mass matrix
(Short)(0): Use full mass matrix

21.3 operator_splitting

Parameters

operator_splitting <Short>

(default=(Short)(1))
"toggles operator splitting"

Possible values are:

Parameters (continued)

(Short)(1): Use operator splitting
(Short)(0): Don't use operator splitting

stimulation

"openCARP offers you the possibility to define one or more stimuli. Different stimuli parameters can specify in the simulation (for example, strength, duration, BCL, and more)."

22.1 num_stim

Parameters

num_stim <Int>

(default=(Int)(2))
"Defines the number of stimuli."

Value must be greater than "(Int)(0)"

Required parameters:

stim
stimulus
stim[PrMelem1].crct
stim[PrMelem1].elec
stim[PrMelem1].ptcl
stim[PrMelem1].pulse
stim[PrMelem1].name
stimulus[PrMelem1].vtx_fcn
stimulus[PrMelem1].data_file
stimulus[PrMelem1].pulse_file
stimulus[PrMelem1].total_current
stimulus[PrMelem1].balance
stimulus[PrMelem1].stimtype
stimulus[PrMelem1].bias
stimulus[PrMelem1].tau_plateau
stimulus[PrMelem1].tau_edge
stimulus[PrMelem1].s2
stimulus[PrMelem1].strength
stimulus[PrMelem1].d1
stimulus[PrMelem1].start
stimulus[PrMelem1].geometry
stimulus[PrMelem1].ctr_def
stimulus[PrMelem1].zd
stimulus[PrMelem1].yd
stimulus[PrMelem1].xd
stimulus[PrMelem1].z0

Parameters (continued)

stimulus[PrMelem1].y0
stimulus[PrMelem1].x0
stimulus[PrMelem1].dump_vtx_file
stimulus[PrMelem1].vtx_file
stimulus[PrMelem1].name
stim[PrMelem1].crct.total_current
stim[PrMelem1].crct.balance
stim[PrMelem1].crct.type
stim[PrMelem1].elec.geom_type
stim[PrMelem1].elec.radius
stim[PrMelem1].elec.p1
stim[PrMelem1].elec.p0
stim[PrMelem1].elec.vtx_fcn
stim[PrMelem1].elec.vtx_file
stim[PrMelem1].elec.domain
stim[PrMelem1].elec.geomID
stim[PrMelem1].ptcl.stimlist
stim[PrMelem1].ptcl.start
stim[PrMelem1].ptcl.name
stim[PrMelem1].pulse.bias
stim[PrMelem1].pulse.tau_plateau
stim[PrMelem1].pulse.tau_edge
stim[PrMelem1].pulse.s2
stim[PrMelem1].pulse.tilt_ampl
stim[PrMelem1].pulse.tilt_time
stim[PrMelem1].pulse.strength
stim[PrMelem1].pulse.file
stim[PrMelem1].pulse.shape
stim[PrMelem1].pulse.name
stimulus[PrMelem1].duration
stim[PrMelem1].ptcl.duration
stimulus[PrMelem1].bcl
stim[PrMelem1].ptcl.bcl
stimulus[PrMelem1].npls
stim[PrMelem1].ptcl.npls

22.2 Stimulus

Description “Array of stimuli“

Parameters

vtx_fcn <Short>

(default=(Short)(0))
“Set true to specify stimulation strengths on a nodal basis. The specific values need to be provided in the stimulus.vtx_file. For file format specifications check the openCARP manual.

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!¡‘

Required parameter:

stimulus

data_file <RFile>

(default=(RFile)(“”))
“Stimulus dependent auxiliary data. Used for presribed takeoff and prescribed phie.

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!¡‘

Required parameter:

stimulus

pulse_file <RFile>

(default=(RFile)(“”))
“Reads in pulse definition from an external file. For file format specifications check the openCARP manual.

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!¡‘

Required parameter:

stimulus

Parameters (continued)

total_current <Short>

(default=(Short)(0))
“Treat strengths as total current (uA) and not density

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!¡‘

Value must be between “(Short)(0)“ and “(Short)(1)“

Required parameter:

stimulus

balance <Int>

(default=(Int)(-1))
“Defines whether the electrode is balancing another electrode. The balance value is interpreted as the electrode index to balance. The waveform is mirrored, but with opposite polarity.
If the balance value is -1, no balancing is applied.

