ICED User Guide
Version 1.0.0

August 24, 2015

www.illinoisrocstar.com

License

ICED sources, executables, and this document are the property of Illinois Rocstar LLC. Licensing and support of the software package, including full source access for government, industrial, and academic partners, are arranged on an individual basis. Please contact Illinois Rocstar at

- **tech@illinoisrocstar.com**
- **sales@illinoisrocstar.com**

for support and licensing.

1 Introduction

Lithium-ion batteries are making significant impacts on a range of applications. The objective of the present effort is to develop models at the electrode–particle scale that incorporate fundamental thermodynamics and chemical kinetics. The software discussed herein is based on the foundation established by Newman and coworkers [2, 4, 3] and solves a tightly-coupled set of electrochemical equations. It provides a modular infrastructure that can be easily extended to provide the user with a desired level of fidelity by selecting from a set of physics modules. Additionally, the software provides the flexibility for users to add their own physics modules specific to the physical phenomena in which they are interested.

Species- and charge-conservation equations are solved to predict temporal and spatial profiles of lithium concentration and electric potentials. Transport through the composite electrodes is modeled using porous electrode theory. Thus, the electrolyte and active electrode material are treated as a superimposed continua [2]. Concentrated solution theory is used to describe transport within the binary liquid electrolyte.

Consider a real system, such as the battery shown in Figure 1. At the electrode–particle scale, the three main components are visible: a composite anode on the left, a separator in the center, and a composite cathode on the right.

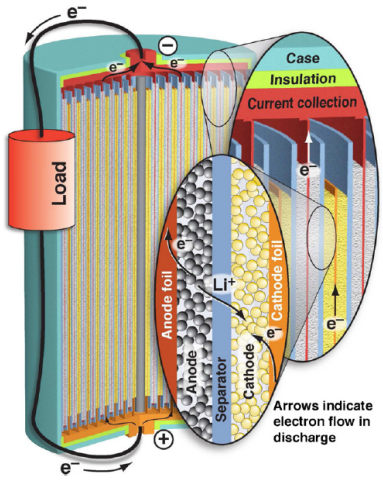

Figure 1: Schemata of the structure of a cylindrical Li ion battery, showing three length scales: device scale, cell scale and electron–particle scale. At the electron–particle scale the composite electrodes and the separator are most easily observable. Figure from [1].

The composite electrodes consist of active particles, binder, and conductive connectors, all of which are surrounded by the lithium ion-conducting electrolyte phase. Lithiated graphite (Li_xC_6, where $0.1 < x < 0.8$) is commonly used as the active material in the anode (negative electrode). The cathode (positive electrode) active material is often a metal oxide (e.g., Li_yCoO_2 where $0.4 < y < 1.0$). A microporous separator film electrically isolates the cathode from the anode. The electrolyte phase is contained within the pores of the separator, which enables ion conduction across the separator.

During discharge intercalated* Li leaves anode particles and enters the electrolyte as Li^+. This charge-transfer process delivers electrons to a current collector and an external load. The Li ions are transported via diffusion and migration through the electrically insulating electrolyte solution toward the cathode. Reacting with electrons from the external load, a charge-transfer process delivers Li into the cathode particles. During charging the processes are reversed.

The reversible charge transfer process at the graphite anode may be represented globally as:

$$\mathrm{Li}(\mathrm{C}_6) \Longleftrightarrow (\mathrm{C}_6)_{vac} + \mathrm{Li}_e^+ + \mathrm{e}_{an}^- \tag{1}$$

where $\mathrm{Li}(\mathrm{C}_6)$ are the intercalated Li in graphite, Li_e^+ are the positively charged Li ions in the electrolyte phase, $(\mathrm{C}_6)_{vac}$ are the intercalation vacancies in within the graphite, and e_{an}^- are the electrons in the anode phase, where the anode phase consists of graphite, conductive filler surrounding the graphite, and the current collector.

The reversible charge transfer process at the cathode (e.g., CoO2) may be represented globally as:

$$(\mathrm{CoO}_2)_{vac} + \mathrm{Li}_e^+ + \mathrm{e}_{cd}^- \Longleftrightarrow \mathrm{Li}(\mathrm{CoO}_2) \tag{2}$$

where $(\mathrm{CoO}_2)_{vac}$ are the intercalation vacancies within the cathode (metal oxide) particles, Li_e^+ are the positively charged Li ions in the electrolyte phase, e_{cd}^- are the electrons in the electronically conducting phases of the cathode, and $\mathrm{Li}(\mathrm{CoO}_2)$ are the intercalated Li in the metal oxide.

2 Capabilities

Application of Newman's P2D model in *ICED* is similar to those discussed elsewhere and is summarized here for convenience [[2]]. Transport in the separator is modeled with concentrated solution theory and assumes a binary electrolyte in a single-phase solvent. Transport through the composite electrodes is modeled using porous electrode theory where the electrodes and active material are treated as a superimposed continua, one representing the solution and another representing the matrix.

Composite electrodes are composed of porous matrices of a single reactive electronic conductor (or mixtures of solids) that include both non-conductive reactive materials in addition to the electronic conductors. In this macroscopic treatment, the actual geometric details of the pores are disregarded. Then, a potential is defined both in the solid, conducting material, ϕ_s, and in the pore-filling electrolyte, ϕ_e. The concentration of lithium ions in the two phases is similarly defined as c_s or c_e, representing either the solid or electrolyte phase, respectively. The volume fraction of the electrolyte phase is represented as ϵ. This leads to the first equation, the time-dependent diffusion of lithium ions in the electrolyte, as represented by Equation 3.

*Intercalation or insertion: An ion from the solution enters into the crystal lattice of a solid.

$$\epsilon \frac{\partial c}{\partial t} = \nabla \cdot (\epsilon D \nabla c) - [\kappa \nabla \phi_e + \kappa_D \nabla \ln c] \cdot \frac{\nabla t^0}{z\nu\mathcal{F}} + \frac{a j_n (1 - t^0)}{\nu} \tag{3}$$

$$\kappa_D = \frac{\kappa \mathcal{R} T}{\mathcal{F}} \left(1 + \frac{\partial \ln f_\pm}{\partial \ln c}\right) \left(\frac{s}{n\nu} + \frac{t^0}{z\nu}\right) \tag{4}$$

Equation 3 is dependent on three transport properties that may be functions of the ion concentration.

ionic conductivity (κ) Measure of a solution's ability to conduct electricity

diffusion coefficient (D) Measure of "how quickly" an ion moves through solution

transference number (t^0) Fraction of current carried by an ion in a solution (of uniform composition)

Also included in this equation is the transfer rate of lithium ions to or from the matrix (j_n) that is solved for in a separate equation. This term is multiplied by the average interfacial area (a). The other unknowns in this equation are the number of cations or anions into which one mole of electrolyte dissociates (ν), the charge number (z), and Faraday's constant ($\mathcal{F}$). These terms are dependent on the electrochemical reactions taking place and are not functions of ion concentration.

Equation 4 includes contributions from the activity coefficient ($f_\pm$), the stoichiometric coefficient in the electrode reaction (s), the number of electrons transferred (n), the temperature (T), and the universal gas law constant ($\mathcal{R}$). The parameter κ_D comes from the derivation of the current density and determines the electrolyte's ability to hold a charge.

The right hand side of Equation 3 defines our first three modules. The first term models the motion of lithium ions due to concentration gradient. This term is solved in the nonlinear diffusion module. The second term represents the movement of lithium ions due to gradients in current density. This effect is reproduced by the ion migration module. The third term represents the transfer of lithium ions to or from the solid matrix. This is reproduced by the wall flux module.

Charge migration in the two phases is represented by the two following equations. First, Ohm's law in the solid is given as

$$\nabla \cdot (\sigma \nabla \phi_s) - a j_n \mathcal{F} = 0 \tag{5}$$

and its counterpart in the electrolyte is

$$\nabla \cdot (\kappa \nabla \phi_e) + \nabla \cdot (\kappa_D \nabla \ln c) + a j_n \mathcal{F} = 0 \tag{6}$$

In both equations we have used the divergence free property of the current field to couple the charge migration in both materials to the interchange between the two via the source term $a j_n \mathcal{F}$. Equations 5 and 6 are solved by the Ohm's law solid and electrolyte modules, respectively.