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!¡‘

Value must be between “(Int)(-1)“ and “(Int)(num_stim-1)“

Required parameters:

num_stim
stimulus

Parameters (continued)

stimtype <Short>

(default=(Short)(0))
"Defines the stimulus type. Closed loop stimuli return to ground (0 mV) when expired.
Open loop stimuli are removed entirely when expired.

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!¡'

Possible values are:

(Short)(10): prescribed phie (use mV)
(Short)(9): Vm clamp (use mV)
(Short)(8): prescribed takeoff (eikonal driven, prescribe LAT vectors)
(Short)(5): extracellular voltage (open loop, use mV)
(Short)(4): intracellular current
(Short)(3): extracellular ground (enforce 0 mV)
(Short)(2): extracellular voltage (closed loop, use mV)
(Short)(1): extracellular current (use uA/cm^3)
(Short)(0): transmembrane current (use uA/cm^2)

Required parameter:

stimulus

bias <Float>

(default=(Float)(0.0))
"Defines a constant term which is added to the stimuluspulse.

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!¡'

Required parameter:

stimulus

Parameters (continued)

tau_plateau <Float>

(default=(Float)(1000.))
“Defines a time constant governing plateau of pulse (>10ˆ5 is infinite)
A larger tau_plateau will result in a longer plateauphase.
Other important parameters for stimulus form definition are: stimulus.tau_edge, stimulus.tau_plateau, stimulus.strength, stimulus.duration

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!¡‘

Value must be between “(Float)(0.1)“ and “(Float)(1000000.)“

Required parameter:

stimulus

tau_edge <Float>

(default=(Float)(0.01))
“Defines the time constant governing leading and trailing edge of pulse. The formulation used resembles the equation: Amp · (1 - exp(-t/tau_edge)) · exp(-t/tau_plateau).
A larger tau_edge will result in a fast drop, while a smaller tau_edge will have a longer exponentialy decreasing flank.
Other important parameters for stimulus form definition are: stimulus.tau_edge, stimulus.tau_plateau, stimulus.strength, stimulus.duration

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!¡‘

Value must be between “(Float)(0.)“ and “(Float)(1000.)“

Required parameter:

Parameters (continued)

stimulus

s2 <Float>

(default=(Float)(0.))
"This value is only used for defining a biphasic pulse.
This value is a relative value and defines the stimulus strength of the trailing pulse (after zero crossing) relative to leading pulse (from start to zero crossing).
Giving the value 0 will result in a flip of polarity
Other important parameters for stimulus form definition are: stimulus.tau_edge, stimulus.tau_plateau, stimulus.strength, stimulus.duration

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!¡'

Value must be between "(Float)(0.)" and "(Float)(10.)"

Required parameter:

stimulus

strength <Float>

(default=(Float)(0.))
"Defines strength of prescribed stimulation. Strength is defined as amplitude of signal. If dealing with currents it is defined as uA/volume. If dealing with voltages it is defined as mV.
To create a stimulation the minimum requirement is a given stimulus.duration and stimulus.strength. This creates a monophasic stimulus. For biphasic stimuli check stimulus.s2 and stimulus.d1.
For more advanced pulsesignals introduce them using a pulse file with stimulus.pulse_file

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!¡'

Parameters (continued)

Required parameter:

stimulus

d1 <Float>

(default=(Float)(1.0))
"This parameter is used to introduce biphasic stimulation pulses. It specifies the duration of the first part of the pulse relative to the duration of the entire pulse (1.0 = monophasic).
Further parameters to define biphasic stimuli are: stimulus.strength, stimulus.s2, stimulus.tau_edge and stimulus.tau_plateau

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!¡'

Value must be between "(Float)(0.)" and "(Float)(1.0)"

Required parameter:

stimulus

duration <Float>

(default=(Float)(tend-stimulus.start))
"Defines the duration of one a single stimulation event.

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!¡'

Value must be between "(Float)(0.)" and "(Float)(tend-stimulus.start)"

Required parameters:

stimulus.start
tend
stimulus

Parameters (continued)

npls <Int>

(default=(Int)(tend > 0 ? 1 : 0))
"Defines the number of pulses in the stimulation protocol. The period of pulses can be set with bcl in ms.
If set to 0, the stimulus will have a single instance.