The last piece of information that is required is the transfer rate of ions from the solid to electrolyte (j_n). The rate of ion production (or consumption) is directly related to the concentration gradient of lithium ions at the surface of the solid. The production of ions can be defined in terms of a global reaction rate through the Butler–Volmer relation. We combined the two to generate a relationship between the cell potential and the ion production rate.

$$j_n = -D_s \frac{\partial c_s}{\partial r} = i_0 \left[\exp\left(\frac{\alpha_a \mathcal{F} \eta}{\mathcal{R} T} \right) - \exp\left(-\frac{\alpha_c \mathcal{F} \eta}{\mathcal{R} T} \right) \right] \tag{7}$$

$$\frac{\partial c_s}{\partial t} = \frac{1}{r^2} \frac{\partial}{\partial r} \left[r^2 D_s \frac{\partial c_s}{\partial r} \right] \tag{8}$$

The right hand side of Equation 7 forms our Butler–Volmer module. By adding this equation, we have also introduced an additional unknown, the concentration of lithium ions in the solid matrix. We can solve the time-dependent, second-order diffusion equation for ion migration (Equation 8) in the solid to close the problem. This piece defines the intercalation module.

Our system as outlined thus far forms a coupled set of non-linear partial differential equations (PDEs) in space and time for the following unknowns.

- potential in the electrolyte (ϕ_e)
- potential in the solid (ϕ_s)
- concentration of lithium ions in the electrolyte (c_e)
- concentration of lithium ions in the solid (c_s)
- transfer rate of lithium ions between the solid and electrolyte (j_n)

With the governing equations fully defined, work shifted to how to break the electrochemical device into representative pieces. Figure 2 gives a graphical representation of our electrochemical device hierarchy. In the implementation, a device level solution, denoted as a global solution space, is where the equations are solved as functions of time. The device domain is divided into arbitrary components with each component being either an electrode or separator. This allows the flexibility to arrange the components as needed and to easily add new component types (e.g., a current collector) in the future. In turn, each component is composed of several physics modules. Each physics module implements a piece of the governing equations, and no physics module is responsible for more than one equation or variable in the system.

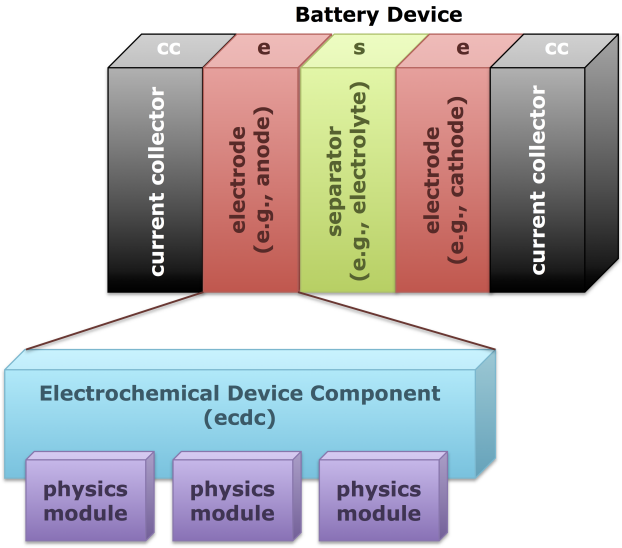

Figure 2: Schematic showing the relationship between the device, component, and module levels.

A distinct advantage of this approach is the ability to define different types of device components that solve essentially the same governing equations, but with fundamentally different material properties. Our approach allows us to arbitrarily load different types of modules depending on the functional forms of the transport equations (i.e., κ, D, t^0). We can also change the porosity (ϵ) in each component or separator by simply changing that value in the component input file. Another key benefit is the ability to configure a certain component to behave in the manner desired. For example, our separator is simply an electrode without the electron generation/migration modules enabled.

2.0.1 Boundary Conditions

Boundary conditions are handled at the module level and is done to ensure that the boundary conditions are applied in a manner that is consistent with the physics module discretization. Each module applies the appropriate Dirichlet or Neumann boundary condition, dependent on whether the user specifies a current collector or separator boundary type, respectively. We also specify a number of ghost nodes for each component that lie outside the component boundary. These nodes are filled by their neighboring components and may be modified by the component for specific boundary types (e.g., when specifying Neumann boundary conditions). When no boundary condition is specified (e.g., internal boundaries), the node solutions are determined by each module and then volume-averaged.

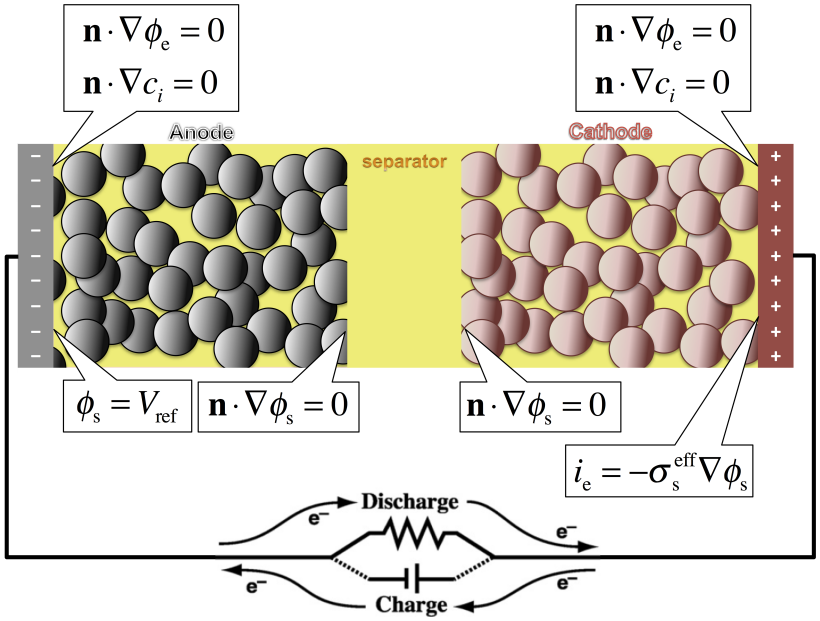

Figure 3: Section of a Li ion cell showing boundary conditions where $\vec{n}$ is the vector normal to the interface. Figure inspired by [[1]].

The expressions in Equations 3, 5, and 6 are spatially second order and thus require two spatial boundary conditions. These conditions are enforced at the edges of the components as shown in Figure 3. Where no boundary conditions are specified, the solution variables are assumed to be continuous.

At the current collector–electrode interface, the flux of each species and charge in a direction normal to the surface must be zero. This assumes that the current collector is a pure electron conductor.

$$\vec{n} \cdot \nabla c_i = 0 \tag{9}$$

$$\vec{n} \cdot \nabla \phi_e = 0 \tag{10}$$

In addition, at the anode, the solid phase electric potential is set to be the reference voltage,

$$\phi_s = V_{ref}, \tag{11}$$

while at the cathode the gradient of the solid phase electric potential (the current density) is set to be the specified current density.

$$\vec{n} \cdot \nabla \phi_s = i_{ref} \tag{12}$$

At the electrode–separator (electrolyte) interface, a similar equation holds but now the current density is specified to be zero. This assumes that the separator is entirely ion conducting.

$$\vec{n} \cdot \nabla \phi_s = 0 \tag{13}$$

The system is solved implicitly with the *NOX* package from *Trilinos*. We form a global solution vector and require that each physics module populate its local portion of a residual

function F and the Jacobian matrix $J = \frac{\partial F_i}{\partial c_j}$. Note that both the residual function and the Jacobian can have distinct values at each mesh point due to the non-linearity of the problem. Spatial derivatives are computed by a central differencing scheme for both first- and second-order derivatives. The discretization has been completed in its entirety but is not reproduced here for the sake of brevity.

2.1 Spatial discretization

The problem at hand requires the discretization of first and second derivatives. There are many options of computational stencils, but for now, we will consider a central difference scheme. We consider a non-uniform mesh spacing.