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!¡'

Value must be between "(Int)(0)" and "(Int)(tend > 0 ? 1+tend/stimulus.bcl : 0)"

Required parameters:

tend
stimulus.bcl
stimulus

bcl <Float>

(default=(Float)(tend-stimulus.start))
"Defines the basic cycle length for repetitive stimulation.
Duration must be smaller than bcl, as it specifies the duration of a single stimulation event.
The full protocol duration is npls · bcl. This is derived automatically from the user input.

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!¡'

Value must be between "(Float)(stimulus.duration)" and "(Float)(tend-stimulus.start)"

Required parameters:

stimulus.start
tend
stimulus.duration
stimulus

Parameters (continued)

start <Double>

(default=(Double)(0.))
“Defines the start time when the stimulationpulse is introduced

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!¡‘

Value must be between “(Double)(0.)“ and “(Double)(tend)“

Required parameters:

tend
stimulus

geometry <Int>

(default=(Int)(-1))
“Refers to a region ID to be used as geometry definition for the stimulus electrode

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!¡‘

Value must be greater than “(Int)(-1)“

Required parameter:

stimulus

ctr_def <Flag>

(default=(Flag)(0))
“setting this to 1 yields limits in the form of [x0-xd/2,x0+xd/2] for x,y and z. Setting to 0 yields [x0,x0+xd] for x,y and z.

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!¡‘

Required parameter:

stimulus

Parameters (continued)

zd <Float>

(default=(Float)(cell_length))
"z dimension of electrode volume. This will give the limits [z0,z0+zd]. To change this to [z0-zd/2,z0+zd/2] use stimulus.ctr_def

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!¡'

Value must be greater than "(Float)(0.)"

Required parameters:

cell_length
stimulus

yd <Float>

(default=(Float)(cell_length))
"y dimension of electrode volume. This will give the limits [y0,y0+yd]. To change this to [y0-yd/2,y0+yd/2] use stimulus.ctr_def

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!¡'

Value must be greater than "(Float)(0.)"

Required parameters:

cell_length
stimulus

xd <Float>

(default=(Float)(cell_length))
"x dimension of electrode volume. This will give the limits [x0,x0+xd]. To change this to [x0-xd/2,x0+xd/2] use stimulus.ctr_def

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!¡'

Value must be greater than "(Float)(0.)"

Required parameters:

Parameters (continued)

cell_length
stimulus

z0 <Float>

(default=(Float)(0.))
“Lower z ordinate of electrode volume (zd defines the spatial electrode extension in z)

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!¡‘

Required parameter:

stimulus

y0 <Float>

(default=(Float)(0.))
“Lower y ordinate of electrode volume (yd defines the spatial electrode extension in y)

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!¡‘

Required parameter:

stimulus

x0 <Float>

(default=(Float)(0.))
“Lower x ordinate of electrode volume (xd defines the spatial electrode extension in x)

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!¡‘

Required parameter:

stimulus

Parameters (continued)

dump_vtx_file <Short>

(default=(Short)(0))
“For volume based electrode definitions, vertices of this electrode are dumped to a file. The output file name is electrode_num.stim

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!¡‘

Value must be between “(Short)(0)“ and “(Short)(1)“

Required parameter:

stimulus

vtx_file <RFile>

(default=(RFile)(““))
“File name allowing a vertex based electrode definition. Using a non-empty string switches to vertex-based definition and ignores other settings. For file format specifications check openCARP manual.

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!¡‘

Required parameter:

stimulus

name <String>

(default=(String)(““))
“Definition of the label for a single electrode.

!!!! The stimulus[] parameter was declared legacy. Please use stim[] !!!¡‘

Required parameter:

Parameters (continued)

stimulus

22.3 Stim

Description “Definition of stimuli“

Parameters

crct <(null)>

“Definition of setup and wiring of stimulation circuitry“

Required parameter:

stim

elec <(null)>

“Definition of electrode geometry“

Required parameter:

stim

ptcl <(null)>

“Definition of stimulation protocol“

Required parameter:

stim

pulse <(null)>

“Definition of stimulation pulse“

Parameters (continued)

	Required parameter: stim
name <String>	(default=(String)(““)) “Definition of the label for a single electrode“
	Required parameter: stim

output

"In this section, you can find the parameters related to all the output files of a simulation. Moreover, you can also find parameters that will modify the output in the terminal."

23.1 simID

Parameters

simID <String>

(default=(String)("OUTPUT_DIR"))
"Defines the Simulation ID to generate output directory.
A good practice is to include date and time of the simulation in the 'simID'to avoid overwriting simulations. If 'simID' already exists, e.g the directory was created in a previous run, the user needs to decide on how to proceed (overwrite, append, abort) the simulation."

23.2 dt

Parameters

dt <Double>

(default=(Double)(5.))
"Defines the time step size to solve the numeric equations for.
Check the openCARP manual for a comprehensive explanation on how to choose 'dt'. "

Value must be greater than "(Double)(0.)"

23.3 tend

Parameters

tend <Double>

(default=(Double)(100.))
"Defines the point in time when the simulation stops.
To get a rough estimation for 'tend', multiply the number of stimulation pulses 'stimulation.npls' with the basic cycle length 'stimulation.bcl'. 'tend' then needs to be larger than 'stimulation.npls · stimulation.bcl' to cover your entire stimulation protocol."

Value must be greater than "(Double)(dt/1000.)"

23.4 buildinfo

Parameters

buildinfo <Flag>

(default=(Flag)(PrMFALSE))
"1 = enabled: prints build information about this executable.
0 = disabled: build informations stays hidden."

23.5 output_level

Parameters

output_level <Int>

(default=(Int)(1))
"Defines the level of verbosity [0, 10] to the terminal output.
0 is considered minimal feedback to the user on the terminal."

Value must be between "(Int)(0)" and "(Int)(10)"

23.6 dump2MatLab

Parameters

dump2MatLab <Flag>

(default=(Flag)(0))
“Dumps stiffness and mass matrices, mappings and stimulation vectors
to be used with MatLab to the simulation folder“

23.7 dump_basename

Parameters

dump_basename <String>

(default=(String)(“MatLabDump“))
“Defines the basename for dumped files. Different endings will be attached to specify different variables:
- _Ki & _Kie ... for the intra- & extracellular stiffness matrices
- _Mi & _Me ... for the intra- & extracellular mass matrices
- _i2e & _e2i ... for the intracellular-to-extracellular and vice-versa mappings
- _Itr & _Ie ... for the transmembrane- & extracellular currents“

23.8 vofile

Parameters

vofile <WFile>

(default=(WFile)(“vm“))
“IGB formatted file of transmembrane voltages.“

Parameters (continued)

23.9 phiefile

Parameters

phiefile <WFile>	(default=(WFile)(“phie“)) “IGB formatted file of extracellular potential (phi).“

23.10 phieifile

Parameters

phieifile <WFile>	(default=(WFile)(“phie_i“)) “IGB formatted file of extracellular potential (phie) on intracellular grid (only in presence of bath required).“

23.11 gridout_i

Parameters

gridout_i <Int>	(default=(Int)(0)) “Defines if intracellular grid is outputted in simulation directory. Settings: 0=none, 1=surface, 2=volumetric mesh, 3=surface & volumetric mesh. “ Value must be greater than “(Int)(0)“

23.12 gridout_e

Parameters

gridout_e <Int>

(default=(Int)(0))
“Defines if extracellular grid is outputted in simulation directory.
Settings: 0=none, 1=surface, 2=volumetric mesh, 3=surface & volumetric mesh.
“

Value must be greater than “(Int)(0)“

23.13 gridout_p

Parameters

gridout_p <Int>

(default=(Int)(0))
“If not 0, writes partition index of each element as .dat file in simulation directory.“

23.14 spacedt

Parameters

spacedt <Double>

(default=(Double)(3.))
“Defines the temporal interval to output data to files.
It can only be as small as ’dt/1000’.
For long simulations outputting every single calculated value, would yield terrabytes of data. So here you can reduce the outputted values.“

Value must be between “(Double)(dt/1000.)“ and “(Double)(tend)“

23.15 timedt

Parameters

timedt <Double>

(default=(Double)(1.))
"Defines the temporal interval between progress updates made to the terminal. (For informational purposes only)."

Value must be between "(Double)(dt/1000.)" and "(Double)(tend)"

23.16 spacetol

Parameters

spacetol <Float>

(default=(Float)(100.))
"Defines the spatial tolerance when searching for a node."