The spatial discretization for a first-order derivatives is

$$f' = \frac{f_{i+1} - f_{i-1}}{h_{i+1} + h_{i-1}} - \frac{h_{i+1} - h_{i-1}}{2} f'' + \frac{h_{i+1}^3 + h_{i-1}^3}{6\left(h_{i+1} + h_{i-1}\right)} f''' \tag{14}$$

We see that we the grid spacing is non-uniform the error is on the order of $h_{i+1} - h_{i-1}$, we will ignore the f'' and f''' error terms in our implementation. When the grid spacing is uniform, the error becomes h^2 and the familiar form is returned, again ignoring the f''' term.

$$f' = \frac{f_{i+1} - f_{i-1}}{2h} + \frac{h^2}{6} f''' \tag{15}$$

Rearranging for the second derivative yields

$$f'' = \frac{2\left(f_{i+1} + \frac{h_{i+1}}{h_{i-1}} f_{i-1}\right) - 2\left(1 + \frac{h_{i+1}}{h_{i-1}}\right) f_i -}{\left(h_{i+1}^2 + h_{i+1} h_{i-1}\right)} - \left(h_{i+1} - h_{i-1}\right) \frac{f'''}{3} \tag{16}$$

We again see that our error term is on the order of $h_{i+1} - h_{i-1}$. When the nodes are uniformly spaced, we again retain the familiar form

$$f'' = \frac{f_{i+1} - 2f_i + f_{i-1}}{h^2} \tag{17}$$

which is second order.

2.1.1 Time discretization

The software as outlined thus far forms a coupled set of non-linear PDE's in space and time. We choose to solve the system implicitly with the NOX package from *Trilinos*. We form a global solution vector and require that each physics module populate their local portion of a residual function F and the Jacobian matrix $J = \frac{\partial F_i}{\partial c_j}$. Note that both the residual function and the Jacobian can have distinct values at each mesh point due to the non-linearity of the problem.

We provide time discretizations with both the explicit forward Euler method

$$\frac{\partial c_{k,n}}{\partial t} = \frac{c_{(k+1),n} - c_{k,n}}{\Delta t} = F_{k,n} \tag{18}$$

and implicit backward Euler.

$$\frac{\partial c_{k,n}}{\partial t} = \frac{c_{(k+1),n} - c_{k,n}}{\Delta t} = F_{(k+1),n} \tag{19}$$

In the case of the forward Euler method, we can use the function evaluation implemented for the NOX nonlinear solver to find the right hand side of our system of equations at the current timestep, denoted here as $F_{k,n}$. For the backward Euler method, we use the NOX nonlinear solver to find $F_{(k+1),n}$ implicitly as a function of the solution at the next point in time ($c_{(k+1),n}$).

3 Installation

3.1 Unpack *RocSVC*

The *ICED* code is written in C++ and has been built and tested with the GNU C++ and gfortran compilers (version 4.9.1). It is assumed that the user has either received the distribution in a tarball or has checked the latest version out of the Illinois Rocstar (IR) source repository (for internal use). *ICED* uses the *CMake*-2.8 build system, which must also be installed on the host system. *CMake* will usually be present already, but if not or the *CMake* version is too old, then *CMake* can be obtained from `www.cmake.org/cmake/resources/software.html`, and easily installed in a user area. *ICED* has three third-party library dependencies: *IMPACT*, *IRAD*, and *Trilinos*. *IMPACT* and *IRAD* are tools developed by IR and are distributed with the source code. *Trilinos* is a object-oriented software framework for the solution of large-scale, complex multiphysics engineering and scientific problems. More information about obtaining and installing *Trilinos* can be found at `www.trilinos.org`. *ICED* has been developed and tested against *Trilinos*-11.12 and utilizes the *Teuchos*, *NOX*, and, *Epetra* libraries.

First, unpack the distribution. (Note that users checking the source out of the IR source repository can omit this step.) Select or create a directory for *ICED* to reside and unpack the packaged distribution in that directory, for example,

```
tar -xvf ICED.tar.gz
```

Unpacking should result in the directory tree depicted in Figure 4. Users may also find an `examples` directory containing input files for running example simulations.

update this figure to include new layout for infrastructure, Physics, and testing subdirectories

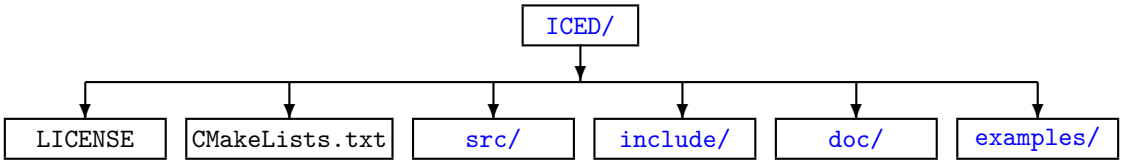

Figure 4: Directory structure for *ICED*.

It is highly recommended that *ICED* be built *out-of-source*. The following instructions assume an out-of-source build. If, for some reason, an out-of-source build is not desired, then these steps can be skipped, and the build can be conducted directly from the ICED directory.

3.2 Building *ICED*

To build *ICED* *out-of-source*, start with the following commands:

```
mkdir RocSVCBuild
cd RocSVCBuild
```

Once a build directory is created, the `Makefiles` can be generated with *CMake*. The *CMake* build system can be invoked in a few different ways on most systems. Regardless of the preferred method to run *CMake*, the target (or argument) will be the `ICED` directory. As mentioned above, these instructions assume an out-of-source build and, furthermore, will describe only the simple, non-interactive use of *CMake*.

To specify the location of the required third-party libraries, the user should specify the library path with the *CMake* variable `CMAKE_PREFIX_PATH`. Users in a C-style shell can issue a command similar to below.

```
setenv CMAKE_PREFIX_PATH <path to Trilinos>:<path to IRAD>:<path to IMPACT>
```

Note that both *IMPACT* and *IRAD* are distributed with the *ICED* software and will be built automatically as part of the install process if the directories are found at *CMake* build-time. The user will only need to specify these library locations if it is desired to use pre-installed versions located in separate directory.

The following command should create the files required to build *ICED*

```
cmake <path to ICED source>
```

The default install path will be `/usr/local`. To change the install path in the non-interactive mode of *CMake*, an additional argument is required:

```
cmake <path to RocSVC> -DCMAKE_INSTALL_PREFIX=<RocSVC install directory>
```

By default *ICED* builds in release mode (`-O3 optimization and -DNDEBUG`). To build other types of releases (i.e., debug) the users can use the following:

```
cmake <path to RocSVC> -DCMAKE_BUILD_TYPE=Debug
```

Users wishing to use a more interactive interface to the build system can replace the `> cmake` command with `> ccmake` for the *curses* interactive text menu version or `> cmake-gui` for a *Qt*-based *CMake* GUI.

Once *CMake* has successfully run and set up the `Makefiles`, then *ICED* can be built by issuing the following command:

```
make
```

and installed to `/usr/local` or the customized install path by issuing:

```
make install
```

This should build the executables and install them as

```
<install path>/bin/batterySim
```

The *RocSVC* installation is now complete.

3.3 Testing the Build

ICED comes packaged with a series of tests designed to ensure proper execution of the code.

complete this section

update this to include information about LD_LIBRARY_PATH for IMPACT testing

3.4 Building Developer Documentation

this doesn't current work from the top level. Have to traverse down into Physcs. Need to fix that.

ICED is documented with *Doxygen*. The documentation is designed to be built by *CMake* in an out-of-source build. To build the *Doxygen* documentation the user add the option `-DBUILD_DOCUMENTATION=ON` to the *CMake* build command. The documentation can then be built with the following sequence of commands:

```
cmake <cmake options> <path to source>
make
make doc
make install
```

This will build the *Doxygen* documentation and install it in <install directory>/share/doc/html. The user can access the documentation by pointing their preferred web browser to the file index.html located in this directory.

4 Pre-processing

This table was made by Rasheed. Needs to be updated for new parameters

Table 1: Solution data variables.