Value must be greater than "(Float)(0.)"

23.17 display_meminfo

Parameters

display_meminfo <Short>

(default=(Short)(0))
"Turns on/off displaying memory malloc info."

Possible values are:

(Short)(1): Turn on meminfo output
(Short)(0): Turn off meminfo output

cell_size

"Defines the length of the cell used in the model."

24.1 cell_length

Parameters

cell_length <Float>	(default=(Float)(100.)) "Defines the default cell length." Value must be greater than "(Float)(0.)"

savestate

"Defines all variables to save and read the simulated system states and therefore the simulation progress. Additionally allows for interval saving of simulation and reserving queue time in batch processing."

25.1 num_tsav

Parameters

num_tsav <Int>

(default=(Int)(0))
"Defines the number of states to be saved. Each time instant of the simulation progression needs to be specified in 'tsav'."

Value must be between "(Int)(0)" and "(Int)(50)"

Required parameters:

tsav
tsav_ext

25.2 write_statef

Parameters

write_statef <WFile>

(default=(WFile)("state"))
"Defines the basename of the output file in which to write a single saved state. Each saved state will have a timestamp appended to the basename."

25.3 start_statef

Parameters

start_statef <RFile> (default=(RFile)(““))
“Loads a statefile and continues the simulation from there.“

25.4 chkpt_start

Parameters

chkpt_start <Float> (default=(Float)(0.))
“Defines the start time to trigger interval-based checkpointing (saving of the simulation progress).“

Value must be between “(Float)(0.)“ and “(Float)(tend)“

25.5 chkpt_intv

Parameters

chkpt_intv <Float> (default=(Float)(0.))
“Defines the interval for checkpointing (saving of the simulation progress) in ms. 0 means no checkpointing.“

Value must be between “(Float)(0.)“ and “(Float)(tend)“

25.6 chkpt_stop

Parameters

chkpt_stop <Float>	(default=(Float)(tend)) "Defines the time to stop interval-based checkpointing (saving of the simulation progress)." Value must be between "(Float)(0.)" and "(Float)(tend)"

25.7 queue_time

Parameters

queue_time <Float>	(default=(Float)(168.)) "Defines the reserved queue time when running in batch mode" Value must be between "(Float)(0.)" and "(Float)(168.)"

25.8 shrimp_pipe

Parameters

shrimp_pipe <String>	(default=(String)("")) "Defines the path to file where info should be dumped when openCARP receives SIGIO (in general, this should be a named pipe)"

25.9 state_dump_buffer

Parameters

state_dump_buffer <Long>	(default=(Long)(100000000)) "Defines the dump buffer size"

Parameters (continued)

	Value must be greater than “(Long)(1000)“

activation

"In openCARP, the user can define several methods to detect an activation threshold in tissue simulations. It also offers several functions to start a simulation based on LATs, stop or restart a simulation. Additionally, you can obtain APD statics as an output file."

26.1 num_LATs

Parameters

num_LATs <Int>

(default=(Int)(0))
"Defines the number of local activation measurements."

Value must be greater than "(Int)(0)"

Required parameters:

lats
lats[PrMelem1].measurand
lats[PrMelem1].all
lats[PrMelem1].mode
lats[PrMelem1].threshold
lats[PrMelem1].method
lats[PrMelem1].ID

26.2 prepacing_lats

Parameters

prepacing_lats <RFile>

(default=(RFile)(""))
"Tissue activation times to guide state set-up for prepacing"

26.3 prepacing_beats

Parameters

prepacing_beats <Int> (default=(Int)(0))
"Defines the number of beats to pre-pace using a single-cell model.
Prepacing of a single cell is used to put the single-cell models into a preconditioned state.
This state is then given all cells in a tissue simulation as a better initial starting point than a completly unstimulated cell"

26.4 prepacing_bcl

Parameters

prepacing_bcl <Float> (default=(Float)(0.))
"Sets the basic cycle length for the prepacing stimulus."

26.5 LAT

Description "Array containing LATs. Index of array corresponds to LAT measurement."

Parameters

ID <WFile> (default=(WFile)(lats.measurand ? LAT_ID[1] : LAT_ID[0]))
"Defines the output filename."