Term	Variable Name	Description
s_i	stoichParameter	
T	temperature	
D_s	solidDiffusionCoefficient	
ϵ	electrolyteVolumeFraction	
σ	solidConductivity	
κ_0	electrolyteConductivity	
	rateConstant	
	coulombicCapacity	
	electrolyteDensity	
	solidDensity	
	electrolyteActivityCoefficient	
	solidTransferenceNumber	
β	symmetryParameter	
c_{max}	polymerMaxConcentration	
k_c	cathodicRateConstant	
k_a	anodicRateConstant	
$\mathcal{F}$	faradayConstant	
$\mathcal{R}$	gasConstant	
v_i	velocityOfSpeciesI	
z_i	chargeNumber	
ν_1	numberOfIons	
κ	effectiveElectrolyteConductivity	
n	numberOfElectrons	
t_i^0	transferenceCoefficients	
$f_\pm$	activityCoefficients	

Continued on next page

Table 1 – continued from previous page

Term	Variable Name	Description
	openCircuitPotentialType	
	initialSaltConcentration	
	bruggemanCoefficient	
R	poreRadius	
	initialElectrolytePotential	
	initialSolidPotential	
	initialSolidLiConcentration	
	initialElectronProductionRate	
	concentrationDiffusionCoefficients	

Table 2: Device data variables.

Term	Variable Name	Description
	iterationLimit	
	transientSolver	
	dumpInterval	
	maxTime	
	dt	
	ecdc1	
	ecdc2	
	ecdc3	
	ecdc4	
	ecdc5	
	ecdc6	
	ecdc7	
	ecdc8	
	ecdc9	

Table 3: Device component data variables.

Term	Variable Name	Description
	numNodes	
	xMin	
	xMax	
	initialSaltConcentration	
	stoichParameter	
	temperature	
	solidDiffusionCoefficient	
	electrolyteVolumeFraction	
	solidConductivity	

Continued on next page

Table 3 – continued from previous page

Term	Variable Name	Description
	electrolyteConductivity	
	rateConstant	
	coulombicCapacity	
	electrolyteDensity	
	solidDensity	
	electrolyteActivityCoefficient	
	solidTransferenceNumber	
	TransferenceNumber	
	ConcentrationDiffusion	
	ActivityCoefficient	
	IonMigration	
	ionMigration	
	electronMigration	
	ButlerVolmer	
	WallFlux	
	Diffusion	
	OhmsLawElectrolyte	
	OhmsLawSolid	
	IonMigration	
	Intercalation	
	testModule	
	leftBC	
	rightBC	

Table 4: Variables present in modules.

Variable name	BV	WF	CD	IM	OLE	OLS	AC	D	I	TN
chargeNumber				✓	✓					
stoichParameter					✓					
numberOfIons		✓		✓	✓					
numberOfElectrons					✓					
faradayConstant	✓			✓	✓	✓				
gasConstant	✓			✓	✓					
temperature	✓			✓	✓					
electrolyteConductivity				✓	✓					
electrolyteVolumeFraction				✓	✓	✓		✓		
effectiveElectrolyteConductivity				✓	✓					
transferenceCoefficients		✓		✓	✓					✓
activityCoefficients				✓	✓		✓			
symmetryParameter	✓									
polymerMaxConcentration	✓									
cathodicRateConstant	✓									
anodicRateConstant	✓									
concentrationDiffusion			✓					✓		
bruggemanCoefficient								✓		
solidDiffusionCoefficient									✓	
poreRadius									✓	
solidConductivity						✓				

BV = Bulter Volmer, **WF** = Wall Flux, **CD** = Concentration Diffusion, **IM** = Ion Migration **OLE** = Ohm's Law Electrolyte, **OLS** = Ohm's Law Solid, **AC** = Activity Coefficient, **D** = Diffusion, **I** = Intercalation, **TN** = Transference Number

5 Runtime

None of this exists yet. Some verbiage from RocSVC.

this has not been updated yet.

Once the user has successfully built and installed *RocSVC* and setup an input file, it can be run from the command line. *RocSVC* requires directories for restart files and output files in the execution directory to function correctly. For the first time user running the pore collapse problem included with the distribution, it is recommended that the execution directory be setup as follows:

```
mkdir <path to run directory>
cd <path to run directory>
mkdir Output Restart
ln -s <path to RocSVC bin directory>/rocsvc .
cp <path to RocSVC example directory>/rocsvc.inp .
```

The directories Output and Restart must be created for the code to function properly. The problem can then be run with the following command:

6 Post-processing

None of this exists yet. Here is some stuff from RocSVC.

this has not been updated yet.