Required parameters:

lats.method
lats.measurand
lats

Parameters (continued)

measurand <Int>

(default=(Int)(0))
"Defines the quantity to measure, e.g. the signal (transmembrane voltage or extracellular potential) to use the thresholds on."

Possible values are:

(Int)(1): Phie
(Int)(0): Vm

Required parameter:

lats

all <Int>

(default=(Int)(1))
"Defines that all activations should be detected (1) or only the first one (0). Detecting 'all' is the default."

Value must be between "(Int)(0)" and "(Int)(1)"

Required parameter:

lats

mode <Short>

(default=(Short)(0))
"Toggles between detecting max derivative or positive(+) slope threshold crossing and detecting minimum derivative and negative(-) slope threshold crossing."

Possible values are:

(Short)(1): detect min derivative or -slope threshold crossing
(Short)(0): detect max derivative or +slope threshold crossing

Parameters (continued)

Required parameter:

lats

threshold <Float>

(default=(Float)(-10.))
"Defines the crossing threshold (for method 1) or maximum derivative threshold (for method 2)"

Required parameter:

lats

method <Int>

(default=(Int)(1))
"Describes the method used to determine the instant of local activation. Define the threshold using structure.threshold. Choose if you want to evaluate during the rising or falling slope of the signal using structure.mode."

Possible values are:

(Int)(2): instant of maximum derivative (do not use, not implemented yet)
(Int)(1): instant of threshold crossing

Required parameter:

lats

26.6 t_sentinel

Parameters

t_sentinel <Float>	(default=(Float)(-1.)) "Sentinel checks for activations, based on LATs. If none are found, exits simulation, else continues. t_sentinel should always be >0 . t_sentinel describes the time sentinel is run for from the t_sentinel_start time. If during this time period no lats[] are detected, savequit() cleanly."

26.7 t_sentinel_start

Parameters

t_sentinel_start <Float>	(default=(Float)(0.)) "Defines the time instant when checking for quiescence is started"

26.8 sentinel_ID

Parameters

sentinel_ID <Int>	(default=(Int)(-1)) "Sentinel will check the LAT ID specified here as a reference to quit or continue the simulation." Value must be smaller than "(Int)(num_LATs-1)"

26.9 compute_APD

Parameters

compute_APD <Flag>	(default=(Flag)(PrMFALSE)) "Defines if the actionpotential duration should be computed - If is set to 1 = computes action potential durations - If is set to 0 = action potential durations are not calculated"

26.10 actthresh

Parameters

actthresh <Float>	(default=(Float)(30.)) "Defines the threshold to determine if element was activated, e.g the threshold where an action potential was triggered. The magnitude is used from the signal to be thresholded."

26.11 recovery_thresh

Parameters

recovery_thresh <Float>	(default=(Float)(-60.)) "Defines the threshold to determine if element wich was activated recovered back to its steadystate. The magnitude is used from the signal to be thresholded."

phys_regions

"openCARP is designed to support multi-physics from the ground up. Therefore, the mesh regions are assigned to different physics. For simple EP experiments on the whole mesh without bath, no additional input w.r.t. CARPentry is required, as all regions are assigned by default to the Intracellular and Extracellular domains. With a bath present, the user needs to inform the simulator which regions form which simulation domains."

27.1 num_phys_regions

Parameters

num_phys_regions <Int>

(default=(Int)(0))
"The number of physics regions"

Value must be greater than "(Int)(0)"

Required parameters:

phys_region
phys_region[PrMelem1].num_IDs
phys_region[PrMelem1].name
phys_region[PrMelem1].ptype
phys_region[PrMelem1].ID

27.2 p_region

Description "Array containing the defined physics regions."

Parameters

ID <Int>

(default=(Int)(0))
"Define a list of tags forming a mesh region."

Required parameters:

phys_region.num_IDs
phys_region

Parameters (continued)

num_IDs <Int>

(default=(Int)(0))
"Defines the number of IDs (equivalent to element tags) listed under 'ID'."

Required parameter:

phys_region

name <String>

(default=(String)(""))
"Symbolic name for the physics region"

Required parameter:

phys_region

ptype <Int>

(default=(Int)(0))
"Defines the type of physics."

Possible values are:

(Int)(1): Extracellular Electrics
(Int)(0): Intracellular Electrics

Required parameter:

phys_region

(null)