6.1 Output File Format

RocSVC writes ASCII output files that contain mesh data at user-specified input times. The files contain different sets of information about the run history for easy visualization. The files that are written are

- timeOut.dat – Key parameters written at a high frequency
- temperature.dat – Temperature profiles in the gas and condensate as a function of radius written at each dumpCycle
- pore.dat – Pore-specific variables as a function of pore radius written at each dumpCycle
- burnRate.dat – Mass burn rate at the pore surface as a function of time

Table 5 give the units for the output variables in each file.

Table 5: *RocSVC* output file description.

Variable Name	Description	SI units
timeOut.dat		
Time	Total simulation time	s
Pore Radius	Pore radius at each time	m
Surface Velocity	Velocity at which the pore surface is moving	$m{\cdot}s^{-1}$
Pore Pressure	Internal pore pressure (constant in the pore)	Pa
Boundary Temperature	Temperature at the pore center	K
Maximum Temperature	Maximum temperature in the mesh	K
Surface Temperature	Temperature at the gas/condensate interface	K
Omega Bar	Species consumption rate in the pore	$kg \cdot m^{-3} \cdot s^{-1}$
pore.dat (data written at each dumpCycle for every point in the pore and condensate)		
Time	Time of a particular dump	s
xi	radial coordinate in dimensionless coordinate frame	
r	radial coordinate in actual coordinate frame	m
Temperature	Temperature	K
pore.dat (data written at each dumpCycle)		
Time	Time of a particular dump	s
xi	Radial coordinate in dimensionless coordinate frame	
r	Radial coordinate in actual coordinate frame	m
Gas Velocity	Radial velocity of the gas in the pore	$m{\cdot}s^{-1}$
Species Fraction	Mass fraction of reactants in the pore	
burnRate.dat		
Time	Total simulation time	s
mcdot	Comparison with [Kang, (2002)] burn rate (unused)	
mdot	Condenstate consumption rate	$kg \cdot m^{-2} \cdot s^{-1}$

The files can easily be visualized with any number of data processing tools. An example using *gnuplot* is provided in Section **??**.

6.2 Associated Tools

Also included with *RocSVC* is a simple post-processing tool used to generate System Response Quantities (SRQ). This code is used to quickly find key information about a run,

including the minimum pore radius, time of the minimum pore radius, maximum energy release, and other selected data. To generate the SRQ, run the following command from the run directory:

```
<path to executable>/SRQGen
```

The anticipated results for the input file located in `examples` directory are shown below in block text.

```
     Time           Value
2.94600000e-08 3.62927333e-06    # Minimum pore radius (m)
2.94600000e-08 1.04170181e+08    # Qbar at minimum pore radius
3.22700000e-08 1.99249710e+08    # Maximum Qbar
3.50000000e-08 4.40986642e+09    # Maximum pore pressure (Pa)
3.50000000e-08 2.94761901e+03    # Maximum temperature (K)
2.85500000e-08 1.05243879e+03    # Maximum surface temperature (K)
```

7 Example Cases

Results from an example problem. Probably one of our V&V cases.

this has not been updated yet.

References

[1] A. M. Colclasure and R. J. Kee. Thermodynamically consistent modeling of elementary electrochemistry in lithium-ion batteries. Electrochim. Acta, 55:8960–8973, 2010.

[2] M. Doyle, T. F. Fuller, and J. Newman. Modeling of galvanostatic charge and discharge of the lithium/polymer/insertion cell. J. Electrochem. Soc., 140:1526–1533, 1993.

[3] M. Doyle, J. Newman, A. S. Gozdz, C. N. Schmutz, and J.-M. Tarascon. Comparison of modeling predictions with experimental data from plastic lithium ion cells. J. Electrochem. Soc., 143:1890–1903, 1996.

[4] T. F. Fuller, M. Doyle, and J. Newman. Simulation and optimization of the dual lithium ion insertion cell. J. Electrochem. Soc., 141:1–10, 1994.